IP Office™ Platform
Description of WebRTC SDK API

© 2020 Avaya Inc. All Rights Reserved.

Avaya grants You a license within the scope of the license types described below, with the exception of Heritage Nortel Software, for which the scope of the license is detailed below. Where the order documentation does not expressly identify a license type, the applicable license will be a Designated System License. The applicable number of licenses and units of capacity for which the license is granted will be one (1), unless a different number of licenses or units of capacity is specified in the documentation or other materials available to You. "Software" means computer programs in object code, provided by Avaya or an Avaya Channel Partner, whether as stand-alone products, pre-installed on hardware products, and any upgrades, updates, patches, bug fixes, or modified versions thereto. "Designated Processor" means a single stand-alone computing device. "Server" means a Designated Processor that hosts a software application to be accessed by multiple users. "Instance" means a single copy of the Software executing at a particular time: (i) on one physical machine; or (ii) on one deployed software virtual machine ("VM") or similar deployment.

License type(s)
Designated System(s) License (DS). End User may install and use each copy or an Instance of the Software only on a number of Designated Processors up to the number indicated in the order. Avaya may require the Designated Processor(s) to be identified in the order by type, serial number, feature key, Instance, location or other specific designation, or to be provided by End User to Avaya through electronic means established by Avaya specifically for this purpose.

Concurrent User License (CU). End User may install and use the Software on multiple Designated Processors or one or more Servers, so long as only the licensed number of Units are accessing and using the Software at any given time. A "Unit" means the unit on which Avaya, at its sole discretion, bases the pricing of its licenses and can be, without limitation, an agent, port or user, an e-mail or voice mail account in the name of a person or corporate function (e.g., webmaster or helpdesk), or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software. Units may be linked to a specific, identified Server or an Instance of the Software.

Database License (DL). End User may install and use each copy or an Instance of the Software on one Server or on multiple Servers provided that each of the Servers on which the Software is installed communicates with no more than one Instance of the same database.

CPU License (CP). End User may install and use each copy or Instance of the Software on a number of Servers up to the number indicated in the order provided that the performance capacity of the Server(s) does not exceed the performance capacity specified for the Software. End User may not re-install or operate the Software on Server(s) with a larger performance capacity without Avaya's prior consent and payment of an upgrade fee.

Named User License (NU). You may: (i) install and use each copy or Instance of the Software on a single Designated Processor or Server per authorized Named User (defined below); or (ii) install and use each copy or Instance of the Software on a Server so long as only authorized Named Users access and use the Software. "Named User", means a user or device that has been expressly authorized by Avaya to access and use the Software. At Avaya's sole discretion, a "Named User" may be, without limitation, designated by name, corporate function (e.g., webmaster or helpdesk), an e-mail or voice mail account in the name of a person or corporate function, or a directory entry in the administrative database utilized by the Software that permits one user to interface with the Software.

Shrinkwrap License (SR). You may install and use the Software in accordance with the terms and conditions of the applicable license agreements, such as "shrinkwrap" or "clickthrough" license accompanying or applicable to the Software ("Shrinkwrap License").

Heritage Nortel Software
"Heritage Nortel Software" means the software that was acquired by Avaya as part of its purchase of the Nortel Enterprise Solutions Business in December 2009. The Heritage Nortel Software is the software contained within the list of Heritage Nortel Products located at https://support.avaya.com/LicenseInfo under the link "Heritage Nortel Products" or such successor site as designated by Avaya. For Heritage Nortel Software, Avaya grants Customer a license to use Heritage Nortel Software provided hereunder solely to the extent of the authorized activation or authorized usage level, solely for the purpose specified in the Documentation, and solely as embedded in, for execution on, or for communication with Avaya equipment. Charges for Heritage Nortel Software may be based on extent of activation or use authorized as specified in an order or invoice.

Copyright
Except where expressly stated otherwise, no use should be made of materials on this site, the Documentation, Software, Hosted Service, or hardware provided by Avaya. All content on this site, the documentation, Hosted Service, and the product provided by Avaya including the selection, arrangement and design of the content is owned either by Avaya or its licensors and is protected by copyright and other intellectual property laws including the sui generis rights relating to the protection of databases. You may not modify, copy, reproduce, republish, upload, post, transmit or distribute in any way any content, in whole or in part, including any code and software unless expressly authorized by Avaya. Unauthorized reproduction, transmission, dissemination, storage, and or use without the express written consent of Avaya can be a criminal, as well as a civil offense under the applicable law.

Virtualization
The following applies if the product is deployed on a virtual machine. Each product has its own ordering code and license types. Note that each Instance of a product must be separately licensed and ordered. For example, if the end user customer or Avaya Channel Partner would like to install two Instances of the same type of products, then two products of that type must be ordered.

Third Party Components
"Third Party Components" mean certain software programs or portions thereof included in the Software or Hosted Service may contain software (including open source software) distributed under third party agreements ("Third Party Components"), which

contain terms regarding the rights to use certain portions of the Software ("Third Party Terms"). As required, information regarding distributed Linux OS source code (for those products that have distributed Linux OS source code) and identifying the copyright holders of the Third Party Components and the Third Party Terms that apply is available in the products, Documentation or on Avaya's website at: https://support.avaya.com/Copyright or such successor site as designated by Avaya. The open source software license terms provided as Third Party Terms are consistent with the license rights granted in these Software License Terms, and may contain additional rights benefiting You, such as modification and distribution of the open source software. The Third Party Terms shall take precedence over these Software License Terms, solely with respect to the applicable Third Party Components to the extent that these Software License Terms impose greater restrictions on You than the applicable Third Party Terms.

The following applies if the H.264 (AVC) codec is distributed with the product. THIS PRODUCT IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO (i) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD ("AVC VIDEO") AND/OR (ii) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

Service Provider
THE FOLLOWING APPLIES TO AVAYA CHANNEL PARTNER'S HOSTING OF AVAYA PRODUCTS OR SERVICES. THE PRODUCT OR HOSTED SERVICE MAY USE THIRD PARTY COMPONENTS SUBJECT TO THIRD PARTY TERMS AND REQUIRE A SERVICE PROVIDER TO BE INDEPENDENTLY LICENSED DIRECTLY FROM THE THIRD PARTY SUPPLIER. AN AVAYA CHANNEL PARTNER'S HOSTING OF AVAYA PRODUCTS MUST BE AUTHORIZED IN WRITING BY AVAYA AND IF THOSE HOSTED PRODUCTS USE OR EMBED CERTAIN THIRD PARTY SOFTWARE, INCLUDING BUT NOT LIMITED TO MICROSOFT SOFTWARE OR CODECS, THE AVAYA CHANNEL PARTNER IS REQUIRED TO INDEPENDENTLY OBTAIN ANY APPLICABLE LICENSE AGREEMENTS, AT THE AVAYA CHANNEL PARTNER'S EXPENSE, DIRECTLY FROM THE APPLICABLE THIRD PARTY SUPPLIER.

WITH RESPECT TO CODECS, IF THE AVAYA CHANNEL PARTNER IS HOSTING ANY PRODUCTS THAT USE OR EMBED THE G.729 CODEC, H.264 CODEC, OR H.265 CODEC, THE AVAYA CHANNEL PARTNER ACKNOWLEDGES AND AGREES THE AVAYA CHANNEL PARTNER IS RESPONSIBLE FOR ANY AND ALL RELATED FEES AND/OR ROYALTIES. THE G.729 CODEC IS LICENSED BY SIPRO LAB TELECOM INC. SEE WWW.SIPRO.COM/CONTACT.HTML. THE H.264 (AVC) CODEC IS LICENSED UNDER THE AVC PATENT PORTFOLIO LICENSE FOR THE PERSONAL USE OF A CONSUMER OR OTHER USES IN WHICH IT DOES NOT RECEIVE REMUNERATION TO: (I) ENCODE VIDEO IN COMPLIANCE WITH THE AVC STANDARD ("AVC VIDEO") AND/OR (II) DECODE AVC VIDEO THAT WAS ENCODED BY A CONSUMER ENGAGED IN A PERSONAL ACTIVITY AND/OR WAS OBTAINED FROM A VIDEO PROVIDER LICENSED TO PROVIDE AVC VIDEO. NO LICENSE IS GRANTED OR SHALL BE IMPLIED FOR ANY OTHER USE. ADDITIONAL INFORMATION FOR H.264 (AVC) AND H.265 (HEVC) CODECS MAY BE OBTAINED FROM MPEG LA, L.L.C. SEE HTTP://WWW.MPEGLA.COM.

Compliance with Laws
Customer acknowledges and agrees that it is responsible for complying with any applicable laws and regulations, including, but not limited to laws and regulations related to call recording, data privacy, intellectual property, trade secret, fraud, and music performance rights, in the country or territory where the Avaya product is used.

Preventing Toll Fraud
"Toll Fraud" is the unauthorized use of your telecommunications system by an unauthorized party (for example, a person who is not a corporate employee, agent, subcontractor, or is not working on your company's behalf). Be aware that there can be a risk of Toll Fraud associated with your system and that, if Toll Fraud occurs, it can result in substantial additional charges for your telecommunications services.

Avaya Toll Fraud intervention
If You suspect that You are being victimized by Toll Fraud and You need technical assistance or support, call Technical Service Center Toll Fraud Intervention Hotline at +1-800-643-2353 for the United States and Canada. For additional support telephone numbers, see the Avaya Support website: https://support.avaya.com or such successor site as designated by Avaya.

Security Vulnerabilities
Information about Avaya's security support policies can be found in the Security Policies and Support section of https://support.avaya.com/security. Suspected Avaya product security vulnerabilities are handled per the Avaya Product Security Support Flow (https://support.avaya.com/css/P8/documents/100161515).

Downloading Documentation
For the most current versions of Documentation, see the Avaya Support website: https://support.avaya.com or such successor site as designated by Avaya.

Contact Avaya Support
See the Avaya Support website: https://support.avaya.com for product or Hosted Service notices and articles, or to report a problem with your Avaya product or Hosted Service. For a list of support telephone numbers and contact addresses, go to the Avaya Support website: https://support.avaya.com (or such successor site as designated by Avaya), scroll to the bottom of the page, and select Contact Avaya Support.

Contents

1 Introduction

Purpose

This document forms part of the SDK for the WebRTC API. An example application is included in the SDK to demonstrate/exercise the interface detailed here. This document provides detailed information about the WebRTC SDK API for IP Office Release 11.1

Intended Audience

This document is for Dev Connect partners developing or integrating web portals or apps to have IP Office based WebRTC functionality integrated in it.

Document Changes

Issue	Date	Description
1.0	July 19, 2015	Initial draft
1.1	Dec 22, 2016	Addition of new optional APIs to get the media devices, select the media devices and attach the video media streams to video media elements. Addition of new optional events to indicate video stream availability and media device list availability.
1.2	Jan 4, 2017	Addition of new API to get Far-end's full-name Addition of new optional event to indicate audio stream availability.
1.3	March 9, 2017	Addition of new API to get the subject of the call or subject of the meeting if any
1.4	May 25, 2017	Changes in the parameters passed to onCallTerminate function
1.5	June 21, 2017	Addition of new API to get alternate server details and changes in setConfiguration API and an optional event to indicate authentication token renewal result.
1.6	July 11, 2017	Addition of a section on Resiliency, new properties passed via callback_onRegistrationStateChanged event, changes in return object of make call API during resiliency
1.7	August 11, 2017	Modification in the parameters passed to onRegistrationStateChanged and onAuthTokenRenewed callback functions. Addition of new API to enable login via resiliency token.
1.8	September 14, 2017	Added Certificate Requirements for Resiliency. Added requirement of hosting Example web page on

		Web Server.
1.9	October 13, 2017	Addition of new API to generate new application instance ID. Modification in the configuration object passed as first parameter to setConfiguration API.
1.10	October 31, 2017	Update Client references
1.11	November 15, 2017	Added new APIs to get and set stun settings any-time after the call to login()
1.12	December 28,2017	User licensing information included in Licensing (1.5.2). Additional information included for addVideo (4.2.13).
1.13	July 13, 2018	Clarify that lack of Safari browser support in WebRTC SDK means no support for iOS at this time (1.4, 2.1) Updated behavior of ICE candidate gathering during makeCall (4.2.1)
1.14	November 19, 2018	SDK zip file updated to resolve two issues below. • Hold Unhold at Called party end fails from Chrome version 69 or later • In some cases of video call the called party is not getting audio-video stream events Call out that WebRTC Gateway resiliency is supported from R11 and later only (1.8).
1.15	January 3, 2019	Update supported releases for each API in section 4
1.16	February 27, 2019	Chrome 72 WebRTC implementation changes updated in SDK
1.17	April 22, 2019	Removed restriction of 6 characters of TLD in FQDN Playing of message provided by remote end in ring back stage
1.18	May 23, 2019	Fix Missing Video Window after resume from hold
1.19	Aug 21, 2019	Updated known issues on Firefox
1.20	Dec 20, 2019	Support three concurrent calls to allow consultative transfer of second call.
1.21	Oct 21, 2020	Support for unified SDP Support for DTLS1.2.

Background

WebRTC is a set of open standards that enables Real-Time Communications (RTC) for web browsers without any plugins.

WebRTC provides an opportunity to enable rich, high quality, RTC applications to be developed for the browser, mobile platforms and allow them all to communicate via a common set of protocols.

WebRTC Gateway for IP Office 11.1 provides capability for IP Office users to have voice, video calling capabilities via WebRTC supported browsers (Chrome and Firefox only – not Internet Explorer, Edge or Safari) across multiple platforms.

Note that Apple requires use of Safari browser for WebRTC support on iOS; the WebRTC SDK does not yet include Safari support – so there is no support for iOS with the WebRTC SDK at this time. Also note that DTMF is only supported with Chrome browser – not with Firefox.

Avaya WebRTC library SDK (AWL SDK) for IP Office is simple JavaScript SDK for adding Softphone functionality for Web based applications, with built in documentation

API

Availability
Avaya IP Office WebRTC SDK API is available for IP Office Release 11.1 with
- IP Office Server Edition
- IP Office Select and
- IP Office Preferred Edition
- Powered By Avaya 3.0 (partner hosted IP Office)

and requires that the WebRTC Gateway be deployed.
WebRTC gateway is bundled with Primary and Linux Application server of IP Office, with the following dependencies
- Server Edition and Select contains application server along with IP Office, so no need to install the application server.
- IP500v2 (only on Preferred Edition) – need to install Linux Application Server separately.

Note
Avaya IP Office WebRTC SDK API Functionality is not supported with (Avaya hosted) Avaya IP Office Cloud releases.

Licensing
There is no license required for WebRTC API.
The associated IP Office users must have Power User or Office Worker profiles to use WebRTC.

What's New in this release
The reference document is updated with details of supported IP Office releases for each API in section 4.

Connectivity

Avaya IP Office WebRTC SDK communicates with WebRTC Gateway ONLY over
- HTTPS Web Socket channels for signaling over port 9443
- DTLS-SRTP channels for Media on 56000-58000 (default)
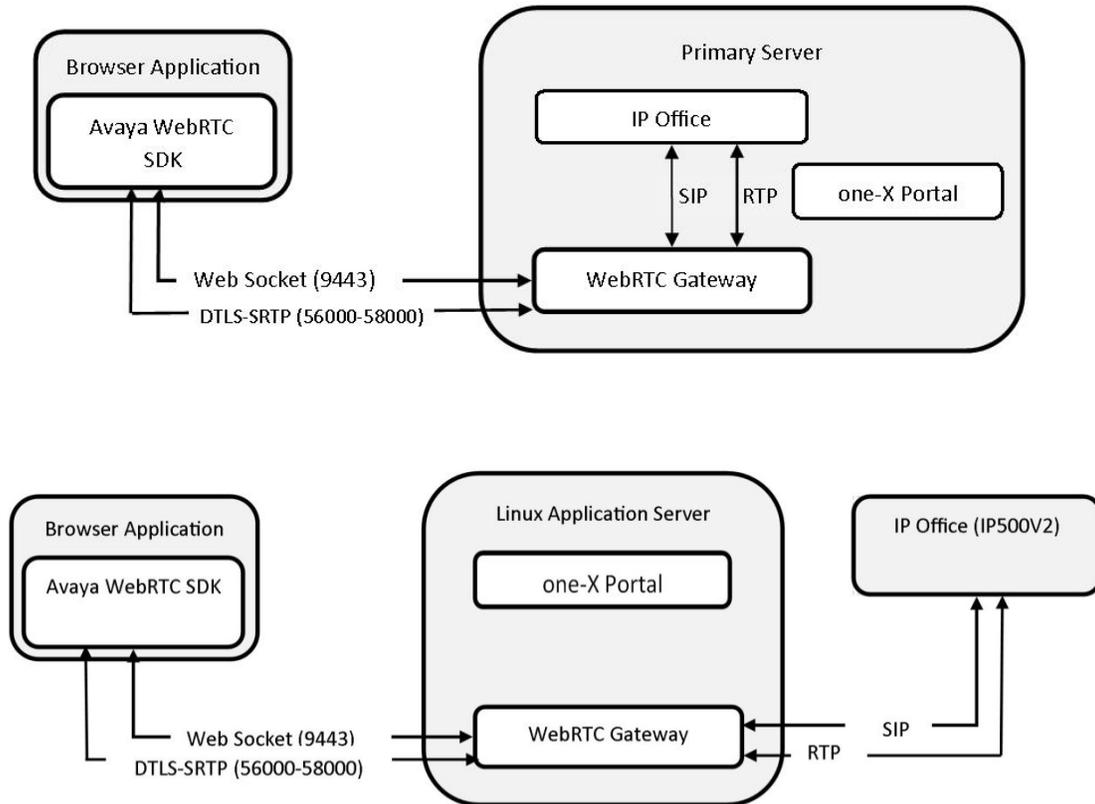
The WebRTC gateway runs as a service at the HTTPS Web Socket port 9443.

Note
The one-X server and WebRTC gateway must be running to use WebRTC functionalities.

WebRTC Gateway Signalling Port
The following TCP ports need to be opened in the firewall /corporate router, in case if web clients reside in public internet.

Release	Port	Network/Application Protocol	Description
11.1	9443	TCP/HTTPS/Web Socket	WebRTC Signaling

The above-mentioned port is fixed and there is no provision to change. The URLs are Web Socket Secure URLs (wss) and are active only when one-X server is running.

WebRTC Gateway Media Port
The below UDP ports need to be opened in the firewall/corporate router in case of clients resides in public internet.

Release	Port	Network/Application Protocol	Description
11.1	56000-58000(default) (Configurable-not to be overlapped with IP Office media endpoint range)	UDP/DTLS-SRTP	WebRTC Media

Certificate Requirements
To improve the security of the WebRTC Gateway link, users should perform the following steps prior to using the client application.

Step 1: Obtain the CA certificate that signed the identity certificate of the WebRTC Gateway server.
Step 2: Install the obtained certificate into Browser's Certificate store and trust the certificate.
Note
1) If certificate is not installed, Web Client will not be able to connect to gateway
2) The gateway does not support mutual authentication and hence, does not require client's certificate
3) The gateway uses the same certificate which one-X portal uses
4) The CA certificate can be obtained from Web Control Portal
5) See *Avaya IP Office™ Platform Security Guidelines* for more information
6) Browser required to install and trust CA Certificates of both Primary and Secondary one-X for Resiliency

WebRTC Resilience

WebRTC Gateway resiliency has a dependency on one-X Portal Resiliency, which is an IP Office Select feature – so WebRTC Gateway resiliency is supported on Select only and from IP Office R11 and later only. The backup WebRTC Gateway is installed by default on the Server Edition Secondary server, providing resiliency for WebRTC clients.

When the Primary WebRTC Gateway service is not available, failover occurs and the Backup WebRTC Gateway becomes active. Clients automatically recognize that the primary WebRTC Gateway is not available and log in to the backup WebRTC Gateway. Logged in users are automatically logged in to the backup WebRTC Gateway.

When the primary WebRTC Gateway is once again available, WebRTC users are automatically failed back to Primary WebRTC Gateway. The backup WebRTC Gateway redirects login requests to the primary WebRTC Gateway.
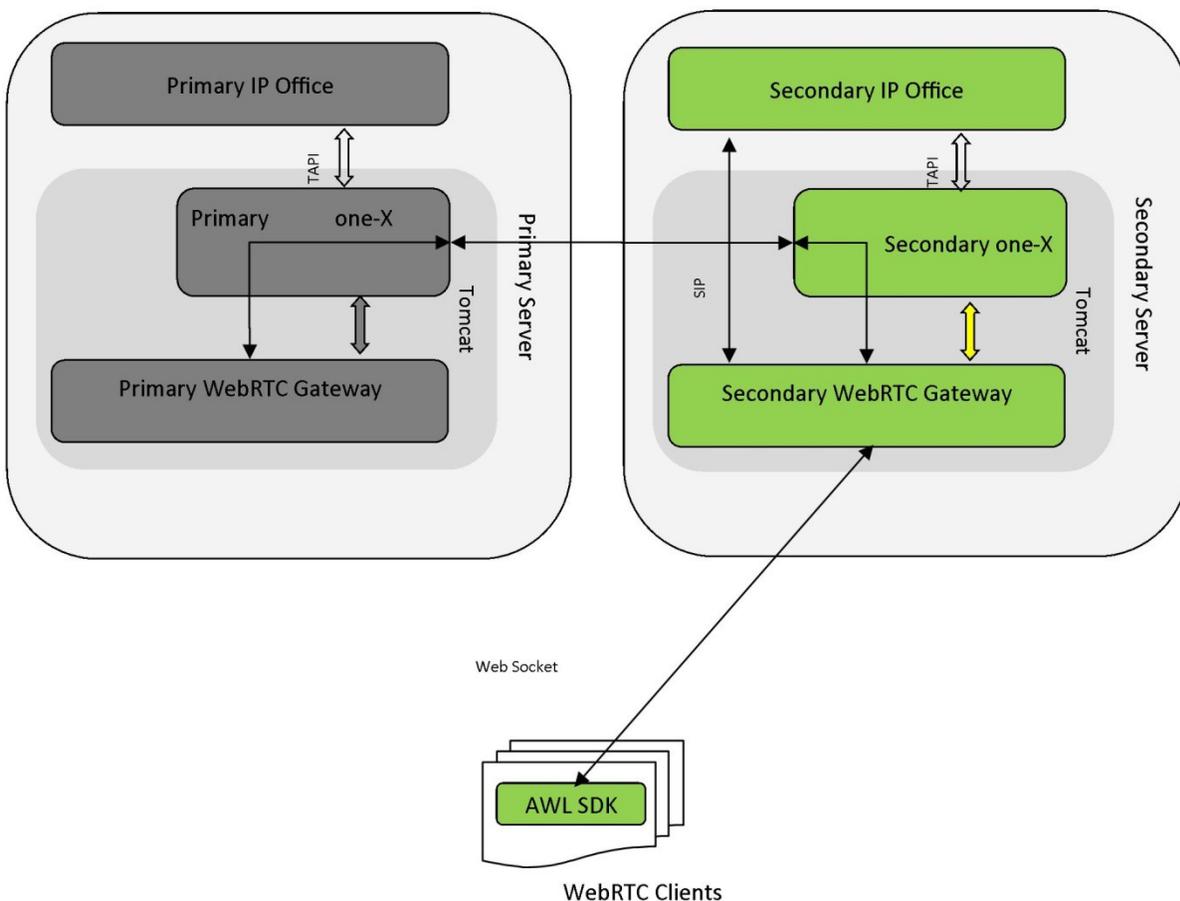
Resiliency API

disableResiliency

>> In a WebRTC Resiliency enabled deployment, the Client application shall enable or disable automatic failover and failback by setting parameter

"disableResiliency" as "false" or "true". Refer section setConfiguration (arg1, callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged) for more details.

getAlternateServerConfig( )

In a WebRTC Resiliency enabled deployment, the Client application can query the Alternate Server details, to manually carry out failover and failback options. Refer section getAlternateServerConfig( ) for more details.

Resiliency Events

In a WebRTC Resiliency enabled deployment, the Client application shall display or act on appropriate resiliency events. Refer sections callback_onRegistrationStateChanged and callback_onAuthTokenRenewed for details.

2   Getting Started

The WebRTC SDK is a collection of JavaScript application programming interfaces (APIs), sample applications and documentation that enable developers to build JavaScript (ECMAScript) based web client/applications.

➢ Avaya distributes the SDK as a zip file, namely "awl.zip". Unzip the downloaded "awl.zip" file to a local drive. The directory structure and the contents will appear as below once unzip is complete.

```
Doc
Example
Avaya_WebRTCSDK_Reference.pdf
awl.min.js
```
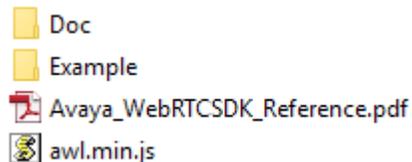
➢ The "Doc" directory contains AWL SDK API usage guide (help documentation) in html format, namely "index.html". The "index.html" is the start page to get started using AWL SDK API. This file is best viewed with Google chrome browser and when launched the webpage shows the summary of the WebRTC SDK module, pre-requisite and mandatory steps to follow when using this AWL SDK file and the list of telephony and non-telephony API's. Detailed explanation with examples is available when clicked on the respective API. The same is available in pdf format which is titled as "Avaya_WebRTCSDK_Reference.pdf".

```
assets
classes
files
modules
api.js
data.json
index.html
logo.png
```
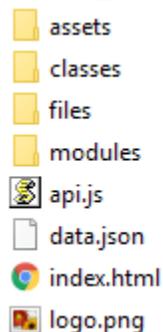
➢ The "Example" directory has contents of a sample web page implementing the AWL SDK API in it and shall be referred in addition to the documentation part. The "Example" directory contains "sdk-testpage" folder. The "sdk-testpage" directory contains "js" folder and index.html. The file "index.html" is best viewed with Google chrome browser and is required to be hosted on Web Server. Include the third-party JQuery library using the HTML <script> tag in "index.html" as follows:

```
<script src="js/thirdparty/jquery-1.8.3.min.js"></script>
```

The "js" folder contains the SDK and the "sdktest.js". The example client code defines a configuration object. The developer shall modify the Gateway IP address and add the port (if necessary) as follows.

```
var cfg = {
        serviceType: "phone",
        enableVideo: true,
        Gateway: {ip: "192.0.2.0", port: "9443"},
        Stunserver: {ip: "", port: "3478"},
        Turnserver: {ip: "", port: "3478", user: "", pwd: ""},
        AppData: {applicationID: "",applicationUA: "", appInstanceID:""},
        disableResiliency: false
        };
```

The application calls the logIn and logOut APIs allowing the user to register and de-register to Avaya IP Office, respectively. After successful registration, it allows the user to make two calls in succession by calling the makeCall API. The developer shall modify the arguments passed to this API as per the requirement. The application requires to be hosted on local or external web server (for example "Apache Tomcat"). The application allows the user to hold, unhold, pauseVideo, resumeVideo, mute, unmute and dropCall by calling the respective Telephony APIs.

➢ File "awl.min.js" represents the complete AWL SDK and developers shall keep this in their web client/application to avail Avaya IP Office WebRTC functionalities.

Supported Browsers

Latest versions of Google Chrome and Firefox browsers are enabled with built-in WebRTC support by default.

Note that Apple requires use of Safari browser for WebRTC support on iOS; the WebRTC SDK does not yet include Safari support – so there is no support for iOS with the WebRTC SDK at this time. Hold and Resume of Video Calls using Firefox browsers result in Audio Call.

Note: Make sure that browsers have permission to access media devices, mic and camera

Tools

Developers shall use JS/HTML5 editors like the ones listed below to view or develop a web client/application with AWL SDK.

➢ Eclipse IDE for JavaScript
➢ Brackets
➢ Sublime Text

Avaya IP Office WebRTC SDK Overview

Avaya IP Office WebRTC SDK Open API is provided for web client application integration using JavaScript language and this API can be used to develop or integrate web pages for consuming the telephony features provided by Avaya IP Office.

Avaya IP Office WebRTC SDK is a JavaScript based minified SDK ("awl.min.js") which takes care of WebRTC functionalities for different WebRTC enabled browsers (latest

versions of Chrome and Firefox) as well handles signaling between the WebRTC Gateway and the web browser client using a secure web socket connection.

The SDK provides Avaya IP Office WebRTC SDK Open API interface, which upon instantiation can be used to invoke the following APIs listed below in order to consume the telephony features provided by Avaya IP Office. The details about these APIs and the parameters passed in them are best described in below sections.

Non Telephony API
1. isWebRTCSupported()
2. getSdkVersion()
3. setConfiguration (arg1, callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged)
4. setConfiguration (arg1, callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged, onAuthTokenRenewed)
5. enableLogging()
6. setDomElements(arg1)
7. getDeviceList(callback_onDeviceListRequested)
8. setDeviceIds(arg1)
9. logIn(arg1, arg2, arg3, arg4)
10. logOut()
11. isloggedIn()
12. setLogObject (arg1)
13. disableLogging()
14. playVideo(arg1)
15. pauseVideo(arg1)
16. getAlternateServerConfig( )
17. tokenLogIn(arg1, arg2, arg3, arg4)
18. generateAppInstanceID ( )
19. getStunConfiguration ( )
20. setStunConfiguration (arg1)

Telephony API
1. makeCall (arg1, arg2)
2. answerCall (arg1)
3. rejectCall (arg1)
4. dropCall (arg1)
5. cancelCall (arg1)
6. doHold (arg1)
7. doUnHold (arg1)
8. doMute (arg1)
9. doUnMute (arg1)
10. sendDTMF (arg1 , arg2)
11. transferCall (arg1 , arg2 , arg3)
12. getStats (arg1)
13. addVideo (arg1)
14. removeVideo (arg1)

15. setMediaStream (arg1 , arg2 , arg3 , arg4)

Network Test API
1. createLoopBackConnection ()
2. endLoopBackConnection ()
3. getLoopBackStats ()

1   Guidelines for using the SDK API

To experience Avaya IP Office WebRTC audio/video calls, following mandatory guidelines have to be met first, before using any other non-Telephony or Telephony API.

Pre-requisites

WebRTC is available with IP Office Server Edition, IP Office Select and IP Office Preferred Edition and to access WebRTC, the associated users must have Power User or Office Worker profiles.
The WebRTC Gateway IP address should be reachable from the web application which is integrated with the SDK.

Mandatory steps

1. Include Avaya IP Office WebRTC SDK file "awl.min.js" in the JavaScript include list

2. Create an AWL.Client instance as this will be used to access all the non-telephony and telephony SDK API in section Detailed Description of API.

   ```
   Example:

   var myWebRTC = new AWL.client();
   ```

3. Set all the configuration (i.e., Populate serviceType("phone" or "agent"), enableVideo, Gateway IP address(Mandatory), Stun/Turn(Optional) server details, application data and disableResiliency details as in the following object literal notation template('cfg')) element(arg1) along with the four callback functions arg2, arg3, arg4 and arg5 which would be triggered upon any configuration changes, registration/un-registration state changes, call state changes and token renewal(if resiliency is not disabled) respectively.

   All these except arg5 are mandatory arguments to be passed while invoking setConfiguration API. Out of all these fields in arg1 template, mandatory data to be filled is Gateway IP address or FQDN in order to setup communication with the Avaya IP Office WebRTC Gateway.

   If the serviceType property is set as Phone service (i.e., "phone"), it provides client to be used as Avaya IP Office extension, with its own UI, whereas if set as Agent service (i.e., "agent") it is better suited for scenarios where the telephony operations are controlled by CTI application, for example in Avaya IP Office Contact Center agent extensions

   For the AppData, appInstanceID is the instance Id of the application and should be unique for each instantiation of application or AWL SDK. The appInstanceID can be obtained from AWL SDK utility API, generateAppInstanceID().The appInstanceID is a mandatory configuration and should be done by every AWL SDK based applications.

   The applicationID and applicationUA are optional configurations to be used by third-party application developers using AWL SDK. The applicationUA represents application name

registered with Avaya and applicationID represents the application identity key obtained from Avaya after the application registered with Avaya.

Note:

1. Client Application must set the remembered/persisted appInstanceID again, when AWL SDK is re-instantiated during Fail-over or Fail-back.

2. applicationID and applicationUA are reserved for future usage.

By default, resiliency is supported at SDK. To disable resiliency support at SDK, applications have to set the disableResiliency property to true while passing the configuration parameter to setConfiguration API. If disableResiliency is set true, then AWL SDK will not renew the authentication token before it expires. Also, autologin using token during failover and failback will not be supported and the applications have to go for manual login during failover and failback.

```
Example:
        var cfg = {
            serviceType: "phone",
            enableVideo: false,
            Gateway: {ip: "192.0.2.0", port: "9443"},
            Stunserver: {ip: "", port: "3478"},
            Turnserver: {ip: "", port: "3478", user: "", pwd: ""},
            AppData: {applicationID : "", applicationUA : "",appInstanceID : ""
},
            disableResiliency : false
        };

        myWebRTC.setConfiguration(cfg, onConfigChanged,
onRegistrationStateChanged, onCallListener, onAuthTokenRenewed);

/*
Where,

            cfg: arg1
            onConfigChanged: callback_onConfigChanged
            onRegistrationStateChanged: callback_onRegistrationStateChanged
            onCallListener: callback_onCallStateChanged
            onAuthTokenRenewed: callback_onAuthTokenRenewed
*/
```

The above API call invokes the configuration change callback function which was passed as 'callback_onConfigChanged ' earlier with the result and reason associated with it.

4. Invoke login API with the Avaya IP Office SIP username and password as the WebRTC client's username and password in the logIn API.

```
    Example:
      myWebRTC.logIn('6501', '********');

                     arg1      arg2
```

The above API call invokes the registration state change callback function which was passed as 'callback_onRegistrationStateChanged ' earlier with the result and reason associated with it.

5. Once the login (i.e., Registration) is reported successful, all the telephony API can be invoked to make or receive calls and other on call relevant features. During call state changes or for an incoming call 'callback_onCallStateChanged' will get triggered.

6. After successful login, if resiliency support at SDK is not disabled by the client, then the authentication token received from the gateway will be used for autologin to alternate server during failover and failback. Also, the authentication token will be renewed by the SDK before its expiry.

2 Detailed description of API

Non-Telephony API

isWebRTCSupported()
This API returns a Boolean result indicating whether the browser (on which this API is called) has WebRTC capability support or not.

Arguments Passed and their description: None
*Return type: **Boolean***

> True indicates that the browser supports WebRTC API.
> False indicates that the browser does not support WebRTC API.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getSdkVersion()
This API returns the Avaya IP Office WebRTC SDK version number being used

Arguments Passed and their description: None
Return type: String

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setConfiguration (arg1, callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged)
This API is used to set configuration parameters in an object literal notation pattern ('cfg' parameter as in below example) and three callback functions which would triggered upon any configuration changes, registration/un-registration state changes and call state changes. This API should be called before doing logIn API call. ALL THE ARGUMENTS TO BE PASSED ARE MANDATORY while invoking setConfiguration API. For the first argument to be passed, the serviceType ("phone" or "agent"), enableVideo (true or false), Gateway IP address or FQDN (Mandatory), Stun/Turn (Optional) server details are to be filled as in the following object literal notation template ('cfg'), along with the three callback functions callback_onConfigChanged, callback_onRegistrationStateChanged and callback_onCallStateChanged which would be triggered upon any configuration changes, registration/un-registration state changes and call state changes respectively. Out of all these fields in arg1 template, mandatory data to be filled is the Gateway IP address in order to setup communication with the Avaya IP Office WebRTC Gateway.

If the serviceType property is set as Phone service (i.e., "phone"), it provides client to be used as Avaya IP Office extension, with its own UI whereas, if set as Agent service (i.e., "agent") it is better suited for scenarios where the telephony operations are controlled by CTI application.

```
Example:
var cfg = {
```

```
            serviceType: "phone",
            enableVideo: false,
            Gateway: {ip: "192.0.2.0", port: "9443"},
            Stunserver: {ip: "", port: "3478"},
            Turnserver: {ip: "", port: "3478", user: "", pwd: ""}
        };

 myWebRTC.setConfiguration(cfg, onConfigChanged, onRegistrationStateChanged,
 onCallListener);

 /*
        cfg: arg1
        onConfigChanged: callback_onConfigChanged
        onRegistrationStateChanged: callback_onRegistrationStateChanged
        onCallListener: callback_onCallStateChanged

 */

 Example of 'callback_onCallStateChanged' class template:
     var onCallListener = function(){
         var _onNewIncomingCall = function(callId, callObj, autoAnswer){
             // application logic here
         }
         var _onCallStateChange = function(callId, callObj, event){
             // application logic here
         }
         var _onCallTerminate = function(callId, reason){
             // application logic here
         }
         var _onLoopBackNotification = function(notification){

         }
        var _onVideoStreamsAvailable = function(callId, localStream,
 remoteStream){
             // application logic here
          }
         var _onAudioStreamsAvailable = function(callId, localStream,
 remoteStream){
             // application logic here
          }
         return{
             onNewIncomingCall: _onNewIncomingCall,
             onCallStateChange: _onCallStateChange,
             onCallTerminate: _onCallTerminate,
             onLoopBackNotification: _onLoopBackNotification,
             onVideoStreamsAvailable: _onVideoStreamsAvailable,
             onAudioStreamsAvailable: _onAudioStreamsAvailable
         };
     }

     var callback_onCallStateChanged= new _onNewIncomingCall();
```

Note: The $setConfiguration$ API call invokes the configuration change callback function which was passed as '$callback\_onConfigChanged$ ' earlier with the result and reason associated with it.

Arguments Passed and their description:

- **arg1** - object literal notation pattern as in above example 'cfg' var.
- **callback_onConfigChanged** - callback function that would get triggered whenever configurations are modified (In above example, 'onConfigChanged' represents this parameter)
- **callback_onRegistrationStateChanged** - callback function that would get triggered whenever there's a change in the registration state (In above example, 'onRegistrationStateChanged' represents this parameter)
- **callback_onCallStateChanged** - callback function object that would trigger 'onCallStateChange' function (when a call state occurs) or 'onNewIncomingCall' function (when there is a new incoming call arrives) or 'onCallTerminate' function(to indicate the call terminate reason, when an established call terminates) or 'onLoopBackNotification' function(notifications and alarms to be handle at application logic) or 'onVideoStreamsAvailable' function (when both local and remote video streams are available) or 'onAudioStreamsAvailable' function (when both local and remote audio streams are available) and this requires a module reveal pattern template to be used as in above example showing 'callback_onCallStateChanged's class template.

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setConfiguration (arg1, callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged, onAuthTokenRenewed)

This API is used to set configuration parameters in an object literal notation pattern ('cfg' parameter as in below example) and four callback functions which would triggered upon any configuration changes, registration/un-registration state changes, call state changes and authentication token renewal. This API should be called before doing logIn API call. **FIRST FOUR ARGUMENTS TO BE PASSED ARE MANDATORY** while invoking setConfiguration API. For the first argument to be passed, the serviceType ("phone" or "agent"), enableVideo (true or false), Gateway IP address or FQDN (Mandatory), Stun/Turn (Optional) server details, Application data and disableResiliency details are to be filled as in the following object literal notation template ('cfg'), along with the four callback functions callback_onConfigChanged, callback_onRegistrationStateChanged, callback_onCallStateChanged and onAuthTokenRenewed which would be triggered upon any configuration changes, registration/un-registration state changes, call state changes and token renewal respectively. Out of all these fields in arg1 template, mandatory data to be filled is the Gateway IP address in order to setup communication with the Avaya IP Office WebRTC Gateway.

If the serviceType property is set as Phone service (i.e., "phone"), it provides client to be used as Avaya IP Office extension, with its own UI whereas, if set as Agent service (i.e., "agent") it is better suited for scenarios where the telephony operations are controlled by CTI application.

For the AppData, appInstanceID is the instance Id of the application and should be unique for each instantiation of application or AWL SDK. The appInstanceID can be obtained from AWL SDK utility API generateAppInstanceID(). The appInstanceID is a mandatory configuration and should be done by every AWL SDK based applications.

The applicationID and applicationUA are optional configurations to be used by third-party application developers using AWL SDK. The applicationUA represents application name registered with Avaya and applicationID represents the application identity key obtained from Avaya after the application registered with Avaya.

Note:

1. Client application must set the remembered/persisted appInstanceID again, when AWL SDK is re-instantiated during Fail-over or Fail-back

2. applicationID and applicationUA are reserved for future usage.

```
Example:
    var cfg = {
        serviceType: "phone",
        enableVideo: false,
        Gateway: {ip: "192.0.2.0", port: "9443"},
        Stunserver: {ip: "", port: "3478"},
        Turnserver: {ip: "", port: "3478", user: "", pwd: ""},
        AppData: {applicationID : "", applicationUA : "", appInstanceID : "" },
        disableResiliency : false
    };


myWebRTC.setConfiguration(cfg, onConfigChanged, onRegistrationStateChanged,
onCallListener, onAuthTokenRenewed);

/*
    cfg: arg1
    onConfigChanged: callback_onConfigChanged
    onRegistrationStateChanged: callback_onRegistrationStateChanged
    onCallListener: callback_onCallStateChanged
    onAuthTokenRenewed: callback_onAuthTokenRenewed


*/


Example of 'callback_onCallStateChanged' class template:
    var onCallListener = function(){
        var _onNewIncomingCall = function(callId, callObj, autoAnswer){
            // application logic here
        }
        var _onCallStateChange = function(callId, callObj, event){
            // application logic here
        }
        var _onCallTerminate = function(callId, reason){
            // application logic here
        }
        var _onLoopBackNotification = function(notification){

        }
```

```
        var _onVideoStreamsAvailable = function(callId, localStream,
remoteStream){
            // application logic here
        }
        var _onAudioStreamsAvailable = function(callId, localStream,
remoteStream){
            // application logic here
        }
        return{
            onNewIncomingCall: _onNewIncomingCall,
            onCallStateChange: _onCallStateChange,
            onCallTerminate: _onCallTerminate,
            onLoopBackNotification: _onLoopBackNotification,
            onVideoStreamsAvailable: _onVideoStreamsAvailable,
            onAudioStreamsAvailable: _onAudioStreamsAvailable
        };
    }

    var callback_onCallStateChanged= new _onNewIncomingCall();
```

Note: The $setConfiguration$ API call invokes the configuration change callback function which was passed as '$callback\_onConfigChanged$ ' earlier with the result and reason associated with it.

Arguments Passed and their description:
  ▪ **arg1** - object literal notation pattern as in above example 'cfg' var.
  ▪ **callback_onConfigChanged** - callback function that would get triggered whenever configurations are modified (In above example, 'onConfigChanged' represents this parameter)
  ▪ **callback_onRegistrationStateChanged** - callback function that would get triggered whenever there's a change in the registration state (In above example, 'onRegistrationStateChanged' represents this parameter)
  ▪ **callback_onCallStateChanged** - callback function object that would trigger 'onCallStateChange' function (when a call state occurs) or 'onNewIncomingCall' function (when there is a new incoming call arrives) or 'onCallTerminate' function(to indicate the call terminate reason, when an established call terminates) or 'onLoopBackNotification' function(notifications and alarms to be handle at application logic) or 'onVideoStreamsAvailable' function (when both local and remote video streams are available) or 'onAudioStreamsAvailable' function (when both local and remote audio streams are available) and this requires a module reveal pattern template to be used as in above example showing 'callback_onCallStateChanged's class template.
  ▪ **callback_onAuthTokenRenewed** - callback function that would get triggered whenever authentication token renewal succeeds or fails (In above example, 'onAuthTokenRenewed' represents this parameter)

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

enableLogging()

This API enables browser console logging for displaying SDK API logs

Arguments Passed and their description: None
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setDomElements(arg1)

This API used to set HTML5 video media DOM elements (i.e., video tag IDs passed in arg1) which would be used to attach WebRTC video streams of local and remote video by the SDK during a video call. ALL THE PARAMETERS TO BE PASSED ARE MANDATORY. This API should be used only when a pair of DOM elements (local & remote) is fixed across multiple calls to show local & remote video stream in it, i.e. DOM elements should be set before any call happens. At any point of call, the active call's stream will be attached to the DOM element and it will be re-used if the active call session changes. If the application requires dynamic video stream control to attach it to the DOM element on fly, "onVideoStreamsAvailable" callback approach should be used which works per call basis.

```
Example:


    var cfg = {
        localVideo : "", /*should pass the video tag's id value here and
this  tag would be used to attach the local video stream to the user
interface*/
        remoteVideo : "" /*should pass the video tag's id value here and
this  tag would be used to attach the remote video stream to the user
interface*/
    };

    myWebRTC.setDomElements(cfg);


                          arg1
```

Note: The setDomElements API call invokes the configuration change callback function which was passed as 'callback_onConfigChanged' earlier in setConfiguration API with the result and reason associated with it.

Arguments Passed and their description:
 ▪ **arg1** Object - object literal notation pattern as in above example 'cfg' var.
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getDeviceList(callback_onDeviceListRequested)

This API used to get all the available media devices attached to the system.

Note: The getDeviceList API call would invoke the callback function which is passed as mandatory parameter of this API.

Arguments Passed and their description:
 callback_onDeviceListRequested Object - callBack function which will be invoked once all the devices information is found.

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setDeviceIds(arg1)

This API used to set Media Devices obtained by getDeviceList API call.

```
Example:

    var dIds = {
        audioInputID : "",  - should pass the audio input device id value here
        videoInputID: "",  - should pass the video input device id value here
        audioOutputID: "",  - should pass the audio output device id value here and
this tag would be used to attach the remote audio stream to the user interface
        defaultId : true
    };
    myWebRTC.setDeviceIds(dIds);



                            arg1
```

Arguments Passed and their description:
arg1 Object - object literal notation pattern as in above example 'dIds' var.

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

logIn(arg1, arg2, arg3, arg4)

WebRTC gateway expects IP Office user name or extension and user's password from the AWL SDK. The gateway sends authentication request to IP Office with the client provided credentials. The client will be notified about the result of the authentication request once it gets the response from IP Office. WebRTC gateway acts as proxy for the client and initiates IP Office user authentication request on-behalf of client.

Notes:
1) The IP Office user name refers to Name field in IP Office Manager
2) The IP Office extension refers to Extension field under User section in IP Office Manager
3) The Password refers to User's password. Not Login Code.

This API registers the WebRTC client as SIP user with the supplied arguments to IP Office and it takes the user extension (arg1) and password (arg2) as input arguments for internal authentication.

Note: The logIn API call invokes the registration state change callback function which was passed as 'callback_onRegistrationStateChanged' earlier in setConfiguration API

with the result and reason associated with it. If resiliency is supported and enabled, then after successful login, authentication token will be passed to registration state change callback function.

Arguments Passed and their description:

- ▪ **arg1** String - SIP user extension
- ▪ **arg2** String - Respective user(i.e., arg1) password
- ▪ **arg3** String – (Optional)Flag to instruct the Gateway that allow SIP login even if extension is taken over by other client of same service type. The value must be either "true" or "false".

  There cannot be two clients of same service type logged into the same extension simultaneously. When an extension is logged into by a client and another client of the same service type attempts to take the same extension, then the latter client can instruct the Gateway to either allow the SIP login by logging out the former client. To do this, client should pass 'true' or 'false' for the parameter arg3 in logIn.

  If "true", Gateway proceeds for SIP login even if there is any logged in client for the same extension.

  If "false", Gateway checks if any client has already logged into the same extension (arg1), proceeds for SIP login only if there is no already logged in client. If there is any already logged in client, Gateway returns error code and reason string containing the already logged in client's user agent.

- • **arg4** String – (Optional)Flag to instruct the gateway that the 'password'(arg2) is a 'token' not password.

  If this argument is 'true', Gateway treats the 'arg2' as token.

  If this argument is 'false', Gateway treats the 'arg2' as password.

  Default setting for arg4 is 'false'.

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

logOut()

This API unregisters the WebRTC client (which is registered earlier as SIP user using logIn API) from Avaya IP Office
Note: The logIn API call invokes the registration state change callback function which was passed as '<u>callback_onRegistrationStateChanged</u>' earlier in setConfiguration API with the result and reason associated with it
Arguments Passed and their description: *None*
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

isloggedIn()

This API is used to check if the client is currently registered or not

Arguments Passed and their description: None
Return type: Boolean
True indicates that the WebRTC user extension is already registered to Avaya IP Office
False indicates that the WebRTC user extension is not registered to Avaya IP Office.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setLogObject (arg1)

This API enables SDK logging to use any JS framework provided logging object instead of console logging unless this API is used; default logging will be console logging.

```
Example:      myWebRTC.setLogObject($log);


                      arg1
```

Arguments Passed and their description:
  ▪ **arg1** Object - Logging Object
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

disableLogging()

This API disables browser console logging for displaying SDK API logs
Arguments Passed and their description: None
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

playVideo(arg1)

This API is to play the previously paused local video stream.
 Caution: Playing local video stream will have video played effect on all the video calls only on 9.1 version where-as from 10.0 version the local video stream shall be controllable against each call param arg1 is mandatory if used with 10.0 and later build versions
Arguments Passed and their description:
  ▪ **arg1** String - call ID parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)
Return type: Boolean
True indicates the API call is successful.
False indicates the API call failed.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

pauseVideo(arg1)

This API is used to pause the local video stream.

Caution: Pausing local video stream will have video paused effect on all the video calls only on 9.1 versions whereas from 10.0 version it shall be controllable against each call param arg1 is mandatory if used with 10.0 and later build versions

Arguments Passed and their description:

- ▪ **arg1** String - call ID parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Boolean
True indicates the API call is successful.
False indicates the API call failed.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getAlternateServerConfig( )

This API returns the alternate server details if resiliency is supported and enabled at the server.

Arguments Passed and their description: *None*

Return type: Object
This return object contains the alternate server details like IP address, domain, port and server type. If the returned object is null, then resiliency is either not supported or not enabled at the server.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

tokenLogIn(arg1, arg2, arg3, arg4)

This API registers the WebRTC client as SIP user with the supplied arguments to Avaya IP office and it takes userextension(arg1), authentication token(arg2), type of authentication token(arg3) as mandatory arguments.
The authentication token can be of two types: RESILIENCY and ESNA
The client should obtain esna token from the one-x server for a user name (Please refer the one-x open API documentations for details). When gateway receives esna token based authentication request, gateway contacts one-x server to obtain the IP Office user's password corresponding to the token. If gateway obtains the password, it initiates regular IP Office user authentication over SIP channel.
The client can obtain resiliency token from AWL SDK.It first obtains the resiliency token from the response object passed in the registration state change callback function 'callback_onRegistrationStateChanged' when it successfully logs in to resilient server with user password via the logIn API. The client has to update the resiliency token

whenever it is renewed. It can obtain the renewed token via the response object passed in authentication token renewed callback function 'callback_onAuthTokenRenewed'.
Arguments Passed and their description:
- **arg1** String - SIP user extension
- **arg2** String - Authentication token
- **arg3** String – Type of the authentication token. The supported token types are RESILIENCY and ESNA.
- **arg4** String – (Optional)Flag to instruct the Gateway that allow SIP login even if extension is taken over by other client of same service type. The value must be either "true" or "false".

  There cannot be two clients of same service type logged into the same extension simultaneously. When an extension is logged into by a client and another client of the same service type attempts to take the same extension, then the latter client can instruct the Gateway to either allow the SIP login by logging out the former client. To do this, client should pass 'true' or 'false' for the parameter arg3 in logIn.

  If "true", Gateway proceeds for SIP login even if there is any logged in client for the same extension.

  If "false", Gateway checks if any client has already logged into the same extension (arg1), proceeds for SIP login only if there is no already logged in client. If there is any already logged in client, Gateway returns error code and reason string containing the already logged in client's user agent.

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

generateAppInstanceID ( )

This is utility API used to generate the application instance ID.
Note:
a) This API generates new appInstanceID for each invocation.
b) SDK will not persist the generated appInstanceID.
Arguments Passed and their description: *None*
Return type: String

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getStunConfiguration ( )

This API returns the STUN server configurations.
Arguments Passed and their description: *None*
Return type: Object
This return object contains the STUN server details like Stun server IP address/FQDN and port.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setStunConfiguration (arg1)

This API is used to configure STUN server details by the application any-time after call to login(). Subsequent calls shall use new STUN server details.

```
Example:
    var stunServer = {ip: "example.com", port: "3478"};
    var result = myWebRTC.setStunConfiguration(stunServer);



                                         arg1

    if(result === "AWL_MSG_SET_STUN_CONFIG_SUCCESS"){
        console.log("Successfully configured STUN settings");
    }else{
        console.log("Could not configure STUN settings. Re-check the values");
    }
```

Arguments Passed and their description:
arg1 Object - object literal notation pattern as in above example 'stunServer' var.
Return type: String

Possible results that could be passed are:
AWL_MSG_SET_STUN_CONFIG_SUCCESS
AWL_MSG_SET_STUN_CONFIG_FAILED

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

Telephony API

makeCall (arg1, arg2)

This API is used to dial out by passing the terminating DN (arg1) and the call type being either audio or video (arg2) and is responsible for creating WebRTC PeerConnection, Offer generation and other call setup signaling.

Arguments Passed and their description:
- **arg1** String - extension number to be dialed
- **arg2** String - call Type (whether "video" or "audio" call)

Return type: Object
This return object represents the current call session's call object using which below subsequent APIs can be invoked to retrieve information's at different phases of the call.

1. getCallId() - Returns the unique call id used in this particular call session

2. getCallState() - Returns the current call state

3. getFarEndNumber() - Returns the Far End's number

4. getFarEndName() – Returns the Far End's full-name

5. getSipUri() - Returns the SipUri in string format
   Example of return string format:

"sip:4001@192.0.2.0:5060"

6. getSubject() – Returns the subject of the call if provided in the meeting

7. isAutoAnswer() - indicates whether the call is auto answered(true) and this is meaningful only when the call object acts as recipient

8. isVideoCall() - indicates whether the call is video type(true) or not(false). This could be useful particularly when there is an incoming call.

Note: The farEndName returned via getFarEndName() API will henceforth not contain the subject of the call if any, as it can be obtained using getSubject() API.

When resiliency is supported and enabled , if the client is either in failing over, failing back or reconnecting state, then call will not be made and make Call API returns null

Note:

SDK waits for completion of gathering ICE candidates in all the network interfaces or a timeout of 10 seconds to initiate call. Applications shall provide progress indication to User after invoking makeCall (arg1, arg2)

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

answerCall (arg1)

This API Answers an incoming call by creating WebRTC PeerConnection, answer SDP generation and other signaling messages. This API takes care of answering the call either as an audio/video call based on the incoming call type and whether video is enabled locally using setDomElements API.

Arguments Passed and their description:
- **arg1** String - callId parameter(obtained through the callback functions 'onNewIncomingCall' or 'onCallStateChange')

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

rejectCall (arg1)

This API is to reject an incoming call and also resets the call session object properties.

Arguments Passed and their description:
- **arg1** String - callId parameter(obtained through the callback functions 'onNewIncomingCall' or 'onCallStateChange')

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

dropCall (arg1)

This API disconnects a connected existing audio/video call and resets the call session object properties.

Arguments Passed and their description:

▪ **arg1** String - callId parameter(obtained through the callback functions 'onNewIncomingCall' or 'onCallStateChange')

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

cancelCall (arg1)

This API is to cancel a dialed call before the call is answered and also resets the call session object properties

Arguments Passed and their description:

▪ **arg1** String - callId parameter(this can be obtained using the call object which is returned with makeCall API call)

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

doHold (arg1)

This API pushes a call to hold state for an audio/video call.

Arguments Passed and their description:

▪ **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

doUnHold (arg1)

This API retrieves a call from held to active state for an audio/video call.

Arguments Passed and their description:

▪ **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

doMute (arg1)

This API pushes the WebRTC mic to mute state.

Arguments Passed and their description:

▪   **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Boolean

True indicates the API call is successful.

False indicates the API call failed

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

doUnMute (arg1)

This API pushes the WebRTC mic to unmute state from mute condition.

Arguments Passed and their description:

▪   **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Boolean

True indicates the API call is successful.

False indicates the API call failed

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

sendDTMF (arg1 , arg2)

This API generates a DTMF Tone based on 'arg2' value and sends it across to the connected peer. DTMF Tones will be generated only when DTMF support is negotiated during a call (i.e., Between Browser and the peer endpoint) and only Chrome browser (latest version) supports DTMF tone generation.

Arguments Passed and their description:

▪   **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

▪   **arg2** Char - It should be one of possible ITU-T supported DTMF Tones(which are '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '*', '#').

Return type: None

Note

sendDTMF() API is not supported in Firefox browser

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

transferCall (arg1 , arg2 , arg3)

This API is used to transfer call by dialing out a new call or merging two existing calls.

Arguments Passed and their description:

- **arg1** String - extension or callId of the call to be transferred

- **arg2** String - CallId of the existing call

- **arg3** String - type of transfer (attended/Unattended)

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getStats (arg1)

This API is used to get the Call-Statistics of that particular call which can be accessed using the returned array reference.

Arguments Passed and their description:

- **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Array

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

addVideo (arg1)

This API is used to upgrade an ongoing audio call stream to an audio+video stream.

Arguments Passed and their description:

- **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Boolean

Note: addVideo functionality is available in Chrome Browser version 64 or later; it is not supported with Firefox.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

removeVideo (arg1)

This API is used to downgrade an ongoing audio+video call stream to an audio only stream.

Arguments Passed and their description:

- **arg1** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

Return type: Boolean

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

setMediaStream (arg1 , arg2 , arg3 , arg4)

This API is to dynamically attach the video media streams with HTML5 video media Elements provided by the application. All the parameters are mandatory. It is mandatory for the application to call this API if DOM elements are not initially set using setDomElements API.

Arguments Passed and their description:

- **arg1** String - domElement parameter is HTML5 video media Element's ID or HTML5 video media element to which the stream has to be attached

- **arg2** String - Stream parameter which is obtained by the call object which is returned with makeCall API or using the stream parameter obtained by the call state change callback function 'onVideoStreamsAvailable'

- **arg3** String - callId parameter(obtained by the call state change callback functions 'onNewIncomingCall' or 'onCallStateChange' or using the call object which is returned with makeCall API call)

- **arg4** String - domType parameter which indicates the stream type i.e. "localVideo" or "remoteVideo"

Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

Network Test API

createLoopBackConnection ()

This API serves for network stability assessment and is accomplished by creating a WebRTC loopback connection between the browser and the WebRTC gateway for audio and data (if respective flags are set to TRUE) streams separately. This provides an easy way to assess the ability of customer network by doing on demand network assessment as well as a long lived, constant check of the network.

Arguments Passed and their description: None
Return type: None

```
Example:

    myWebRTC.createLoopBackConnection();
```

Note: The createLoopBackConnection API call invokes the callback function which was passed as 'callback_onCallStateChanged' earlier in setConfiguration API with result(either 'CONST.AWL_MSG_LOOPBACK_CONN_SUCCESSFULL' or 'CONST.AWL_MSG_LOOPBACK_CONN_FAILED' or 'AWL_MSG_LOOPBACK_CONN_LINK_ISSUE' and reason associated with it.

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

endLoopBackConnection ()

This API terminates the network assessment loop back connection towards the WebRTC gateway.

Arguments Passed and their description: None
Return type: None

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

getLoopBackStats ()

This API generates and updates the below statistical parameters based on which network analyzer application shall determine the network stability. These parameters are accessible using the returned array reference.

1. 'nowPLoss' - packet lost count in each second

2. 'totPSent' - Total Packets Sent

3. 'totPLost' - Total Packets Lost

4. 'nowRTT' - Current Round Trip delay(RTT)

5. 'totPLossPercent' - Percentage of packet lost so far

6. 'maxRTT' - Peak Round trip delay

7. 'minRTT' - Min Round trip delay

8. 'avgRTT' - Average Round trip delay

9. 'totJitter' - Jitter Received (total)

10. 'nowPLossPercent' - Observed packet Loss percentage in each second

Arguments Passed and their description: None
Return type: Array

IP Office Releases	10.1	11.0	11.1
API Support	☐	☐	☐

3 Properties

Below are the possible resultant (string constants – response .result) properties that could be passed in any of the call-back function events

AWL_MSG_CALL_CONNECTED String   – Indicates that the call is currently in CONNECTED state

AWL_MSG_CALL_DISCONNECTED String   - Indicates that the call is currently in DISCONNECTED state

AWL_MSG_CALL_FAILED String   - Indicates the call has FAILED for some reason

AWL_MSG_CALL_FAREND_UPDATE String   - Indicates that during a call, far end's number is UPDATED

AWL_MSG_CALL_HELD String   - Indicates that the call is currently in HELD state

AWL_MSG_CALL_IDLE String   - Indicates that a call object current state is IDLE and is available for dialing outgoing call or receiving an incoming call

AWL_MSG_CALL_INCOMING String   - Indicates an INCOMING call state

AWL_MSG_CALL_MAXCAP_REACHED String   - Indicates that the current call is denied due to maximum allowed concurrent call limit of three is reached already

AWL_MSG_CALL_PROGRESSING String   - Indicates that the call is progressing with early media. When early media received from gateway, call is connected temporarily and User may need to enter account code in this state.

AWL_MSG_CALL_RINGING String   - Indicates that the far end is in RINGING state

AWL_MSG_CALL_TRANSFER_FAILED String   - Indicates that the call TRANSFER FAILED and Existing call connection will be retained. This is same as AWL_MSG_CALL_CONNECTED state except the transfer failure notification.

AWL_MSG_DEVICEACCESS_FAILURE String   - Indicates logIn (arg1, arg2) API call failure due to audio (Mic) or video (Camera) device access failure and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_DUPLICATE_LOGIN String   - Indicates a duplicate logIn (arg1, arg2) request is received when the SIP extension is already registered and this could be a possible result value of the response object (i.e., resp.result) passed in the registration state change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_FAIL_BACK_FAILED String   - indicates that fail-back is not successful and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_FAIL_BACK_SUCCESS String   - indicates that client has successfully failed back and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_FAIL_OVER_FAILED String   - indicates that fail-over is not successful and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'

AWL_MSG_FAIL_OVER_SUCCESS String   - indicates that client has successfully failed over and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'

AWL_MSG_FAILING_BACK String   - indicates that client is failing back to alternate server and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'

AWL_MSG_FAILING_OVER String   - indicates that client is failing over to alternate server and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LINK_ISSUE_DETECTED String   - Indicates there is a link issue between the SDK and WebRTC Gateway (i.e., no keep alive message received for about 180seconds from WebRTC Gateway) and this could be a possible result value of the response object (i.e., resp.result) passed in the registration state change callback function 'callback_onRegistrationStateChanged'. This shall be further used in UI to display link issue and also let take corrective action.

AWL_MSG_LOGGEDOUT String   - Indicates logOut () API call is successful and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_FAILED String   - Indicates logIn (arg1, arg2) API call failure and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_GW_NOTCONFIGURED String   - Indicates logIn(arg1, arg2) API call failure due to gateway IP address not configured correctly and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_EMPTYPASSWORD String   - Indicates logIn(arg1, arg2) API call failure due to empty password and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_EMPTYTOKEN String   - Indicates tokenLogIn(arg1, arg2, arg3) API call failure due to empty token and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_EMPTYUSERNAME String   - Indicates logIn(arg1, arg2) or tokenLogIn(arg1, arg2, arg3) API call failure due to empty username and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_INVALID_TOKENTYPE String   - Indicates tokenLogIn(arg1, arg2, arg3) API call failure due to invalid token type and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'
AWL_MSG_LOGIN_INVALID_TOKENTYPE

AWL_MSG_LOGIN_SUCCESS String   - Indicates logIn (arg1, arg2) API call is successful and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOGIN_WEBSOCKET_FAILURE String   - Indicates logIn(arg1, arg2) API call failure or registration failure due to web socket connectivity failure and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_LOOPBACK_CONN_CLOSED String   - Indicates that the loop back is closed

AWL_MSG_LOOPBACK_CONN_FAILED String   - Indicates that the loop back connection to WebRTC gateway failed

AWL_MSG_LOOPBACK_CONN_LINK_ISSUE String   - Indicates that a established loop back connection has link issue detected

AWL_MSG_LOOPBACK_CONN_SUCCESSFULL String   - Indicates that the loop back connection to WebRTC gateway is created successfully

AWL_MSG_LOOPBACK_STATS_FAILURE String   - Indicates failure while fetching stats on the loop back connection

AWL_MSG_RECONNECTING String   - Indicates that connection to the server is lost and the client is attempting to reconnect to the server. This could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'

AWL_MSG_RELOGGED_IN String   - Indicates that client is re-logged in to the server and this could be a possible result value of the response object(i.e., resp.result) passed in the configuration change callback function 'callback_onRegistrationStateChanged'.

AWL_MSG_SETCONFIG_FAILED String   - Indicates API call of setConfiguration (arg1, arg2, arg3, arg4) has failed and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onConfigChanged'.

AWL_MSG_SETCONFIG_SUCCESS String   - Indicates API call of setConfiguration (arg1, arg2, arg3, arg4) is successful and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onConfigChanged'.

AWL_MSG_SETDOM_FAILED String   - Indicates API call of setDomElements (arg1) has failed and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onConfigChanged'.

AWL_MSG_SETDOM_SUCCESS String   - Indicates API call of setDomElements (arg1) is successful and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onConfigChanged'.

AWL_MSG_TOKEN_RENEW_FAILED String   - Indicates failure while renewing the authentication token and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onAuthTokenRenewed'.

AWL_MSG_TOKEN_RENEW_SUCCESS String   - Indicates that authentication token is successfully renewed and this could be a possible result value of the response object (i.e., resp.result) passed in the configuration change callback function 'callback_onAuthTokenRenewed'.

AWL_MSG_WEBRTC_NOTSUPPORTED String   - Indicates that the browser does not support WebRTC API and this could be a possible return value for isWebRTCSupported API call

AWL_MSG_WEBRTC_SUPPORTED String   - Indicates that the browser does support WebRTC API and this could be a possible return value for isWebRTCSupported API call

Below are the string constants that could be passed as arguments in tokenLogIn API:

RESILIENCY String   - This could be a possible argument value of the authentication type (i.e., authType) passed in the tokenLogIn(arg1, arg2, arg3).

ESNA String   - This could be a possible argument value of the authentication type (i.e., authType) passed in the tokenLogIn(arg1, arg2, arg3) API.

AWL_MSG_SET_STUN_CONFIG_FAILED String   - Indicates API call of setStunConfiguration (arg1) has failed.

AWL_MSG_SET_STUN_CONFIG_SUCCESS String   - Indicates API call of setStunConfiguration (arg1) is successful.

3   Events

callback_onConfigChanged

This callback function parameter (as arg2 parameter) used in **setConfiguration** API and this has to be implemented at the application level. This would be invoked whenever configuration change or DOM Element change is attempted using $setConfiguration$ API and this callback function will be triggered with response object(arg1) containing a result(arg1.result) and reason(arg1.reason) for the configuration change attempted with $setConfiguration$ API call. Possible result (string constants - resp.result) that could be passed in this callback function is

```
AWL_MSG_SETCONFIG_SUCCESS
AWL_MSG_SETCONFIG_FAILED
AWL_MSG_SETDOM_SUCCESS
AWL_MSG_SETDOM_FAILED
```

Below example is a sample implementation of '$callback\_onConfigChanged$'(i.e., onConfigChanged)' function at the application level:

```
function onConfigChanged (resp) {
    console.log('\n onConfigChanged :: RESULT = ' + resp.result);
    console.log('\n onConfigChanged :: reason = ' + resp.reason);
}
```

Event Payload:

resp Object - This object can be used to retrieve result and reason properties further. All the properties are string constants.

```
Example:
    var resp = {
        result: "",
        reason: ""
    };
```

callback_onRegistrationStateChanged

This callback function is the parameter (as arg3 parameter) used in **setConfiguration** API and this has to be implemented at the application level. This callback function will be triggered whenever there's a change in the registration/un-registration state or when there is a web socket connectivity failure. The response object (arg1) would contain a result (arg1.result) and reason (arg1.reason). When resiliency is supported and enabled, the response object would also contain authentication token (arg1.authToken) on successful login/fail-over/fail-back. Possible result (string constants – response result) that could be passed in this callback function is

```
        AWL_MSG_LOGIN_EMPTYUSERNAME
        AWL_MSG_LOGIN_EMPTYPASSWORD
        AWL_MSG_LOGIN_EMPTYTOKEN
        AWL_MSG_LOGIN_GW_NOTCONFIGURED
        AWL_MSG_LOGIN_SUCCESS
        AWL_MSG_LOGIN_FAILED
        AWL_MSG_LOGIN_WEBSOCKET_FAILURE
        AWL_MSG_LINK_ISSUE_DETECTED
        AWL_MSG_DEVICEACCESS_FAILURE
        AWL_MSG_LOGGEDOUT
        AWL_MSG_FAILING_OVER
        AWL_MSG_FAILING_BACK
        AWL_MSG_FAIL_OVER_SUCCESS
        AWL_MSG_FAIL_BACK_SUCCESS
        AWL_MSG_FAIL_OVER_FAILED
        AWL_MSG_FAIL_BACK_FAILED
        AWL_MSG_RECONNECTING
        AWL_MSG_RELOGGED_IN
        AWL_MSG_LOGIN_INVALID_TOKENTYPE
```

Below example is a sample implementation of 'callback_onRegistrationStateChanged' (i.e., onRegistrationStateChanged) function at the application level:

```
function onRegistrationStateChanged(resp){
    console.log('\n onRegistrationStateChange :: RESULT = ' + resp.result);
    console.log('\n onRegistrationStateChange :: reason = ' + resp.reason);
    if(resp.result === "AWL_MSG_LOGIN_SUCCESS") {
        var authToken = resp.authToken.token;
        var expiryTime = resp.authToken.expiry;
        //logic code
    }
    else{
         //logic code
    }
}
```

Event Payload:

▪ **resp** Object - This object could be used to retrieve result and reason properties further. These two properties are string constants. When resiliency is supported and enabled, the resp object will have authToken property on successful login/failover/failback. 'authToken' is an object which can be used to retrieve token and expiry properties.

callback_onCallStateChanged

This is the callback function's instance that is used in setConfiguration API (as arg4 parameter) and this has to be implemented at the application level. This callback function's instance will be used to invoke onNewIncomingCall function by the SDK itself whenever there is a new incoming call or if any change in the call state happens the SDK will invoke onCallStateChange function. This implementation has to follow the revealing Model Pattern as in the below example implementation to implement the following functions.

```
1. onNewIncomingCall(arg1, arg2, arg3)
2. onCallStateChange(arg1, arg2, arg3)
3. onCallTerminate(arg1, arg2)
4. onLoopBackNotification(arg1)
5. onVideoStreamsAvailable(arg1, arg2, arg3)
5. onAudioStreamsAvailable(arg1, arg2, arg3)
```

onNewIncomingCall(arg1, arg2, arg3)

This application level implemented function would be invoked by the SDK whenever there's an incoming call to this client and this function's parameters are described below.

- **arg1** - this is the unique call ID used in the current call session. This has to be stored at the application level in order to track a call session and this has to be passed as an argument in all the telephony APIs (except $makeCall$ API) further.

- **arg2** - this is the call object used in the current call session. With this, application can get additional call information like call ID, current call state, far end's full-name, far end's number, subject of the call or subject of the meeting if any, whether the call is auto answered by the SDK itself and whether a video call is attempted or received with below APIs respectively.

```
1. getCallId()
2. getCallState()
3. getFarEndNumber()
4. getFarEndName()
5. getSipUri()
6. getSubject()
7. isAutoAnswer()
8. isVideoCall()
```

- **arg3** - this is the autoAnswer flag, if setup to true indicates the call is auto answered by the SDK itself and if set to false, it is left to the application to take the control of the call session further.

onCallStateChange(callId, callObject, currentCallState)

This application level implemented function would be invoked by the SDK whenever there's a change in the call state and this function's parameters are described below.

- arg1 - this is the unique call ID used in the current call session. This has to be stored at the application level in order to track a call session and this has to be passed as an argument in all the telephony API's (except $makeCall$ API) further.

- arg2 - this is the call object used in the current call session. With this, application can get additional call information like call ID, current call state, far end's full-name, far end's number, subject of the call or subject of the meeting if any, whether the call is auto

answered by the SDK itself and whether a video call is attempted or received with below API's respectively.

```
1. getCallId()
2. getCallState()
3. getFarEndNumber()
4. getFarEndName()
5. getSipUri()
6. getSubject()
7. isAutoAnswer()
8. isVideoCall()
```

- arg3 - this reflects the current call state of that particular call.

Below are possible call state values (String constant) that this parameter 'arg3' can hold:

```
AWL_MSG_CALL_IDLE
AWL_MSG_CALL_CONNECTED
AWL_MSG_CALL_DISCONNECTED
AWL_MSG_CALL_FAILED
AWL_MSG_CALL_INCOMING
AWL_MSG_CALL_RINGING
AWL_MSG_CALL_HELD
AWL_MSG_CALL_FAREND_UPDATE
AWL_MSG_CALL_MAXCAP_REACHED
```

onCallTerminate(arg1, arg2)

This application level implemented function would be invoked by the SDK whenever a call disconnects and this function takes following parameters:

- arg1 - this is the call ID of the object used in the current call session.

- arg2 - string indicating the call termination reason.

Application shall clean up objects for this call after receiving this onCallTerminate event .

onLoopBackNotification(arg1)
This application level implemented function would be invoked by the SDK during relay service and notifies any of the below predefined alarms (string) arg1 - this is a notification alarm parameter. With this, application can get information about the loopback connection status. Below are possible predefined alarm values (String constant) that this parameter 'arg1' can hold: AWL_MSG_LOOPBACK_CONN_SUCCESSFULL
AWL_MSG_LOOPBACK_CONN_FAILED
AWL_MSG_LOOPBACK_CONN_LINK_ISSUE
AWL_MSG_LOOPBACK_CONN_CLOSED
AWL_MSG_LOOPBACK_STATS_FAILURE

onVideoStreamsAvailable(callId, localStream, remoteStream)

This application level implemented function will be invoked by the SDK whenever local and remote video streams are available and the application has registered for this event. This callback function is invoked only once per call i.e. when the call is connected for the first time. After retrieving the media streams, it is mandatory for the application to attach the streams to the respective DOM elements using setMediaStream API. This event is not triggered if setDomElements API is called initially as WebRTC video streams of local and remote video would be attached to the respective DOM elements (set in setDomElements API) by the SDK. The application should register to this callback if it requires dynamic video stream control to attach it to the DOM element on fly i.e., DOM elements need not be fixed across multiple calls to show local and remote video streams. But if the application requires a pair of DOM elements(local & remote) to be fixed across multiple calls i.e., to have the active call's stream to be attached to the DOM element at any point of call, then the application shall use *setDomElements* API to set the HTML5 video media DOM elements.

- callId - this is the call id of the call object used in the current call session

- localStream - local video stream

- remoteStream - remote video stream

onAudioStreamsAvailable(callId, localStream, remoteStream)

This application level implemented function will be invoked by the SDK whenever local and remote audio streams are available and the application has registered for this event. The audio streams are already attached to the DOM elements by the SDK. This callback is provided for notification purpose only and not for audio play. This callback function is invoked only once per call i.e., when the call is connected for the first time.

- callId - this is the call id of the call object used in the current call session

- localStream - local audio stream

- remoteStream - remote audio stream

Below example is a sample implementation of 'callback_onCallStateChanged(i.e., onCallListener)' function at the application level:

Example:

```
    var CallListener = function () {
        var _onNewIncomingCall = function (callId, callObj, autoAnswer) {
            console.log("onNewIncomingCall : getFarEndNumber =
"+callObj.getFarEndNumber());
            console.log("onNewIncomingCall : getSipUri = "+callObj.getSipUri());
            console.log("onNewIncomingCall : autoAnswer = "+autoAnswer);
```

```
            if (callObj1 === null) {
                callObj1 = callObj;
                callmap[callObj1.getCallId()] = callObj1;
            }
        };

        var _onCallStateChange = function (callId, callObj, event) {
            if (typeof(callmap[callObj.getCallId()]) === 'undefined') {
                if (callObj1 === null) {
                    callObj1 = callObj;
                    callmap[callObj1.getCallId()] = callObj1;
                }
            }
            if (callObj.getCallId() === callObj1.getCallId()) {
                switch (callObj1.getCallState()) {
                    case "AWL_MSG_CALL_IDLE":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_CONNECTED":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_RINGING":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_DISCONNECTED":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_FAILED":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_INCOMING":
                    // state specific handling code
                    break;
                    case "AWL_MSG_CALL_HELD":
                    // state specific handling code
                    case "AWL_MSG_CALL_FAREND_UPDATE":
                    // state specific handling code (For example update far end DN
and sipUri information)
                    break;
                    default:
                }
            }
        }

        var _onCallTerminate = function(callId, reason){
            //application logic to display call terminate reason
        }

        var _onLoopBackNotification = function(notification){
            //application logic to handle notifications and alarms from the loopback
connection
        }

        var _onVideoStreamsAvailable = function(callId, localStream, remoteStream){
```

```
            //application logic to handle the retrieved streams
        }

        var _onAudioStreamsAvailable = function(callId, localStream, remoteStream){
            //application logic to handle the retrieved streams
        }

        return{
            onNewIncomingCall: _onNewIncomingCall,
            onCallStateChange: _onCallStateChange,
            onCallTerminate: _onCallTerminate,
            onLoopBackNotification: _onLoopBackNotification,
            onVideoStreamsAvailable: _onVideoStreamsAvailable,
            onAudioStreamsAvailable: _onAudioStreamsAvailable
        };

    };

    var onCallListener = new CallListener();
```

callback_onDeviceListRequested

This callback function parameter (as arg1 parameter) is used in getDeviceList API and this has
to be implemented at the application level. This would be invoked whenever information of all
the devices is found as requested using getDeviceList API and this callback function will be
triggered with response object(arg1) containing a list of media devices and their information i.e.
ID and label of all the media devices.

Below example is a sample implementation of 'callback_onDeviceListRequested ' (i.e.,
onDeviceListRequested)' function at the application level:

Example:

```
    function onDeviceListRequested(deviceList){
        if(deviceList.length !== 0){
            $.each(deviceList, function (index, value) {
                if(value[0] === "audioinput"){
                    //logic code
                }else if(value[0] === "videoinput"){
                    //logic code
                }else if(value[0] === "audiooutput"){
                    //logic code
                }
            });
        }
    }
```

Event Payload:

resp Object - This object could be used to retrieve the ID and label of all the
media devices.

callback_onAuthTokenRenewed

This callback function parameter (as arg5 parameter) used in setConfiguration API and this has to be implemented at the application level. This would be invoked when resiliency is enabled (resiliency support is enabled at server and not disabled by the client application) and renewal of authentication token succeeds or fails. If the token renewal fails, then auto login using token will not be supported during failover and failback and the applications have to either relogin or go for manual login during failover and failback. This callback function will be triggered with response object(arg1) containing a result(arg1.result) and reason(arg1.reason). The response object will also contain the authentication token(arg1.authToken) when the token renewal succeeds. The authentication token is an object containing the token(arg1.authToken.token) and its expiration time(arg1.authToken.expiry). Possible result (string constants - resp.result) that could be passed in this callback function is

```
AWL_MSG_TOKEN_RENEW_SUCCESS
AWL_MSG_TOKEN_RENEW_FAILED
```

Below example is a sample implementation of 'callback_onAuthTokenRenewed' (i.e., onAuthTokenRenewed) function at the application level:

```javascript
function onAuthTokenRenewed(resp){
        console.log('\n onAuthTokenRenewed :: RESULT = ' + resp.result);
        console.log('\n onAuthTokenRenewed :: reason = ' + resp.reason);
        if(resp.result === "AWL_MSG_TOKEN_RENEW_SUCCESS"){
            if(typeof(resp.authToken)!=="undefined" && resp.authToken!== null){
                var token = resp.authToken.token;
                var expiry = resp.authToken.expiry;
                //logic code
            }
        }else{
            //logic code
        }
    }
```

Event Payload:

- **resp** Object - This object could be used to retrieve result and reason properties further. These two properties are string constants. When token renewal is successful, this response object will also contain 'authToken' property. 'authToken' is an object which can be used to retrieve token and expiry properties.

4 Terms and Acronyms

Acronyms	Description
API	Application Programming Interface
CA	Certificate Authority
CTI	Computer Telephony Integration
DN	Directory Number
DTMF	Dual-Tone Multi-Frequency
HTTP	Hyper Text Transfer Protocol
IPO	IP Office
ITU-T	International Telecommunication Union - Telecommunications section
RTT	Round Trip Time
SDK	Software Development Kit
SDP	Session Description Protocol
UDP	User Datagram Protocol
UI	User Interface
URL	Uniform Resource Locator
DTLS	Datagram Transport Layer Security
WebRTC	Web Real-Time Communication