API Reference Manual

AudioCodes WebRTC Solutions for Enterprises

WebRTC Web Browser Client

Version 1.19

WebRTC

audiocodes

Table of Contents

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 website at 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.

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()

LTRT	Description
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.
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(). • Instead 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()

LTRT	Description
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/)
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
14051	Updates for Version 1.12: ■ Added partial support of IPhone iOS Safari ■ Added setModes ringing_header_mode (to add arbitrary SIP headers to SIP 180 response) ■ Added support of SBC REGISTER 3xx response (redirect to other SBC) ■ Modified argument for methods: startSendingVideo, stopSendingVideo, sendReInvite
14052	Updates for Version 1.13: ■ Added methods to switch current SBC: getNumberOfSBC and switchSBC ■ Added screen sharing support: methods: startScreenSharing, stopScreenSharing and callback callScreenSharingEnded ■ Added argument extraOptions to methods call.answer() and phone.call()
14053	Updates for Version 1.14: ■ Using outbound DTMF for Safari. SDK detects if Safari API supported DTMF sender, for obsolete Safari versions DTMF via SIP INFO will be used. ■ Improved callback loginStateChanged() for ‘login’ set response argument to allow checking REGISTER OK response. ■ Modified screen-sharing API to use for multiple calls. • Added methods openScreenSharing(), closeScreenSharing(), isScreenSharingSupported(), isScreenSharing() and doesScreenSharingReplaceCamera() • Modified startScreenSharing(), stopScreenSharing() and usage of callback callScreenSharingEnded. ■ Modified method setWebSocketKeepAlive() ■ Added setNetworkPriority
14054	Updates for Version 1.15: ■ Added methods subscribe, notify and incomingSubscribe callback. ■ Added Automatic Call Distribution (ACD) support. Files broadsoft_acd.js and pjxml.js.
14055	Updates for Version 1.16: ■ JsSIP fix to support auth-int authentication ■ Improved keep alive ping/pong algorithm: received pong reset timer. Modified setWebSocketKeepAlive(): argument timerThrottlingBestEffort=true not increase pong interval.

LTRT	Description
14056	Updates for Version 1.17: ■ Fixed setJsSipLogger ■ Added setModes setting: sbc_switch_register5xx_mode ■ Added setModes setting: cache_register_auth_mode ■ Added setModes setting: check_remote_sdp_mode ■ Added Citrix phone example
14057	Updates for Version 1.18: ■ Added setCodecFilter method ■ Modified setBrowsersConstraints, setConstraints methods ■ Removed obsolete setChromeAudioConstraints method, use setConstraints('chrome', 'audio', obj) instead. ■ Added setConstraint() method. ■ Added SelectDevices class to file utils.js ■ Added device selection (microphone, camera, speaker, ringer) to a single call phone prototype, multi-call phone prototype and Citrix phones.
14058	Updates for Version 1.19: ■ Added a list of causes for call Terminated callback. ■ Added Dual Registration Phone prototype and the backup_sbc.js module. ■ Added click-to-call options: to use keypad panel to send DTMF during call and optional checkbox to view local video.
14059	URL to demo updated

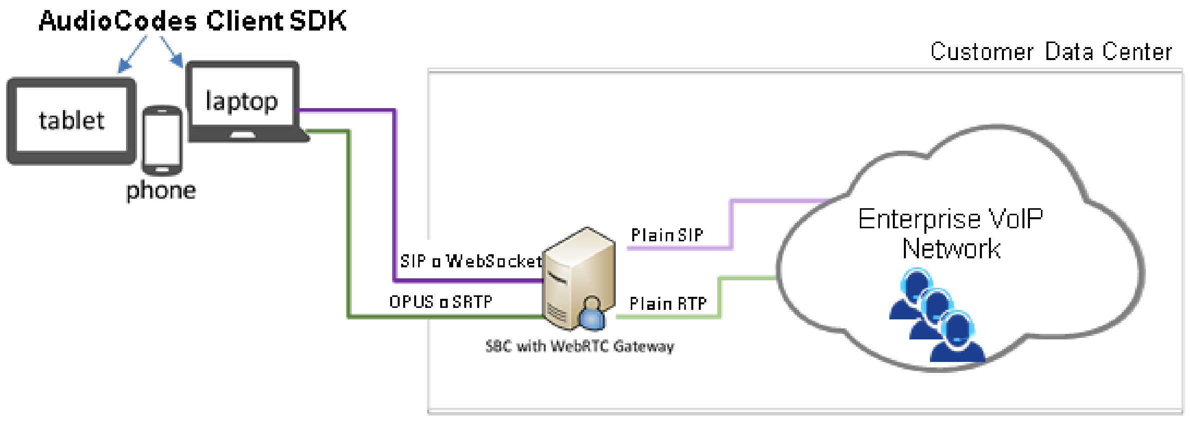
Documentation Feedback

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 website 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.



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. Refer to the *WebRTC Widget Installation and Configuration Guide*.

1.1 Purpose

This *Reference Guide* defines the 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

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  callTo,
extraHeaders=null, extraOptions=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);
void setConstraint(String type, String constraint-name, Object
constraint);
setBrowsersConstraints(Object object);
void setAcLogger(Function logger);
void setJsSipLogger(Function logger);
void setWebSocketKeepAlive(int pingInterval, Boolean
pongTimeout=true, Boolean timerThrottlingBestEffort=true, int
pongReport=0, Boolean pongDistribution=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)
int  getNumberOfSBC()
void switchSBC()
Promise openScreenSharing()
void closeScreenSharing(stream)
Boolean isScreenSharingSupported()
void setNetworkPriority(String priority)
Subscriber subscribe(String target, String eventName,
  String accept, SubscriberOption option)
Notifier notify(IncomingRequest subscribe, String contentType,
  Notifier option)
}
```

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.

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: Defines if the call must be initiated with video, or phone.AUDIO
- callTo [string]: Defines a string of the destination address/number

Return Values

- A call session object is defined here.

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.

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.
- callTo [string]: Destination address/number
- extraHeaders (Optional). An array of strings, each representing a SIP header, to be added to the INVITE request.
- extraOptions (Optional). The object sets options to the internal method jssip.call.
  - Can be used to set custom media streams for created calls.

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 setConstraints

> 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 [string]:
  - null – current browser.
  - browser name, one of ‘chrome’, ‘firefox’, ‘safari’, ‘other’.
  - browser name, vertical line, os name:
  - ‘chrome|windows’, ‘chrome|macos’, …

  The OS name refers to one of the following: ‘windows’, ‘android’, ‘macos’, ‘ios’, ‘linux’, ‘other’.
  During SDK initialization, it gets the current browser name, and the OS name, (which can be obtained by phone.getBrowser() and phone.getOS()).

  When called, the setConstraints SDK checks what is the used browser name or checks that the browser-os-name is the name of the current browser and OS. Otherwise, the method is ignored.
- 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(null, '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(null, '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

2.1.2.8 setConstraint

Adds or removes a single audio or video constraint without modifying other constraints. This can be used to set the deviceId constraint.

Parameters

- type [string] “audio” or “video”
- constraint name [string]. E.g. “deviceId”
- constraints [object]
  - e.g., “device-id-string” or { exact: “device-id-string” }
  - null value used to remove the constraint.

Return Values

n/a

2.1.2.9 setBrowsersConstraints

Used to set constraints for all browsers. Internally used setConstraints function is used with its limitations.

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 }
    },
    safari: {
        audio: { echoCancellation: true }
    },
    "safari|ios": {
        audio: { echoCancellation: true }
    },
}
```

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

The browser name can be used with the os name: 'chrome|windows', 'safari|ios', 'safari|macos'. Possible OS names are 'windows', 'android', 'linux', 'macos', 'ios', 'other'.

For each browser or browser-os entry, the setConstraints method is executed.

If the browser name or browser name/os name does not match the name of the current browser, the entry is ignored.

If it matches two entries, the first entry is replaced by the second entry. In the example, 'safari' and 'safari|ios' entries are defined. If the current browser is Safari in iOS, constraints from 'safari' entry are set. They are replaced by constraints defined in 'safari|ios' entry.

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 JsSIP for logging. By default, the console log is used.

Parameter

- User-defined Logger function name.

Return Values

n/a

2.1.2.13 setWebSocketKeepAlive

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

Purpose

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

Refresh NAT connection.

Use

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

Description

Sends a ping to the server and expects a ping response before the next ping is sent.

Detects intensive timer throttling (it can be applied if the page is hidden).

If timerThrottlingBestEffort is enabled (value != false or object interval > 0), increase the ping interval to exit from intensive timer throttling mode.

Parameters

- **pingInterval** ping interval in seconds [Number]
  - 0: Do not send pings
  - >0: Ping interval
- **pongTimeout** [Boolean]
  - true (default) Close WebSocket after ping timeout
  - false: Log warning and do not close WebSocket after ping timeout
- **timerThrottlingBestEffort** [Boolean or Object]
  - **true**: (default)
    For Chrome: detects timer throttling and does not increase ping interval.

Since SDK 1.16, the ping/pong algorithm improved – now receives pong reset timer interval.

No need to increase ping interval when timer throttling is detected as was necessary in previous version.

If programmer wants to use SDK 1.15 algorithm, then instead argument parameter value "true", set object with interval value 62. In this case after timer throttling detection the ping interval will be increased.

Check when the page is visible or there is an active call; in such cases, restore the original value of the timer interval.

For Firefox and Safari, a warning is displayed and no action is taken.

- **false**: A warning is displayed if timer throttling is detected. No action is taken.
- Object with the format:

```
{   log: 0,
    chrome: {
      interval: 1,
      visibility: true,
      call: true,
      log: 1
    }
}
```

For normal usage, set the value for timerThrottlingBestEffort: true or false.

For debugging or fine tuning, use the object.

The object {log: 0, chrome: { interval: 1, visibility: true, call: true, log: 1 }}

is used internally if timerThrottlingBestEffort is set to true

The object **{ log: 0 }**

is used internally if timerThrottlingBestEffort is set to false

Can be used with other values:

To check how Safari uses intense throttling, set

{   log: 1, chrome: { interval: 1, visibility: true, call: true} }

In this case, each big (> 10 seconds) ping timer deviation will be displayed.

In Safari 14.0.3, the deviation is random and less than 60 seconds, but in future versions it may change.

When the used timerThrottlingBestEffort=true or false for a browser that is different to

Chrome, the log level is 0. This means that the detected timer throttling warning will be displayed only once.

Another example: For debugging the display timer deviation, in milliseconds, for each ping sent:

{   log: 3, chrome: { interval: 1, visibility: true, call: true}}

log: [integer] log level for all browsers.

0 changed ping interval, detected timer throttling displayed only once.

1 changed ping interval and each detected timer throttling.

2 all relative events (changed page visibility or call started/terminated)

3 each ping shows the timer deviation, in milliseconds

Levels 2 and 3 for debugging.

The parameter **log** can be set also for the browser setting if it overloaded the value.

Browser names and actions.

Can be seen that defined action only for Chrome.

interval: [integer] if this value is not 0 and is greater than the current ping interval, then the value is will increased ping interval after timer throttling is detected.

To allow Chrome exit from intensive timer throttling mode, the interval must be more than 60 seconds (in SDK 1.16, there is no need increases the interval to exit the mode) If programmer want use compatible with SDK 1.15 setting, he can set the interval 62 seconds.

visibility: [Boolean] check page visibility:

when page is visible - use original ping interval value.

when page is hidden - use increased ping interval value provided that timer throttling was previously detected.

call: [Boolean] Check if there is an active call.

If there is active call use original ping interval value.

log: [integer] optional.

Log value for the browser, if defined overload value for all browsers.

- **pongReport** Printing interval [Number]: 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
  - >0: Print report every number of pongs
- **pongDistribution** The ping delay distribution with time step 0.25 seconds was added to the logs. [Boolean]
  - true: Add to ping report interval distribution.
  - false: (default) No print ping 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.

- 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
```

- **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.

- **ringing_header_mode** Default value: undefined

Add additional SIP header(s) to ringing response (SIP response code 180)

For example, add Allow-Events SIP header:

```
ringing_header_mode: 'Allow-Events: talk,hold,conference'
```

Multiple SIP headers can be added, for example

```
ringing_header_mode: ['Allow-Events: talk,hold,conference',
'X-Greeting: Have a nice day !']
```

- **sbc_switch_register5xx_mode** Default value: true

If SBC send REGISTER response 5xx and in phone configuration set more than one SBC URL,

switch websocket connection to the next SBC.

- **cache_register_auth_mode** Default value: true
- The next SIP REGISTER message includes an “Authorization” header from the previous one.
- REGISTER with “Expires: 0” clears the authorization header cache.
- **check_remote_sdp_mode** Default value: true

Checks the remote SDP compatibility with the one used in the previous SDP negotiation.

For cases where:

- The same payload type and rtpmap codec name are associated with different “fmtp” values.
- The same payload type are used for different “rtpmap” codec names.

To find the warnings in the print log, search for ‘AC:SDP’.

Parameter

- modes object

Return Values

n/a

2.1.2.27 getNumberOfSBC

Returns the number of SBC Server Addresses (set by setServerConfig serverAddresses).

Parameters

n/a

Return Values

- The number of SBC servers set in server configuration [integer].

2.1.2.28 switchSBC

Disconnects from the current SBC and switches to the other SBC.

- You should have more than one SBC to use the method.
- Before switching SBC, check that you have no open calls on the current SBC.

Parameters

n/a

Return Values

n/a

2.1.2.29 openScreenSharing

Returns the Promise object resolved with a screen sharing video stream.

Parameters

n/a

Return Values

- The Promise object is resolved with a video stream value and is rejected with a string describing the problem (screen sharing is not supported or disabled by user).

2.1.2.30 closeScreenSharing

Returns the number of SBC Server Addresses (set by setServerConfig serverAddresses).

Parameters

- Video stream object created previously by method openScreenSharing()

Return Values

n/a

2.1.2.31 isScreenSharingSupported

Checks if screen sharing API is supported in the current browser.

Parameter

n/a

Return Values

- Boolean

2.1.2.32 setNetworkPriority

Changes the sending RTP packets IP DSCP field.

- The method works only for Chrome browser and ignored by others.

Parameters

- priority [string or undefined]

  Allowed value: undefined, 'high', 'medium', 'low', 'very-low'

  If method don't used or set undefined value – DSCP don't changed.

2.1.2.33 subscribe

Uses JsSIP subscribe extension to create SIP SUBSCRIBE dialog.

Parameters

- target [string]: send SUBSCRIBE request to the target.
- eventName [string]: event name
- accept [string]: header value
- subscriberOption [object]: All properties are optional.
    - expires [number]: SUBSCRIBE expires. Default is 900
    - contentType [string]: Content-Type header value. Used for SUBSCRIBE with body
    - allowEvents [string]: Allow-Events header value.
    - params [RequestParams] If set, define: to_uri, to_display_name,
    - from_uri, from_display_name
    - extraHeaders [array of strings]: adds extra SIP headers to SUBSCRIBE.

Return Values

- Subscriber class instance.

2.1.2.34 notify

Use JsSIP notify extension to create SIP NOTIFY dialog.

Parameters

- subscribe [IncomingRequest]: subscribe incoming request.
- contentType [string]: Content-Type header value. Used for NOTIFY
- option [NotifierOption] All properties are optional.
    - pending [boolean]: Create notifier in 'pending' state
    - Default is false – notifier created in 'active' state.
    - allowEvents [string]: Allow-Events header value.
    - extraHeaders [array of strings]: adds extra SIP headers to NOTIFY.

Return Values

- Notifier class instance.

2.1.2.35 setCodecFilter

Modifies the codec priority and removes codecs.

This function works with Chrome and Safari browsers. It is not supported with the Firefox browser. This function should be used once to set all audio and video filters.

Parameters

- Object with audio and/or video part.
- Each part can be used as a priority list or remove list.
- The codec can be set as a name - e.g., **‘pcma’** (case insensitive).
- The name can be set with a frequency - e.g., **‘pcma/8000’**.
- The name, optional frequency and Media Format Parameter Capability" (fmtp) can be set - e.g., **'VP9/90000#profile-id=0'** or **'VP9#profile-id=0'.**

Return Values

n/a

Usage examples:

- **Changing codec priorities:**

  Codec priorities are defined in SDP in m=audio and m=video lines.

- **Changing audio codec priorities:**

  By default, Chrome sends INVITE with the SDP.

```
m=audio 59963 UDP/TLS/RTP/SAVPF 111 63 103 104 9 0 8 106 105
13 110 112 113 126
111 is OPUS payload type. It’s the first, so most preferrable.
```

To configure:

```
phone.setCodecFilter({
  audio: { priority: ['pcmu', 'pcma'] }
});


m=audio 55641 UDP/TLS/RTP/SAVPF 0 8 111 63 103 104 9 106 105
13 110 112 113 126
Codec priorities have changed - the first payload type is 0
(pcmu) and the next is 8 (pcma):

The same can be seen from the phone console log:
```

AC: audio codec-filter original: ["opus/48000#minptime=10;useinbandfec=1","red/48000#111/111","isac/16000","isac/32000","g722/8000","pcmu/8000","pcma/8000","cn/32000","cn/16000","cn/8000","telephone-event/48000","telephone-event/32000","telephone-event/16000","telephone-event/8000"]

```
AC: audio codec-filter changed priority:
["pcmu/8000","pcma/8000","opus/48000#minptime=10;useinbandfec=1","red/48000#111/111","isac/16000","i
sac/32000","g722/8000","cn/32000","cn/16000","cn/8000","telephone-event/48000","telephone-
event/32000","telephone-event/16000","telephone-event/8000"]

Here RED is redundant coding to recover packets lost under
difficult network conditions. Its priority is not
important, so no need to change it.

CN - Comfort Noise codec is used for different frequencies.
Telephone event (to send DTMF) for different frequencies.
```

- **Changing video codec priorities**:

  By default:

  m=video 53284 UDP/TLS/RTP/SAVPF 96 97 98 99 100 101 127 121 125 107 108 109 124 120 123 119 35 36 41 42 114 115 116 117 118

  The default codec priorities are: VP8, VP9 with different profiles, H.264 with different profiles and AV1.

```
phone.setCodecFilter({
  video: { priority: ['av1', 'vp9', 'vp8'] }
});
```

  After the codec filter offer, SDP will be:

  m=video 57619 UDP/TLS/RTP/SAVPF 41 42 98 99 100 101 96 97 127 121 125 107 108 109 124 120 123 119 35 36 114 115 116 117 118

  41 is the payload type of the AV1 video codec.

  You can also see in the phone log, the original codec priorities as well as the priorities after they change:

```
AC: video codec-filter original: ["vp8/90000","rtx/90000","vp9/90000#profile-id=0","vp9/90000#profile-
id=2","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=42001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=42001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=42e01f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=42e01f","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=4d001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=4d001f","av1/90000","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=64001f","red/90000","ulpfec/90000"]

AC: video codec-filter changed priority: ["av1/90000","vp9/90000#profile-id=0","vp9/90000#profile-
id=2","vp8/90000","rtx/90000","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=42001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=42001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=42e01f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=42e01f","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=4d001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=0;profile-level-
id=4d001f","h264/90000#level-asymmetry-allowed=1;packetization-mode=1;profile-level-
id=64001f","red/90000","ulpfec/90000"]
```

  The video codecs include:

  - VP8
  - VP9
  - H.264
  - AV1

  and special codecs used as additions to normal video codecs:

  - RED for redundancy
  - RTX for retransmission
  - ULPFEC forward error correction

The priority of the codecs is ignored as there is no need to use them in the priority list.

The codecs can be used in the remove list.

- **Removing Codecs:**

> - It’s not recommended to remove codecs.
> - It is better to keep all browser-provided codecs to ensure compatibility with different browsers and operating systems.
> - If you remove some codecs, there is a chance that the phone that received the call will not be able to use the audio or video codec you offer.
> - For example, if you remove all video codecs except the most modern - AV1, you won't be able to use video if one side is Windows Chrome and the other side is iPhone Safari, Firefox or not the latest version of Android Chrome.

- **To remove some audio codecs and change priority:**

```
   phone.setCodecFilter({
    audio: {
        remove: ['isac', 'g722'],
        priority: ['pcma']
    }
 });
```

Filtering the result can be seen in console log:

```
AC: audio codec-filter original:
["opus/48000#minptime=10;useinbandfec=1","red/48000#111/111
","isac/16000","isac/32000","g722/8000","pcmu/8000","pcma/8
000","cn/32000","cn/16000","cn/8000","telephone-
event/48000","telephone-event/32000","telephone-
event/16000","telephone-event/8000"]

AC: audio codec-filter remaining:
["opus/48000#minptime=10;useinbandfec=1","red/48000#111/111
","pcmu/8000","pcma/8000","cn/32000","cn/16000","cn/8000","
telephone-event/48000","telephone-event/32000","telephone-
event/16000","telephone-event/8000"]

AC: audio codec-filter changed priority:
["pcma/8000","opus/48000#minptime=10;useinbandfec=1","red/4
8000#111/111","pcmu/8000","cn/32000","cn/16000","cn/8000","
telephone-event/48000","telephone-event/32000","telephone-
event/16000","telephone-event/8000"]
```

- To remove some video codecs and change priorities:

```
 phone.setCodecFilter({
     video: {
            remove: ['h264', 'vp9#profile-id=2', 'av1',
'ulpfec'],
            priority: ['vp9', 'vp8']
     }
 });
```

'RTX' and 'RED' can be removed if needed. The Mediant SBC has modes to support the RTX retransmission and RED redundancy. Do not remove the codecs.

Filtering result can be seen in console log:

```
AC: video codec-filter original:
["vp8/90000","rtx/90000","vp9/90000#profile-
id=0","vp9/90000#profile-id=2","h264/90000#level-asymmetry-
allowed=1;packetization-mode=1;profile-level-
id=42001f","h264/90000#level-asymmetry-allowed=1;packetization-
mode=0;profile-level-id=42001f","h264/90000#level-asymmetry-
allowed=1;packetization-mode=1;profile-level-
id=42e01f","h264/90000#level-asymmetry-allowed=1;packetization-
mode=0;profile-level-id=42e01f","h264/90000#level-asymmetry-
allowed=1;packetization-mode=1;profile-level-
id=4d001f","h264/90000#level-asymmetry-allowed=1;packetization-
mode=0;profile-level-id=4d001f","av1/90000","h264/90000#level-
asymmetry-allowed=1;packetization-mode=1;profile-level-
id=64001f","red/90000","ulpfec/90000"]

AC: video codec-filter remaining:
["vp8/90000","rtx/90000","vp9/90000#profile-id=0","red/90000"]

AC: video codec-filter changed priority: ["vp9/90000#profile-
id=0","vp8/90000","rtx/90000","red/90000"]
```

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, extraHeaders=null,
extraOptions=null);
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(options = {})
Promise stopSendingVideo(options = {})
Boolean hasVideo()
Boolean hasSendVideo()
Boolean hasReceiveVideo()
string getVideoState()
Boolean hasEnabledSendVideo()
Boolean hasEnabledReceiveVideo()
string getEnabledVideoState()
void setRemoteHoldState()
RTCPeerConnection getRTCPeerConnection()
MediaStream getRTCLocalStream()
MediaStream getRTCRemoteStream()
Promise sendReInvite(options = {})
Boolean wasAccepted()
void sendInfo(body, contentType, extraHeaders=null)
Promise startScreenSharing(stream)
void    stopScreenSharing()
Boolean isScreenSharing()
Boolean doesScreenSharingReplaceCamera()
}
```

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

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

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

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

Return Values

n/a

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 isAudioMuted

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 isVideoMuted

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.

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.
- extraOptions (Optional): The object set options to internal method jssip answer.

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

- CallTo [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.

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.

> 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

- Object options. Default value is {}
  - options.enableReceiveVideo [Boolean] by default is 'true'.
    The argument sets the internal call flag hasEnabledReceiveVideo().
  - options.extraHeaders [Array of String] by default undefined.
    Adds extra SIP headers to re-INVITE.

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

- Object options. Default value is {}
  options.extraHeaders [Array of String] by default undefined.
  Adds extra SIP headers to re-INVITE.

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’)

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

- Object options. Default value is {}
  - options.showStreams [Boolean] by default is 'false'.
    Call callShowStreams callback after re INVITE transaction is completed.
    Used to assign stream value to the srcObject HTML video element.
  - options.extraHeaders [Array of String] by default is undefined.
    Adds extra SIP headers to re-INVITE.

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

2.2.2.18 startScreenSharing

For audio or video calls, this method starts sending screen sharing video track to the other side.

For audio call will be added screen sharing video track.

For video call web camera video track will be replaced for screen sharing video track.

Set internal call flag setEnabledSendVideo()

Parameters

- Screen sharing video stream opened by openScreenSharing()

Return Values

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

2.2.2.19 stopScreenSharing

Stop sending screen sharing video track.

For video call will be restored previously sent web camera video track.

Parameters

n/a

Return Values

n/a

2.2.2.20 isScreenSharing

Checks if the call is currently sending a screen-sharing video stream.

Parameters

n/a

Return Values

- Boolean

2.2.2.21 doesScreenSharingReplaceCamera

Checks if the call is sending a screen-sharing video stream and if it replaced the stream previously sent from the camera. The method used to restore the call after page reloading.

Parameters

n/a

Return Values

- Boolean

2.3 Subscriber

It is AudioCodes JsSIP extension that implements RFC 6665. The class instance is created by the phone.subscribe() method.

Refer to the API usage examples in the SDK:

- single call phone prototype (file phone.js)
- phone prototype with ACD (file broadsoft_acd.js)

Syntax

```
class Subscriber {
  int state
  String id
  void subscribe(String body = null)
  void terminate(String body = null)
  void on(String event, Function callback)
  void removeAllListeners(String event)
}
```

2.3.1 get state

Gets the subscriber dialog state.

Return Values

- number

Possible values are:

- subscriber.C.INIT
- subscriber.C.STATE_NOTIFY_WAIT
- subscriber.C.STATE_PENDING
- subscriber.C.STATE_ACTIVE
- subscriber.C.STATE_TERMINATED

2.3.2 get id

Gets the unique subscriber dialog id.

Return Values

- String

2.3.3 subscribe

Sends the SIP SUBSCRIBE request to the server.

Parameter

- body [String or null value]
- The value will be send as SUBSCRIBE request body.

Return Values

n/a

2.3.4 terminate

Sends the SIP un-SUBSCRIBE (subscribe with header Expires: 0) request to server.

Parameter

- body [String or null value]

  The value will be sent as SUBSCRIBE request body.

Return Values

n/a

2.3.5 on

The Subscriber class extends the EventEmitter class that provide the on() method.

This is used to set the callback function to subscriber events.

Parameter

- eventName [String]
  - ‘dialogCreated’ fired when the subscriber receives the OK response to the initial SUBSCRIBE message. After the event, subsequent queued SUBSCRIBE requests (if they exist) will be sent.

    Callback function arguments: N/A
  - ‘pending’ fired when the subscriber state has changed to ‘pending’ state.

    (received 1st NOTIFY with header Subscription-State: pending)

    Callback function arguments: N/A
  - ‘active’ fired when the subscriber state has changed to ‘active’ state.

    (received 1st NOTIFY with header Subscription-State: active)

    Callback function arguments: N/A
  - ‘notify’ fired when the NOTIFY request has been received with no empty body.

    Callback function arguments:
    - isFinal [Boolean]. Is received NOTIFY with Subscription-State: terminated
    - notify [IncomingRequest]. Received NOTIFY request.
    - body [String] NOTIFY request body
    - contentType [String]. Value of Content-Type header.
  - ‘terminated’ fired when the subscriber has terminated.

    Callback function argument:

        - terminationCode [Number].

          Possible values are:

            - subscriber.C.SUBSCRIBE_RESPONSE_TIMEOUT
            - subscriber.C.SUBSCRIBE_TRANSPORT_ERROR
            - subscriber.C.SUBSCRIBE_NON_OK_RESPONSE
            - subscriber.C.SUBSCRIBE_BAD_OK_RESPONSE
            - subscriber.C.SUBSCRIBE_FAILED_AUTHENTICATION
            - subscriber.C.UNSUBSCRIBE_TIMEOUT
            - subscriber.C.RECEIVE_FINAL_NOTIFY
            - subscriber.C.RECEIVE_BAD_NOTIFY
    - reason [String or undefined] termination reason.

      Defined when subscription termination reason is incoming final. NOTIFY and header Subscription-State contains reason parameter.
    - retryAfter [Number or undefined]

      Defined when subscription termination reason is incoming final. NOTIFY and header Subscription-State contains retryAfter parameter.
- callback [Function]

  Function with corresponding to event name arguments

Return Values

n/a

2.3.6 removeAllListeners

The Subscriber class extends the EventEmitter class that provides the removeAllListeners() method.

Remove event listener function set by on() method.

Parameter

- eventName [String] Defines the event name.

Return Values

n/a

2.4 Notifier

It is AudioCodes JsSIP extension that implements RFC 6665. The class instance is created by the phone.notify() method. See the API usage example in the SDK single call phone prototype (file phone.js).

Syntax

```
class Notifier {
  int state
  String id
  void start()
  void setActiveState()
  void notify(String body = null)
  void terminate(String body = null, String reason = null,
    int retryAfter = null)
  void on(String event, Function callback)
  void removeAllListeners(String event)
}
```

2.4.1 get state

Gets the notifier dialog state.

Return Values

- number

  Possible values are:

  - notifer.C.STATE_PENDING
  - notifier.C.STATE_ACTIVE
  - notifier.STATE_TERMINATED

2.4.2 get id

Gets the unique notifier dialog id.

Return Values

- String

2.4.3 start

Allows the call after creating the Notifier instance, and then setting the event handlers.

Starts processing the initial SUBSCRIBE.

Parameters

n/a

Return Values

n/a

2.4.4 setActiveState

Switches notifier state from ‘pending’ to ‘active’ state.

Parameter

n/a

Return Values

n/a

2.4.5 notify

Sends the SIP NOTIFY request to the subscriber.

Used after receiving SUBSCRIBE or the system state has changed.

Parameter

- body [String or null value]
  The value will be sent as NOTIFY request body.

Return Values

n/a

2.4.6 terminate

Sends the final SIP NOTIFY request to the subscriber. This is used after receiving an un-SUBSCRIBE, subscription time has expired or the

notifier wants to terminate the subscription.

Parameter

- body [String or null value]
- The value will be sent as final NOTIFY request body.

Return Values

n/a

2.4.7 on

The Notifier class extends the EventEmitter class which provides the on() method.

This is used to set the callback function to notifier events.

Parameter

- eventName [String]
  - ‘subscribe’ fired when received subscribe message (includes initial subscribe)
    Callback function arguments:
    - is_unsubscribe [Boolean] if received un-subscribe (with expires: 0)
    - request [IncomingRequest] received subscribe
    - body [String] optional subscribe body
    - contentType [String] optional body content types
  - ‘terminated’ fired when notifier is terminated.

Callback function argument:

- terminationCode [Number].
  Possible values are:
  notifier.C.NOTIFY_RESPONSE_TIMEOUT
  notifier.C.NOTIFY_TRANSPORT_ERROR
  notifier.C.NOTIFY_NON_OK_RESPONSE
  notifier.C.NOTIFY_FAILED_AUTHENTICATION
  notifier.C.SEND_FINAL_NOTIFY
  notifier.C.RECEIVE_UNSUBSCRIBE
  notifier.C.SUBSCRIPTION_EXPIRED

- callback [Function]: function that will be called when the event fired.

Return Values

n/a

2.4.8 removeAllListeners

The Notifier class extends the EventEmitter class which provides the removeAllListeners() method.

This removes the event listener function set by the on() method.

Parameter

- eventName [String] The event name.

Return Values

n/a

2.5 BroadsoftAcdAgent

The Automatic Call Distributor (ACD) is a device or system that distributes incoming calls to a specific group of terminals that agents use.

Genesys server uses an ACD system built on the basis of the Broadsoft ACD specification.

To use the API, add the following to the project files:

- broadsoft_acd.js (classes to work with ACD SUBSCRIBE/NOTIFY)
- pjxml.js (open source tiny XML parser)

Take the files from the SDK phone prototype with ACD. The phone prototype with ACD demonstrates the use of the API.

Syntax

```
class BroadsoftAcdAgent {
  constructor()
  void setListeners(Function stateChanged, Function errorListener)
  void setLog(Function logger)
  void setAccount(String acdUser, String acdPassword, int expires)
  void setLoginState(String state, int substate)
  void start()
  void stop()
  void logon()
  void logoff()
  void setState(String state, int substate)
}
```

2.5.1 constructor

Creates the object instance.

2.5.2 setListeners

Defines the listeners.

Parameter

- state listener [Function state(String state, int or undefined code]
- error listener [Function error(String message, Boolean isSubscriptionTerminated)

Return Values

n/a

2.5.3 setLog

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

2.5.4 setAccount

Defines the account details.

Parameters

- acdUser [string]: acd user name. Used as hoteling guest address.
- password [string]: authenticating user password
- expires [int]: set subscription expires (seconds)

Return Values

n/a

2.5.5 setLoginState

Sets the state that will be set during login. It should be used before the call logon() method.

Parameters

- state [string]: set acd agent state. Can be used values: 'ready', 'notReady', 'workingAfterCall'
- substateCode [int or undefined]: code for 'notReady' state. Positive number

Return Values

n/a

2.5.6 start

Performs subscription to ACD service.

Parameters

n/a

Return Values

n/a

2.5.7 stop

Performs un-subscription to the ACD service.

Parameters

n/a

Return Values

n/a

2.5.8 logon

Performs the logon on the ACD account.

Parameters

n/a

Return Values

n/a

2.5.9 logoff

Performs logoff on the ACD account.

Parameters

n/a

Return Values

n/a

2.5.10 setState

Sets the ACD state.

Parameters

- state [string]: set ACD agent state.
- Values that can be used: ‘ready’, ‘notReady’, ‘workingAfterCall’
- substateCode [int or undefined]: code for ‘notReady’ state.
- Positive number: 0, 1, 2, …

Return Values

n/a

2.6 Citrix desktop phone

Citrix provides WebRTC Redirection SDK. The SDK can be used to build Electron applications or browser single page applications (SPA). If you use Citrix desktop, see: citrix.com.

Browser SPA only supports audio calls.

2.6.1 Citrix SDK conversion

The Citrix SDK file CitrixWebRTC.js (or CitrixWebRTC.min.js) is a Node module. To use it in the browser, it must be converted.

- Use the latest Citrix SDK release.
- The file CitrixWebRTC.js used in this example was taken from the SDK beta release and could become obsolete.

```
Npm install –global browserify
browserify CitrixWebRTC.js –outfile browserifyCitrixWebRTC.js
```

2.6.2 JsSIP modification

Citrix API is very close to standard WebRTC API, but not 100% compatible. Therefore, Citrix API JsSIP must be modified.

> We replaced a few WebRTC API functions to Citrix analogs. Review python script source to see replaced functions.

To convert AudioCodes SDK:

1. Install Python3.
2. Run the following script:

```
py citrix_convert.py <ac_webrtc.min.js >citrix_ac_webrtc.min.js
```

3. For debugging, you can replace the obfuscated *ac_webrtc.min.js* file with the non-obfuscated files *acapi.1.?.0.js* and *citrix_jssip.js*

To build citrix_jssip.js:

- Run the following script:

```
py citrix_convert.py <jssip.js >citrix_jssip.js
```

2.6.3 Citrix cloud windows configuration

To enable the Citrix SDK:

1. You must set the registry in the remote Citrix Windows system. Edit Windows registry (use regedit.exe command).
2. Enable Citrix redirection.

```
Key Path: HKCU\Software\Citrix\HDXMediaStream
Key Name: MSTeamsRedirSupport
Key Type: DWORD
Key Value: 1
Add the Chrome program to the allow list.
Key Path: HKLM\Software\WOW6432Node\Citrix\WebSocketService
Key Name: ProcessWhitelist
Key Type: MULTISZ
Key Value: chrome.exe
```

3. Configure microphone privacy settings.

   In Citrix Desktop Windows, open "microphone privacy setting" and enable microphone usage.

2.6.4 Modification Simple Phone Prototype

The provided Citrix phone prototype is a modified simple phone prototype. Compare the simple phone prototype code with this version.

- Support for video calls removed.
- Citrix API in phone.js was used.

> The phone.js and citrix_jssip.js does not call the Citrix API directly, but via the citrix_adapter.js wrapper.

2.6.5 How Citrix Phone Starts

1. The phone waits for the initialization of the Citrix SDK.

> There will be an error, if the browser did not start on the Citrix desktop or the desktop is not configured.

2. The phone uses the Citrix API to collect available microphones and speakers.
3. The phone selects the microphone and speaker.

> The Citrix API does not work without selected devices:
> - getUserMedia requires microphone deviceId constraints.
> - AudioElement requires speaker deviceId to be assigned to sinkId.

4. The phone attempts to use the same microphone and speaker that were selected before.

> This does not always work, for example:
> - When the user detached the USB headset that was previously used.
> - When the phone starts up for the first time.
>
> In such cases, the settings screen opens, allowing the user to select the microphone and speaker.

5. The phone starts JsSIP stack and works in the same way as other phone examples.

2.7 Dual Registration Phone

The JsSIP stack with the AudioCodes CRLF keep-alive extension, quickly detects (10..20 seconds) WebSocket disconnection of the SBC, after it reconnects with the same or other SBC. However, there is a need for the phone to simultaneously open two WebSocket connections with the main and backup SBC. JsSIP is well maintained and has reliable code.

The problem is that JsSIP stack can only work with one WebSocket. Making such drastic changes to it, would be difficult to test and can significantly reduce its reliability. Customers do not want a phone that can simultaneously call through the main and backup SBC. In this situation, only one SBC (the one to which the JsSIP is connected) is active.

Without changing JsSIP, an optional backup SBC module (backup_sbc.js) will be added to connect to the backup SBC and sending a SIP REGISTER sequence. If necessary, the two WebSockets (JsSIP WebSocket and backup SBC WebSocket) can be swapped.

2.7.1 Backup SBC Module Functionality Description

- The main and backup SBC use the same account credentials (user, password, realm, domain name).
- The backup SBC module does not implement the complete SIP protocol. It supports the following:
  - Sending REGISTER sequences and REGISTER authentication
  - Receiving or rejecting INVITEs (send 486 response)
  - Swapping main/backup transport (to answer incoming call in JsSIP stack)
  - Sending keep-alive CRLF ping/pong

- If the registration on the backup SBC transport fails, no actions will be performed beside reconnection attempts.
- If the main transport fails, the backup WebSocket transport can be swapped to the main WebSocket transport. The phone code detects the case (as sequence of login disconnect events) and swaps the main/backup transport (if backup SBC is registered).
- If you receive an INVITE on the backup WebSocket, the transport is called incomingInvite() callback. The phone code checks if there are open calls in the main JsSIP transport. If open calls exist in the main JsSIP transport, the phone rejects the call. Otherwise, it switches the main/backup transport and receives the call in the main (JsSIP) transport.
- ACD (subscribe/notify) will be supported on the main channel only. In case of a swap we will re-subscribe to ACD service.

2.7.2 How Phone Code Should be Modified to Use Dual Registration

The easiest way is to open the dual registration phone prototype and find all the references to the *backupSBC* variable in the phone.js file.

- Creating backupSBC object

```
let backupSBC = new BackupSBC();
```

- Setting parameters

```
backupSBC.setAddresses(backupAddresses);
backupSBC.setLogger(backup_sbc_log);
backupSBC.setReconnectIntervals(phoneConfig.reconnectIntervalMin,
  phoneConfig.reconnectIntervalMax);
backupSBC.setKeepAlive(phoneConfig.pingInterval);
```

- Setting listeners: backupSBC.setListeners(…);

  There are 2 listeners:

  - loginStateChanged(isLogin, cause) – notify when backup SBC server is connected and registered.
  - incomingInvite(invite) – incoming in backup SBC server INVITE.

    ```
    When there is incoming in backup SBC INVITE
    phone must decide if answer the call
    (backupSBC.swap(invite))
    or reject it (backupSBC.reject(invite))
    ```

  - If the phone detects a sequence of main SBC failures and the backup SBC is registered, it swaps the main/backup SBC. Here is part of the code:

```
        if (disconnectCounter > disconnectMaxCounter) {
              . . .
              if (backupSBC.isRegistered) {
                  . . .
                  backupSBC.swap();
                  . . .
              }
        }
```

- Code keeps currently used main and backup SBC after HTML page reloading.

See usage of **isSwapped** variable. It is set in the **onBeforeUnload** function**,** saved to session storage, and used when the phone has restarted.

3 API Callbacks / Listeners Interfaces

This API provides the capability to register and 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 outgoingCallProgress (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);
```

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 then 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:

- "ACK received" for incoming call
- ACK sent for outgoing call

3.2.3 Call Terminated

Triggered when a call has 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']
- Cause: text description of termination reason

The following is a list of frequently used Call Termination cases:

1. For outgoing redirected calls (received SIP 3xx response to sent initial SIP INVITE):
```
message = 3xx response
cause = "Redirected"
redirectTo = [String] NameAddress part of received 3xx
response contact header
```
2. For incoming call called method:
```
reject(), redirect() or terminate()
message = null
cause = "Rejected"
redirectTo = null
```
3. For outgoing call to non-existing user:
```
message = response 404
cause = "Not Found"
redirectTo = null
```
4. For open call called method: terminate():
```
message = null
cause = "Terminated"
redirectTo = null
```
5. For open call received BYE message:
```
message received BYE
cause = "Terminated"
redirectTo = null
```

The following is a list of causes from JsSIP documentation.

Common Causes

- Connection Error: WebSocket connection error occurred
- Request Timeout: No response received, timeout expired for the client transaction
- SIP Failure Code: A negative SIP response was received which is not part of any of the groups defined in SIP Error Causes
- Internal Error: Unexpected error
- Missing SDP: Received a request/response that should have SDP body but did not'

SIP Error Causes

Busy SIP codes:	486, 600
Rejected SIP codes:	403, 603
Redirected SIP codes:	300, 301, 302, 305, 380
Unavailable SIP codes:	480, 410, 408, 430
Not Found SIP codes:	404, 604
Address Incomplete SIP code:	484
Incompatible SDP SIP codes:	488, 606
Authentication Error SIP codes:	401, 407

RTCSession Causes

Terminated:	RTCSession terminated normally by local or remote peer
Canceled:	RTCSession canceled by local or remote peer
No Answer:	Incoming call was not answered in the time given in the configuration no_answer_timeout parameter
Expires:	Incoming call contains a Expires header and local user did not answer within the time given in the header
No ACK:	An incoming INVITE was replied with 2XX status code, but no ACK was received
Dialog Error:	An in-dialog request received a 408 or 481 SIP error
User Denied Media Access:	Local user denied media access when prompted for audio/video devices
Bad Media Description:	Received SDP is wrong
RTP Timeout	Session ended due to loss of RTP

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 a 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]

3.2.13 callScreenSharingEnded

Notifies that a screen sharing video stream is closed.

It can be closed when:

- The called method is stopScreenSharing.
- Pressing the Chrome built-in “Stop sharing” button.
- The call using screen sharing has terminated.

The callback can be used for the control phone GUI.

Syntax

```
void callScreenSharingEnded(call, stream)
```

Parameters

- call [AudioCodesSession]: The call session object
- stream [MediaStream]: The media stream created by phone.openScreenSharing().

3.2.14 incomingSubscribe

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

Syntax

```
int incomingSubscribe(subscribe, eventName, accepts)
```

Parameters

- subscribe [IncomingRequest]: The SUBSCRIBE SIP request
- eventName [string]: Event header value.
- Accepts [array of string]: Values of accept headers.

Return Value

- responseCode > 0: Reject incoming SUBSCRIBE with the SIP response code.
- 0: Accept incoming SUBSCRIBE request.
- The request is used to create a subscribe dialog via the subscribe() method.

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');
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.

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 Replace header (that points to an existing open call) in the 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 Alert-Info header. 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 ----
```

5 Tutorial

You can find a useful tutorial with AudioCodes-provided Web RTC examples at https://demo.webrtc.audiocodes.com/sdk/webrtc-api-base/examples/tutorial.html.

International Headquarters
Naimi Park
6 Ofra Haza Street
Or Yehuda, Israel
Tel: +972-3-976-4000
Fax: +972-3-976-4040

AudioCodes Inc.
80 Kingsbridge Rd
Piscataway, NJ 08854, USA
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-14059**