Table of Contents

1 Introduction .................................................................................................................. 7
  1.1 Purpose ................................................................................................................. 7
  1.2 Scope ..................................................................................................................... 7
  1.3 Benefits .................................................................................................................. 7

2 API Classes .................................................................................................................. 9
  2.1 AudioCodesUA ........................................................................................................ 9
    2.1.1 Standard Methods .......................................................................................... 10
    2.1.1.1 constructor ............................................................................................... 10
    2.1.1.2 init ............................................................................................................ 10
    2.1.1.3 setServerConfig ......................................................................................... 10
    2.1.1.4 setAccount ................................................................................................. 11
    2.1.1.5 login .......................................................................................................... 11
    2.1.1.6 logout ........................................................................................................ 11
    2.1.1.7 setListeners ............................................................................................... 11
    2.1.1.8 call ............................................................................................................. 12
    2.1.2 Advanced Methods .......................................................................................... 13
    2.1.2.1 setRegisterExtraHeaders ......................................................................... 13
    2.1.2.2 call ............................................................................................................ 13
    2.1.2.3 setUseSessionTimer ................................................................................ 13
    2.1.2.4 setRegisterExpires .................................................................................. 14
    2.1.2.5 isInitialized .............................................................................................. 14
    2.1.2.6 version ....................................................................................................... 14
    2.1.2.7 setChromeAudioConstraints .................................................................... 14
    2.1.2.8 setUserAgent ........................................................................................... 15
    2.1.2.9 setAcLogger .............................................................................................. 15
    2.1.2.10 setJsSipLogger ...................................................................................... 15
    2.1.2.11 setWebSocketKeepAlive ........................................................................ 16
    2.1.2.12 setReconnectIntervals ............................................................................ 16
    2.1.2.13 deinit ....................................................................................................... 17
    2.1.2.14 setDtmfOptions ...................................................................................... 17
    2.1.2.15 setOAuthToken ....................................................................................... 17
    2.1.2.16 setEnableAddVideo ............................................................................... 18
    2.1.2.20 getStreamInfo ......................................................................................... 19
  2.2 AudioCodesSession ................................................................................................. 20
    2.2.1 Standard Methods .......................................................................................... 20
    2.2.1.1 answer ...................................................................................................... 20
    2.2.1.2 reject ....................................................................................................... 21
    2.2.1.3 redirect ..................................................................................................... 21
    2.2.1.4 terminate ................................................................................................. 21
    2.2.1.5 muteAudio ............................................................................................... 22
    2.2.1.6 muteVideo ............................................................................................... 22
    2.2.1.7 isAudioMute ............................................................................................. 22
    2.2.1.8 isVideoMute ............................................................................................. 22
    2.2.1.9 sendDTMF ............................................................................................... 23
    2.2.1.10 isOutgoing ............................................................................................ 23
    2.2.1.11 data: map<String, Object> .................................................................. 23
    2.2.1.12 duration ................................................................................................. 24
    2.2.1.13 isLocalHold ............................................................................................. 24
    2.2.1.14 isRemoteHold ......................................................................................... 24
    2.2.1.15 IsReadyToReOffer ............................................................................... 24
    2.2.1.16 hold ........................................................................................................ 25
    2.2.2 Advanced Methods .......................................................................................... 25
    2.2.2.1 answer ...................................................................................................... 25
3 API Callbacks / Listeners Interfaces

3.1 Standard Callbacks
3.1.1 Login State Changed Event
3.1.2 Incoming Call Event
3.1.3 Call Confirmed
3.1.4 Call Terminated
3.1.5 Outgoing Call Progress
3.1.6 Call Show Streams

3.2 Advanced Callbacks
3.2.1 Incoming call event
3.2.2 Call Confirmed
3.2.3 Call Terminated
3.2.4 Outgoing Call Progress
3.2.5 callHoldStateChanged
3.2.6 callIncomingReinvite
3.2.7 transferorNotification
3.2.8 transfereeRefer
3.2.9 transfereeCreatedCall

4 Use Examples
4.1 User Agent: Create Instance, Set Server and Account
4.2 User Agent: Set Listeners (Callbacks)
4.3 User Agent Init: Connection to SBC Server and Login
4.4 Make a Call
4.5 Send DTMF During Call
4.6 Mute / Unmute During Call
4.7 Accept Incoming Call
4.8 Reject Incoming Call
4.9 Terminate a Call
4.10 Use of Remote Streams Video
4.11 Restore Call after Page Refresh
4.12 Set Custom Logger
4.13 Getting Statistics

5 Tutorial
Notice

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

This document is subject to change without notice.

Date Published: January-23-2019

WEEE EU Directive

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

Customer Support

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

Stay in the Loop with AudioCodes

Abbreviations and Terminology

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

Related Documentation

<table>
<thead>
<tr>
<th>Document Name</th>
</tr>
</thead>
</table>
## Document Revision Record

<table>
<thead>
<tr>
<th>LTRT</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>14040</td>
<td>Initial document release for Version 1.0</td>
</tr>
<tr>
<td>14041</td>
<td>Updated to Version 1.1</td>
</tr>
<tr>
<td>14042</td>
<td>Added deinit and setDtmfOptions</td>
</tr>
<tr>
<td>14043</td>
<td>Update for Version 1.3: Added setOAuthToken and setEnableAddVideo</td>
</tr>
</tbody>
</table>
| 14044 | Updates for Version 1.4 (added the following new functions):  
  - getBrowserName()  
  - getServerAddress()  
  - checkAvailableDevices()  
  - getStreamInfo()  
  - startSendingVideo()  
  - hasVideo()  
  - hasSendVideo()  
  - hasReceiveVideo()  
  - getVideoState()  
  - setRemoteHoldState() |
| 14045 | Updates for Version 1.5:  
  - Added optional callbacks:  
    ✓ callIncomingReinvite()  
    ✓ transferorNotification()  
    ✓ transfereeRefer()  
    ✓ transfereeCreatedCall()  
  - Added function: sendRefer() |

## Documentation Feedback

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

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

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

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

1.1 Purpose

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

1.2 Scope

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

1.3 Benefits

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

- Simple deployment - a single WebRTC gateway device for both signaling and media
- Strong security and interoperability capabilities resulting from integration with the SBC
- Client SDK for browsers.
- OPUS powered IP phones for superb, transcoder-less voice quality
This page is intentionally left blank.
2 API Classes

Two usable objects are available:
- AudioCodesUA – Audio Codes User Agent (single tone) – see below
- AudioCodesSession – for call representation – see below

2.1 AudioCodesUA

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

```java
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(autologin=true);
    void login();
    void logout();
    AudioCodesSession call(withVideo, call_to, replacesHeader=null);
    void setRegisterExtraHeaders (List<SipHeader> extraHeaders);
    void setInviteExtraHeaders (boolean isRequest, List<SipHeader> extraHeaders);
    void setUseSessionTimer(boolean use);
    void setRegisterExpires(int expires);
    boolean isInitialized();
    String version();
    void setUserAgent(String name);
    void setChromeAudioConstraints(String constraings)
    void setAcLogger(Function logger);
    void setJsSipLogger(Function logger);
    void setWebSocketKeepAlive(int ping, int pong=0, int stats=0);
    void setReconnectIntervals(int min, int max);
    void deinit();
    void setDtmfOptions(boolean useWebRTC, int duration=null, int interToneGap=null)
    void setOAuthToken(String token)
    void setEnableAddVideo(boolean enable)
    String getBrowserName()
    String getServerAddress()
    Promise checkAvailableDevices()
    Promise getStreamInfo(RTCConnection conn, boolean isLocalStream, boolean isAudio)
}
```
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 [By default, 'True'; after connection, automatically call login()] - Optional

Return Values
N/A

2.1.1.3 setServerConfig
Configures the server.

Parameters
- ServerAddresses (2 options)
  - serverAddresses [inetSocket Address List of the AudioCodes Mediant servers]
  - serverAddresses [Two elements array list; each array in the list contains the inetSocket Address of the AudioCodes Mediant servers and priority(integer)]
- serverDomain [String of the domain name to register to]
- 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 call opening, there will be no usage of an external STUN server.
  - This mode is useful when the phone is used towards an SBC server.

Note: After call open client send periodically STUN requests to the port used for RTP stream, to checking that RTC channel is alive.

Return Values
N/A
2.1.1.4 setAccount

Defines the account details.

Parameters
- userName [String, authenticating user name]
- displayName [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**
- withVideo [Boolean 'True' if the call must be initiated with video]
- call_to [String of a destination address/number]

**Return Values**
A call session object is defined here.

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

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

### 2.1.2.1 `setRegisterExtraHeaders`

Allows adding additional headers to the registration request.

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

**Parameter**
- ExtraHeaders [List of headers]

**Return Values**
- N/A

### 2.1.2.2 `call`

Initiates an outgoing call.

**Parameters**
- withVideo [Boolean 'True' if the call must be initiated with video]
- call_to [String of a destination address/number]
- extraHeaders (Optional). An array of strings, each representing a SIP header, to be added to the INVITE request.

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

**Return Values**
- A call session object defined as shown under Section 2.2.

### 2.1.2.3 `setUseSessionTimer`

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

**Parameter**
- enable [Boolean ['True' if yes; 'False' if no]]

**Return Values**
- N/A
### 2.1.2.4 setRegisterExpires

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

**Parameter**

- expires [integer [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 with the API version.

### 2.1.2.7 setChromeAudioConstraints

Changes the echo cancellation and noise suppression flags specifically for the Chrome browser.

**Parameters**

- String with parameters, delimited by comma. The following can be used:
  
  `'echoCancellation,googEchoCancellation,googExperimentalEchoCancellation,googDAEchoCancellation,googAutoGainControl,googNoiseSuppression,googHighpassFilter,googTypingNoiseDetection,googAudioMirroring'`

**Return Values**

N/A
2.1.2.8 **setUserAgent**

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

**Parameter**
- User agent [String]

**Return Values**
N/A

2.1.2.9 **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

2.1.2.10 **setJsSipLogger**

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

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

**Parameter**
- User-defined logger function name.

**Return Values**
N/A
2.1.2.11 **setWebSocketKeepAlive**

Used to send a CRLF X 2 keepalive via websocket to the SBC. Based on RFC 7118, #6.

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

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

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

**Parameters**
- Ping interval [seconds]
- Ping timeout interval [seconds] (Optional): Timeout awaiting the ping response before declaring the connection as failed.
  - 0 (default): No timeout (managed by the operating system's TCP/IP stack)
- Printing interval [number] (Optional): Printed to the logs every set number of times
  - 0 (default): No printing

**Return Values**
N/A

2.1.2.12 **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
2.1.2.13 **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

2.1.2.14 **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

2.1.2.15 **setOAuthToken**

Sets the access token to OAuth2 authorization. This token will be 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.

**Return Values**

N/A
2.1.2.16 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. We cannot add two-way video, 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

2.1.2.17 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.

2.1.2.18 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.

2.1.2.19 checkAvailableDevices

1. Check if WebRTC API is supported in used browser. If not, the Promise object will be rejected with the following string: "WebRTC is not supported in the browser"

2. Check available devices (speaker, microphone, camera). If the speaker is not connected, the speaker promise object will be rejected with the following string:

"Missing a speaker! Please connect one and reload"

If the microphone is not connected, the microphone will be 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.
The Promise object is rejected with a string describing the problem (see above)

2.1.2.20 getStreamInfo
Get stream information for debug purposes.

Parameters
- RTCConnection [object]
- isLocalStream [boolean] select from the connection's local or remote stream.
- isAudio [boolean] select information about audio or video tracks.

Return Values
The Promise object resolves string with track enabled or disabled information.
In a simple scenario, the connection has one for stream that include one track. In this case, it returns the string in the following format: “true” or “false” (means the track is enabled or disabled).

In complex cases, the information can be in the following format: [true, false], [false] This implies that the connection has two local streams:
- The first stream has two audio tracks: enabled and disabled
- The second stream has one disabled audio track
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

```java
class AudioCodesSession {
    void answer (withVideo)
    void reject ()
    void redirect(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()
    boolean hasVideo()
    boolean hasSendVideo()
    boolean hasReceiveVideo()
    string getVideoState()
    void setRemoteHoldState()
}
```

2.2.1 Standard Methods

2.2.1.1 answer

Initiates the object and establishes the call.

Parameter

- withVideo [Boolean 'True' if the call is with video]

Return Values

N/A

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

Rejects a call.

Parameters
N/A

Return Values
N/A

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

2.2.1.3 redirect

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

Parameter
Call_to [String of destination address/number] Used for SIP response code 302 with Contact header.

Return Values
N/A

Note: This call is also provided with another parameter (see Section 0)

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 value. '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 value; 'True' to mute video; 'False' to unmute video]

Return Values
N/A

2.2.1.7 isAudioMute
Checks the audio mute status.

Parameters
N/A

Return Values
Boolean ['True' if audio is muted, 'False' if audio is unmuted]

2.2.1.8 isVideoMute
Checks video mute status.

Parameters
N/A

Return Values
Boolean ['True' if video is muted, 'False' if video is unmuted]
2.2.1.9 sendDTMF
Sends a DTMF character.

**Parameter**
- dtmf [One DTMF character]

**Return Values**
N/A

2.2.1.10 isOutgoing
Checks if a call is outgoing.

**Parameters**
N/A

**Return Values**
Boolean 'True' if a call is outgoing, 'False' if a call is incoming.

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

Four values are available and predefined by the API (distinguishable from the other objects listed by the underscore prefix _ in the key):

<table>
<thead>
<tr>
<th>Predefined Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>data['_user']</td>
</tr>
<tr>
<td>data['_display_name']</td>
</tr>
<tr>
<td>data['_create_time']</td>
</tr>
</tbody>
</table>

**Note:** Names that start with '_' are reserved for internal API use. Do not use such names. Save your values in `call.data`. 
2.2.1.12 duration

Defines the call duration, in seconds. This value will be ‘Null’ if the call has still not been established or has not yet been terminated.

2.2.1.13 isLocalHold

Defines whether the client initiates the hold. 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.

2.2.1.14 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.

2.2.1.15 IsReadyToReOffer

Implements a hold using SIP re-INVITE; checks before using 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.
2.2.1.16 **hold**

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

**Parameter**
- Hold [Boolean] - sets the call to hold.

**Return Values**
Promise to wait till the end of the operation.

**2.2.2 Advanced Methods**

**2.2.2.1 answer**

Initiates the object and establishes the call.

**Parameters**
- withVideo [Boolean 'True' if the call is with video]
- extraHeaders (Optional): An array of strings, each representing a SIP header, to be added to the request.

**Return Values**
N/A

**2.2.2.2 reject**

Rejects a call.

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

**Return Values**
N/A
2.2.2.3 redirect

Redirects the call. Asks the caller to call the destination.

Parameters

- Call_to [String of the destination address/number] that is used for SIP response code 302 with Contact header.
- Status code (Optional): Integer representing the reject reason (3xx codes, 302 moved temporarily (default))
- extraHeaders (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

String replaces the header value according RFC 3891, or 'Null' if the call is not established. The string may be used to re-establish a failed call session on the client side (if a session still exists on the SBC, for example, in the case of a page refresh). See the Use Examples under Section 4 for more details.

2.2.2.5 getRTCPeerConnection

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

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

For more information see:
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 startSendingVideo
For audio call, starts sending video stream to other side.

Parameters
N/A

Return Values
Promise object with true if successfully started, and false otherwise.

2.2.2.7 hasVideo, hasSendVideo, hasReceiveVideo
Checks call video status:
- hasVideo returns true, if two-way video is connected for the session.
- hasSendVideo return true, if outgoing video stream is connected for the session.
- hasReceiveVideo return true, if incoming video stream is connected for the session.

In cases where 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 call restore functionality.

Parameters
N/A

Return Values
Boolean

2.2.2.8 getVideoStatus
Returns the string representation of the call video status.

2.2.2.9 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
2.2.2.10 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]

Return Values
N/A
3 API Callbacks / Listeners Interfaces

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

3.1 Standard Callbacks

3.1.1 Login State Changed Event

Triggered when the login state is changed.

Syntax

```java
void loginStateChanged(boolean isLogin, string cause);
```

Parameter

- `isLogin`: 'True' if logged in, and 'False' if not logged in.
- `cause`: one of these strings:
  - "connected"
  - "disconnected"
  - "login failed"
  - "login"
  - "logout"

3.1.2 Incoming Call Event

Triggered when receiving an incoming call.

Syntax

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

Parameter

- `AudioCodesSession`: [The call session object]

3.1.3 Call Confirmed

Triggered when the call is established.

Syntax

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

Parameter

- `AudioCodesSession`: [The call session object]
### 3.1.4 Call Terminated

Triggered when a call is terminated or fails.

**Syntax**

```java
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**

```java
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**

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

**Note:** Only relevant for the browser API.

**Parameters**

- AudioCodesSession [The call session object]
- localStream [The stream from the local camera and microphone]
- remoteStream [The stream from the remote camera and microphone]
3.2 Advanced Callbacks

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

3.2.1 Incoming call event

Triggered when receiving an incoming call.

Syntax
void incomingCall(AudioCodesSession call, SipRequest invite);

Parameters
- AudioCodesSession [The call session object]
- SipRequest [The SIP request object]

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 is one of these strings:
- "received ack"
- "sent ack"

3.2.3 Call Terminated

Triggered when a call is terminated or if it fails.

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

Parameters
- AudioCodesSession [The call session object]
- SipMessage [The BYE SIP message; optional, might be 'Null']
3.2.4 Outgoing Call Progress

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

Syntax

```java
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

```java
void callHoldStateChanged(AudioCodesSession call, boolean isHold, boolean isRemote);
```

Parameters

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

3.2.6 callIncomingReinvite

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

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

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

Syntax

```java
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 be used only 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 could check REFER message headers and should accept or reject an incoming REFER message. The callback is optional, and should be used only if the phone supports call transfer as 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 get the reference to the newly created call object. The callback is optional, and should be used only if the phone supports call transfer as transferee.
Syntax
void transfereeCreatedCall(AudioCodesSession call);

Parameters
- AudioCodesSession [Newly created call session object]
4 Use Examples

This chapter shows examples that can help guide your implementation.

4.1 User Agent: Create Instance, Set Server and Account

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

4.2 User Agent: Set Listeners (Callbacks)

```javascript
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

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

4.4 Make a Call

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

4.5 Send DTMF During Call

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

4.6 Mute / Unmute During Call

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

4.7 Accept Incoming Call

```javascript
activeCall.answer(withVideo);
```
4.8 Reject Incoming Call

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

4.9 Terminate a Call

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

4.10 Use of Remote Streams Video

```javascript
// 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.

```javascript
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.

```javascript
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', 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.

```javascript
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 example output to the console for outband-rtp and inbound-rtp.

```javascript
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;
        }
    });
}
```
```javascript
})
}).then(()=>{
    ac_log('call stats: ' + str);
})
.catch((err)=>{
    ac_log('stat error', err);
});
```
5 Tutorial

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