API Reference Guide

AudioCodes WebRTC Solutions for Enterprises

WebRTC Android Client SDK

Version 2.6.0

audiocodes

Table of Contents

Notice ..... **vi**
Security Vulnerabilities vi
WEEE EU Directive vi
Customer Support vi
Stay in the Loop with AudioCodes vi
Abbreviations and Terminology vi
Related Documentation vi
Document Revision Record vii
Documentation Feedback viii
1 Introduction **1**
1.1 Purpose 1
1.2 Scope 1
1.3 Benefits 1
2 Android SDK **2**
2.1 Before you Start 2
2.2 Android SDK and Demo Client Supported Versions 2
2.3 Supported Architectures 2
2.4 Installation 3
3 API Classes **5**
3.1 AudioCodesUA 6
3.1.1 Standard Methods 7
3.1.1.1 getInstance 7
3.1.1.2 setServerConfig 7
3.1.1.3 setAccount (1) 7
3.1.1.4 setAccount (2) 8
3.1.1.5 setListeners 8
3.1.1.6 login 8
3.1.1.7 login (click to call) 9
3.1.1.8 logout 9
3.1.1.9 call 9
3.1.1.10 setVerifyServer 10
3.1.1.11 setVerifyServer 10
3.1.2 Advanced Methods 11
3.1.2.1 setRegisterExtraHeaders 11
3.1.2.2 setInviteExtraHeaders 11
3.1.2.3 getUserAgent 11
3.1.2.4 setUserAgent 12
3.1.2.5 getRegExpires 12
3.1.2.6 setRegExpires 12

3.1.2.7 setUseSessionTimer 12
3.1.2.8 setLogLevel 13
3.1.2.9 setLogger 13
3.1.2.10 handleNetworkChange 13
3.1.2.11 getSessionList 13
3.1.2.12 List of Sessions [ArrayList<AudiocodesSession>, List of Active Sessions]disconnectOnBrokenConnection 14
3.1.2.13 setContactRewrite 14
3.1.2.14 setOauthToken 14
3.1.2.15 setPushToken 15
3.1.2.16 updatePushToken 15
3.1.2.17 sendInstantMessage 15
3.1.2.18 setAllowHeader 16
3.1.2.19 setDynamicPushNotifications 16
3.2 AudioCodesSession 17
3.2.1 Standard Methods 18
3.2.1.1 getSessionID 18
3.2.1.2 answer 18
3.2.1.3 reject 18
3.2.1.4 Terminate 19
3.2.1.5 muteAudio 19
3.2.1.6 muteVideo 19
3.2.1.7 isAudioMuted 19
3.2.1.8 isVideoMuted 20
3.2.1.9 sendDTMF 20
3.2.1.10 sendInfo 20
3.2.1.11 isOutgoing 20
3.2.1.12 hasVideo 21
3.2.1.13 getCallState 21
3.2.1.14 getTermination 21
3.2.1.15 duration 21
3.2.1.16 isLocalHold 22
3.2.1.17 isRemoteHold 22
3.2.1.18 setUserData 22
3.2.1.19 getUserData 22
3.2.1.20 hold 23
3.2.1.21 switchCamera 23
3.2.1.22 showVideo (1) 23
3.2.1.23 showVideo (2) 24
3.2.1.24 stopVideo 24
3.2.1.25 setLocalRenderPosition 24
3.2.1.26 addSessionEventListener 25

3.2.1.27 removeSessionEventListener 25
3.2.1.28 getStats 25
3.2.1.29 redirect 25
3.2.1.30 reinviteWithVideo 26
3.2.1.31 transferCall (blind transfer) 26
3.2.1.32 transferCall (attended transfer) 26
3.2.1.33 getTransferContact 27
3.2.1.34 getTransferState 27
3.3 WebRTCAudioManager 28
3.3.1 Standard Methods 28
3.3.1.1 getInstance 28
3.3.1.2 setWebRTCAudioRouteListener 28
3.3.1.3 setAudioRoute 29
3.3.1.4 getAudioRoute 29
3.3.1.5 getAvailableAudioRoutes 29
3.4 ACConfiguration 30
3.4.1 Standard Methods 30
3.4.1.1 getConfiguration 30
3.4.1.2 version 30
3.4.1.3 getLocalServerPort 31
3.4.1.4 setLocalServerPort 31
3.4.1.5 getDtmfOptions 31
3.4.1.6 setDtmfOptions 32
3.4.1.7 getVideoConfiguration 32
3.4.1.8 setVideoConfiguration 32
3.4.1.9 setAutomaticCallOnRedirect 33
3.4.1.10 getAutomaticCallOnRedirect 33
3.4.1.11 setRedirect 33
3.4.1.12 getRedirectContact 34
3.4.1.13 getRedirectEnabled 34
3.5 VideoConfiguration 34
3.5.1 Camera Parameters 34
3.5.2 Rendering Views Parameters 35
3.6 DTMFOptions 35
3.6.1 DTMF Parameters 35
3.7 RemoteContact 36
3.7.1 Standard Methods 36
3.7.1.1 getDisplayName 36
3.7.1.2 getUserName 36
3.7.1.3 getDomain 36
3.7.1.4 setDisplayName 37
3.7.1.5 setUsername 37

3.7.1.6 setDomain 37
3.8 TerminationInfo 38
3.8.1 TerminationInfo attribute 38
3.9 InfoAlert 38
3.9.1 InfoAlert attribute 38
4 API Callbacks/ Listeners Interfaces 39
4.1 AudioCodesEventListener 39
4.1.1 Login state changed event 39
4.1.2 Incoming call event 39
4.1.3 Incoming IM message event 39
4.1.4 IM Message status event 40
4.2 AudioCodesSessionEventListener 40
4.2.1 callTerminated 40
4.2.2 callProgress 40
4.2.3 cameraSwitched 41
4.2.4 reinviteWithvideoCallback 41
4.2.5 mediaFailed 41
4.2.6 incoming Notify 42
4.3 WebRTCAudioRouteListener 42
4.3.1 audioRoutesChanged 42
4.3.2 currentAudioRouteChanged 43
5 Use Examples 44
5.1 User Agent: Create Instance, Set Server and Account 44
5.2 User Agent: Set Listeners (Callbacks) 44
5.3 User Agent login: Connection to SBC Server and Login 44
5.4 Make a Call 44
5.5 Send DTMF During a Call 44
5.6 Send SIP Message During a Call 44
5.7 Mute / Unmute During a Call 44
5.8 Accept Incoming Call (with video) 45
5.9 Reject Incoming Call 45
5.10 Terminate a Call 45
5.11 Use of Video 45

Notice

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

This document is subject to change without notice.

Date Published: November-19-2026

Security Vulnerabilities

All security vulnerabilities should be reported to vulnerability@audiocodes.com.

WEEE EU Directive

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

Customer Support

Customer technical support and services are provided by AudioCodes or by an authorized AudioCodes Service Partner. For more information on how to buy technical support for AudioCodes products and for contact information, please visit our website at

https://www.audiocodes.com/services-support/maintenance-and-support.

Stay in the Loop with AudioCodes



Abbreviations and Terminology

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

Related Documentation

Document Name
https://www.audiocodes.com/solutions-products/solutions/enterprise-voice/webrtc-connectivity
WebRTC Client SDK API Reference Guide
WebRTC Click-to-Call Widget Installation and Configuration Guide

Document Revision Record

LTRT	Description
14060	■ Initial document release for Version 1.0
14061	■ Added functionality for SDK Version 1.0.1
14062	■ Updated to software Version 1.0.6 ■ Update for new function “setContactRewrite”
14063	■ Updated to software Version 1.1.0. ■ Blind transfer: • New function - transferCall(RemoteContact transferTo) • New function - getTransferContact • New function - getTransferState ■ Attended transfer - New function - transferCall(AudioCodesSession transferToSession) ■ OAuth authorization - New function setOauthToken ■ 64 bit support (arm64-v8a) ■ IPv6 support ■ Click to call support (calls without registration) - Added new function - login(Context, Boolean autologin) ■ Push support: • New function - setPushToken • New function - updatePushToken ■ Support for Google WebRTC SDK 1.0.26885 ■ Support for delayed offer
14064	■ Updated software to Version 1.1.2 ■ Support for SIP IM Messaging
14065	■ Updated software to Version 1.1.5 ■ Support for Android API Level 29 ■ Support for AndroidX ■ Support for continuing call on broken connection
14066	■ Updated software to Version 1.1.13 ■ Added TerminationInfo parameter in callTerminated callback in AudioCodesEventListener ■ Added mediaFailed callback in AudiocodesSessionEventListener ■ Support for verify server over TLS
14067	■ Updated software to Version 1.1.16 ■ Added InfoAlert parameter in incomingCall callback in AudiocodesSessionEventListener ■ Added incomingNotify callback in AudioCodesSessionEventListener
14068	■ Added support for Android tablet devices
14069	■ Updated software to Version 1.2.1 ■ Updated SIP stack to pjsip2.10 ■ Support for Android API Level 31
14070	■ Updated software to Version 1.2.2 ■ Added SIP message support
14071	■ Updated software to Version 2.0.1 ■ Updated SIP stack to PJSIP 2.13 ■ Updated WebRTC library to M123 (branch 6312) ■ Updated OpenSSL to 3.2.0 ■ Updated setAccount description ■ Added permission description

LTRT	Description
14072	■ Updated software to Version 2.1.0 ■ Support for Android 15 • SDK version 35 • Gradle version 8.7.3 • Kotlin version 1.9.0 • Android 16KB memory page is NOT supported in 2.1.0 ■ Support for Android emulator • x86_64 libraries added • abiFilters are set to 'arm64-v8a', 'armeabi-v7a', 'x86_64' – x86_64 is optional and can be removed from the application build gradle file • The Android SDK size has increased significantly because of these added files, but the x86_64 can be excluded from the actual build by using abiFilters. ■ Updated server verification process • Support for Android CA Keystore • Support for custom certificate list (supplied as object) • See setVerifyServer(boolean verifyServer, boolean useAndroidKeystore, X509Certificate[] customCertificateList) ■ Fix for armeabi-v7a support ■ Call handover fix
14073	■ Support for 16KB memory page size • Upgraded to OpenSSL 3.4.2 • Upgraded WebRTC to M136 (branch 7103) • Upgraded to Android NDK 28c • Upgraded to Android SDK 35 ■ Support for Android SDK obfuscation – when obfuscation is enabled in the application that uses the SDK, the SDK uses its own set of obfuscation rules.
14074	■ Updated software to Version 2.2.1 ■ Fix message id mismatch on 32-bit devices ■ Added randomization of local SIP port ■ Fix crash during active call on 32-bit devices
14075	■ Fix username when transferring call ■ Fix early reject of call in DemoClient app
14077	■ Support for Android 16 • SDK version 36 • Gradle version 9.4.1 ■ Fixed setRegisterExtraHeaders(HashTable<String, String> headers) method ■ Added dynamic push notification registration mode • see method: setDynamicPushNotifications(Boolean dynamicPushNotifications) ■ Added dynamic configuration for setting PeerConnection.KeyType • see method: setServerConfig(String proxyAddress, int port, String domain, Transport transport, List<PeerConnection.IceServer> iceServerList, PeerConnection.KeyType keyType) ■ Fixed autoLogin functionality • see method: login(Context context, Boolean autoLogin) • register request is sent immediately if autoLogin is set to true ■ Support for optional Bluetooth permission
14078	■ Fixed crash in method: handleNetworkChange() when changing device's network during active call
14079	■ Fixed TLS handshake verification at the SDK layer without requiring the integrating application to provide the whole chain. ■ Fixed SDK crashing when the network connection is lost during the early call setup phase. ■ Fixed SDK crashing on retry when making a call while SBC is unavailable. ■ Fixed SDK crashing when the internet connection drops during outgoing call setup.
14098	■ Upgrade PJSIP to version 2.13.1 ■ Fixed error handling in PJSIP which caused application crash

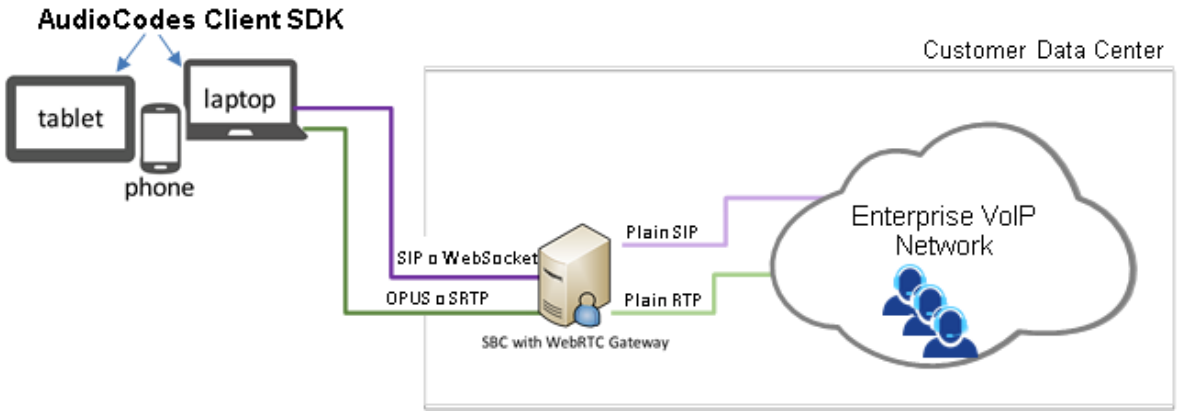
Documentation Feedback

To report an error or suggest an improvement, use the Documentation Feedback form at https://online.audiocodes.com/documentation-feedback.

1 Introduction

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

A typical WebRTC solution is comprised of a WebRTC Gateway, which is an integrated functionality on AudioCodes SBCs, and a client application running on a browser or a mobile app. AudioCodes WebRTC Android client SDK is a Java code-based API that allows Android developers to integrate WebRTC functionality into Android applications for placing calls from the Android device to the SBC.



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

1.1 Purpose

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

1.2 Scope

The guide describes the API that must be implemented to use AudioCodes' WebRTC Android SDK to build an Android application that will interact with AudioCodes' server to establish voice and video calls.

The guide may be used by Android developers who want to use the AudioCodes-provided SDK to build Web RTC clients.

1.3 Benefits

Here's a summary of the benefits:

- Simple deployment - a single WebRTC gateway device for both signaling and media
- Strong security and interoperability capabilities resulting from integration with SBC
- Client SDK for Android application.
- OPUS powered IP phones for superb, transcoder-less voice quality
- Optional recording of WebRTC calls

2 Android SDK

The sections below describe how to install the Android SDK.

2.1 Before you Start



This document contains references to the Demo Client (which includes code examples).

Before you start using the Android SDK, ensure that you have the following:

- Android Studio (recommended versions: Panda1 through Rabbit2).
- Android device or emulator with minimum OS 8.0 (API LEVEL: 26) or later.
- The following components are provided with the SDK:
  - **WebRTC SDK AAR File:** The Android Archive (AAR) file that must be included in your project to use the SDK.
  - **Android Demo Client Project:** An Android Studio project that can be opened in Android Studio. This is a fully functional client application that demonstrates how to use the SDK and the AAR file.
  - **Javadocs:** Located in the Android Demo Client project. These Dokka-generated documents can be used as a reference for the WebRTC SDK APIs.

2.2 Android SDK and Demo Client Supported Versions

The Android SDK and Demo Client support the following Android SDK requirements:

- Android SDK
  - Minimum SDK version (*minSdkVersion*): API 26 (Android 8.0)
  - Compile SDK version (*compileSdkVersion*): API 36 (Android 16)
- Demo Client
  - Minimum SDK version (*minSdkVersion*): API 26 (Android 8.0)
  - Compile SDK version (*compileSdkVersion*): API 36 (Android 16)
  - Target SDK version (*targetSdkVersion*): API 36 (Android 16)

2.3 Supported Architectures

The Android SDK supports the following architectures:

- arm64-v8a
- armeabi-v7a
- x86_64

2.4 Installation

Here's what you need to do to install the SDK.

To install the SDK:

1. Install Android Studio.
2. Open a project for the demo client.
3. Follow the instructions from the Android Studio (necessary Android SDK files will need to be downloaded and installed.)
4. The WebRTC SDK is currently required to add the following lines to the application build.gradle files (see Demo client):

```
ndk {
    abiFilters 'armeabi-v7a', 'arm64-v8a'
}
```

   You can add support for emulators with architecture x86_64:

```
ndk {
    abiFilters 'armeabi-v7a', 'arm64-v8a', 'x86-64'
}
```

5. The SDK needs specific Android permissions. The SDK does not check if these permissions are present for the Android application. Any Android application that uses the SDK needs to make sure the permission has been requested and provided by the user. The demo client provides examples on how to do this.
6. Permission description:

Required permissions:

- Internet permission:
  <uses-permission android:name="android.permission.INTERNET" />
- Microphone permission:
  <uses-permission android:name="android.permission.RECORD_AUDIO" />
- Permission to route audio to speaker/headset/Bluetooth
  <uses permission android:name="android.permission.MODIFY_AUDIO_SETTINGS" />

Recommended permissions:

- Handling of incoming GSM calls (prevent simultaneous voip and gsm calls)
  <uses-permission android:name="android.permission.READ_PHONE_STATE" />
- Is used to access WifiManager and receive events of connection/disconnection, which allows calls to be transferred from Wifi to Cellular (and vice versa)
  <uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
- Same as 2, but allows access to ConnectivityManager to get events for network changes
  <uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
- To keep the phone awake during calls (the call will be dropped if the device goes to sleep)
  <uses-permission android:name="android.permission.WAKE_LOCK" />
- For incoming call vibrate
  <uses-permission android:name="android.permission.VIBRATE" />
- To be used if custom certificates (PEM files) are to be used
  <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />
- Used for audio routing from speaker/headset/bluetooth
  <uses-permission android:name="android.permission.BLUETOOTH" />
  <uses-permission android:name="android.permission.BLUETOOTH_CONNECT" />
- Not required but older devices might still require this:
  <uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

3 API Classes

The API consists of:

- Main Classes
  - AudioCodesUA – AudioCodes User Agent (*Singleton*) – see Section 3.1
  - AudioCodesSession – For call representation (*Interface*) – see Section 3.2
  - WebRTCAudioManager – Class for managing audio routes – see Section 3.3
    TerminationInfo – Cass for information about Call Termination – see Section 3.8
- Helper Classes
  - ACConfiguration – (Optional) class for general configuration options – see Section 3.4
  - VideoConfiguration – (Optional) class for video configuration – see Section 3.5
  - DTMFOptions – Class for settings DTMF options – see Section 3.6
  - RemoteContact – Class representing the remote contact – see Section 3.7
- Listener Interfaces
  - AudioCodesEventListener – Event listener for incoming calls and login state changes – see Section 4.1
  - AudioCodesSessionEventListener – Event listener for call related events – see Section 4.2
  - WebRTCAudioRouteListener – Event listener for audio route events – see Section 4.3

3.1 AudioCodesUA

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

```
Class AudioCodesUA{
    AudioCodesUA getInstance();
    void setServerConfig(String proxyAddress, int port, String
    domain, Transport transport, List<PeerConnection.IceServer>
    iceServerList, PeerConnection.KeyType keyType);
    void setAccount (String userName, String displayName, String
    password);
    void setAccount (String userName, String displayName, String
    password, String authName);
    void setDynamicPushNotifications(Boolean
    dynamicPushNotifications);
    void setListeners (AudioCodesEventListener listener);
    void login(Context context);
    void login(Context, Boolean autoLogin);
    void logout();
    AudioCodesSession call(RemoteContact call_to, Boolean
    withVideo, Hashtable<String, String> inviteHeaders);
    void setRegisterExtraHeaders(Hashtable<String, String>
    headers);
    void setInviteExtraHeaders(Hashtable<String, String> headers);
    String getUserAgent();
    void setUserAgent(String userAgent);
    int getRegExpires();
    void setRegExpires(int registerExpires);
    void setUseSessionTimer(Boolean use);
    void setLogLevel(LogLevel level);
    void setLogger(ILog log);
    void handleNetworkChange();
    ArrayList<AudioCodesSession> getSessionList();
    void setContactRewrite(Boolean enable);
    void setOauthToken(String accessToken);
    void setPushToken(Context context, String registrationToken, String
    projectID)
    void updatePushToken(Context context, String registrationToken)
    long sendInstantMessage(RemoteContact contact, String
    message);
    void disconnectOnBrokenConnection(Boolean disconnect);
    }
    void setVerifyServer(Boolean verifyServer);
    void setVerifyServer(Boolean verifyServer, Boolean
    useAndroidKeystore, X509Certificate[] customCertificateList);

    void setAllowHeader(String[] events);
```

3.1.1 Standard Methods

3.1.1.1 getInstance

Returns the singleton object instance of class *AudioCodesUA*.

3.1.1.2 setServerConfig

Configures the server.

Parameters

- proxyAddress [*String*, address of server]
- port [*int*, port of the proxy server address
- serverDomain [*String*, domain name to register to]
- transport [*Transport*, transport for connection to the server – UDP/TCP/TLS]
- iceServerList [*List<PeerConnection.IceServer>*, list of STUN and TURN servers – null value may be supplied]
- keyType [*PeerConnection.KeyType*], configurable keyType, default: ECDSA

Return Values

N/A

3.1.1.3 setAccount (1)

Defines the account details. For this method, the authorization name is the same as the username.

Parameters

- userName [*String*, authenticating user name]
- displayName [*String*, display name for the userpassword [*String*, authenticating user password]

Return Values

N/A

3.1.1.4 setAccount (2)

Defines the account details. This is the same as setAccount (1) but with the option of having a different authorization name.

Parameters

- userName [*String*, user name]
- displayName [*String*, display name for the user]
- password [*String*, authenticating user password]
- authName [*String*, authorization user name]

Return Values

N/A

3.1.1.5 setListeners

Defines the listener's object.

Parameters

listener [*AudioCodesEventListener,* instance implementation of the AudioCodesEventListener interface that holds the methods to be triggered; see Section 4.1 for details on how it is defined; see also Section 5.2 for an example] API Callbacks/ Listeners Interfaces User Agent: Set

Return Values

N/A

3.1.1.6 login

Performs registration to the service.

Parameters

context [*Context*, Android application context]

Return Values

N/A

3.1.1.7 login (click to call)

Performs initialization to the service. The SDK will not register to the service but will allow outgoing calls to be made.

Parameters

- **context** [*Context*, Android application context]
- **AutoLogin** (Boolean):
  - **true:** The SDK registers to the service (same as login 3.1.1.6)
  - **false:** The SDK does not register to the service, but calls can be made (click to call feature)

Return Values

N/A

3.1.1.8 logout

Performs de-registration from the service.

Parameters

N/A

Return Values

N/A

3.1.1.9 call

Initiates an outgoing call.

Parameters

- call_to [RemoteContact, destination address/number]
- withVideo [*Boolean*, 'true' if the call is initiated with video]
- inviteHeaders [*Hashtable<String, String>,* list of headers with a key/value where each key is added as a header to the SIP INVITE with the specified value]

Return Values

- session [*AudioCodesSession*, a call session object defined here.]

3.1.1.10 setVerifyServer

Enables/disables the certificate verification when establishing a TLS connection; the default state is disabled. This method is the same as calling setVerifyServer(true,true,null). For example, see Section 3.1.1.11. By default, Android CA Keystore is used.

Parameters

verifyServer [*Boolean*, 'true' to enable verification]

Return Values

N/A

3.1.1.11 setVerifyServer

Method for setting server certificate verification. The SDK allows for custom certificates to be added to the verification process. This method allows for a custom certificate list to be provided, where the customer can choose how to store the certificates. E.g. the certificate can be stored encrypted in the app and be provided decrypted as an object list to the SDK.

If verifyServer is set to false – no server certificate will be verified

If Android CA Keystore is set to true – the Android CA Keystore certificates will be added to the custom keystore that will be used to verify the server certificate

If a X509Certificate list is provided – this list will be added to the custom keystore that will be used to verify the server certificate

If a path for certificates is provided through setCaCertFilePath - – this list will be added to the custom keystore that will be used to verify the server certificate

The final custom keystore used for verification will contain:

List of Android CA Keystore certificates (if useAndroidKeystore is true) + X509Certificate list (if provided) + caCertFilPath certificates (if provided).

If none is provided and verifyServer is set to true, the verification process will fail.

The custom keystore will be used for verification by Android Trustmanagers.

Parameters

verifyServer [*Boolean*, 'true' to enable verification]

useAndroidKeyStore [Boolean, ‘true’ to enable verification with Android Keystore certificates

X509Certificate[] [X509Certificate list, a list with custom added certificates to be used for certificate verification. This list may be null, contain a single certificate, or multiple certificate (to build the certificate chain).

Return Values

N/A

3.1.2 Advanced Methods

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

3.1.2.1 setRegisterExtraHeaders

Allows adding additional headers to the registration request.



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

Parameters

headers [*Hashtable<String, String>,* list of headers with a key/value where each key is added as a header to the registration request with the specified value]

Return Values

N/A

3.1.2.2 setInviteExtraHeaders

Allows adding additional headers to the INVITE request or response.



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

Parameters

headers [*Hashtable<String, String>,* list of headers with a key/value where each key is added as a header to the SIP INVITE with the specified value]

Return Values

N/A

3.1.2.3 getUserAgent

Gets the current user-agent string, used to build the SIP header User-Agent.

Parameters

N/A

Return Values

User agent [String, text describing the SIP user agent]

3.1.2.4 setUserAgent

Sets the user-agent string, used to build the SIP header User-Agent.

Parameters

User agent [*String,* text describing the SIP User Agent]

Return Values

N/A

3.1.2.5 getRegExpires

Gets the current default registration interval.

Parameters

N/A

Return Values

expires [*integer*, seconds]

3.1.2.6 setRegExpires

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

Parameters

expires [*integer*, seconds]

Return Values

N/A

3.1.2.7 setUseSessionTimer

Allows enabling session timers in the call session.

Parameters

- **UseSessionTimer** (Boolean):
  - **true:** Session timers are optionally supported. [e.g., the SBC initiates session timers if configured to do so (default value)
  - **false:** The session timers are not enabled

Return Values

N/A

3.1.2.8 setLogLevel

Changes the log level used by the application. Release builds might want to set the log level higher for security reasons. WebRTC internal logs are only enabled if the debug level is lower than or equal to DEBUG level.

Parameters

logLevel [*LogLevel*, log level enum]

Return Values

N/A

3.1.2.9 setLogger

Changes the logger used by the SDK. Logs for the WebRTC SDK and logs for PjSIP are written to the custom logger. WebRTC internal logs are still be written to the console (if the log level is lower than debug)

Parameters

logger [*ILog*, instance object of the ILog interface]

Return Values

N/A

3.1.2.10 handleNetworkChange

Handles network changes when called. This re-registers the client and reestablishes the audio sessions when the network has been changed.

It must be explicitly called by the client application. The SDK does not automatically detect a network change. Ideally, this must be called when the network is reconnected, not when disconnected.

Parameters

- N/A

Return Values

N/A

3.1.2.11 getSessionList

Gets the current session list.

Parameters

- N/A

Return Values

3.1.2.12 List of Sessions [ArrayList<AudiocodesSession>, List of Active Sessions]disconnectOnBrokenConnection

This method changes the way call handover is being handled by the SDK. Default behavior is to disconnect a call when there is no RTP for a period of time. This method (when set to 'false') allows for the call to continue even when the connection is broken.

Parameters

- **DisconnectOnBrokenConnection**(Boolean):
  - **true:** Calls are terminated when a network connection error occurs in the audio stream (default value)
  - **false:** Calls continue when a network connection error occurs

Note: Setting this to 'false' forces enable contact rewrite.

Return Values

N/A

3.1.2.13 setContactRewrite

This option is used to update the transport address and the Contact header of REGISTER request. When this option is enabled, the SDK keeps track of the public IP address from the response of the REGISTER request. Once it detects that the transport address has changed, it will unregister the current Contact, update the Contact with the transport address learned from the Via header, and register a new Contact to the registrar. It will also update the public name of the UDP transport if STUN (Session Traversal Utilities for NAT) is configured.

Parameters

- **ContactRewrite**(Boolean):
  - **true**: The library tracks the public IP address from the response of the REGISTER request
  - **false:** The library does not track the public IP address from the response of the REGISTER request. (Default value)

Return Values

N/A

3.1.2.14 setOauthToken

Optional method to allow the SDK to use Oauth authentication for registration to the service. The SDK will add an authorization header with the supplied access token. (The SBC will need to be configured to use Oauth authorization as well.)

Parameters

accessToken [*String*]– access token as received from the Oauth server (refer to the Democlient for an example on Oauth registration)

Return Values

N/A

3.1.2.15 setPushToken

Optional method to allow the SDK to use push for incoming calls. If set than the SDK will send the push credentials to the SBC which will allow the SBC to send push messages for incoming calls. Refer to the Democlient for an example.

This method sets the push parameters for the PNS according to:
Push Notification with the Session Initiation Protocol (SIP) see: https://www.ietf.org/id/draft- *ietf-sipcore-sip-push-20.txt*

The values are stored in permanent memory and will be used by the WebRTC SDK until the values are reset to null values. It is recommend to call upon this method before calling upon login as calling upon this method after will cause a re-register of the SIP stack.

Parameters

- **context** [*Context*, Android application context]
- registrationToken [*String*, Google push registration token]
- projectID [*String*, Google project ID]

Return Values

N/A

3.1.2.16 updatePushToken

This method should be called if the client's registration token has been changed. Refer to the Democlient for an example.

This method is the same as setPushToken except the projectID that is emitted. This method uses the projectID as set in setPushToken. Therefore, setPushToken must have been called once before.

Parameters

- context [*Context*, Android application context]
- registrationToken [*String*, Google push registration token]

Return Values

N/A

3.1.2.17 sendInstantMessage

This method sends a message to the remote contact. This is done according to RFC 3428 - Session Initiation Protocol (SIP) Extension for Instant Messaging. The status of the message will be returned in AudioCodesEventListener (see onInstantMessageStatus).

Parameters

- contact [RemoteContact, destination address/number]
- message [*String*, message that will be sent to the remote contact]

Return Values

- message ID [long, ID for the message, this message ID will be used in the callback for the message status]

3.1.2.18 setAllowHeader

Sets/adds a new allow header to the SDP in case of 180 ringing message

Parameters

User agent [*String[],* comma separated value for the header]

Return Values

N/A

3.1.2.19 setDynamicPushNotifications

This method introduces new way of handling push notifications – dynamic way. If parameter dynamicPushNotifications is set to true, stop using method setPushToken and updatePushToken. Instead, you need to have your own push notification gateway that is registered in SBC that supports this dynamic mode.

Parameters

dynamicPushNotifications [Boolean]

Return Values

N/A

3.2 AudioCodesSession

Represents a call session. Used in two scenarios:

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

Syntax

```
class AudioCodesSession {
    int getSessionID();
    void answer(Hashtable<String, String> inviteHeaders, Boolean
    withVideo);
    void reject(Hashtable<String, String> inviteHeaders);
    void terminate();
    void muteAudio(Boolean mute);
    void muteVideo(Boolean mute);
    boolean isAudioMuted();
    boolean isVideoMuted();
    void sendDTMF(DTMF dtmf);
    void sendInfo(String info);
    Boolean isOutgoing();
    Boolean hasVideo();
    RemoteContact getRemoteNumber();
    CallState getCallState();
    int duration();
    long getCallStartTime();
    CallTermination getTermination();
    Boolean isLocalHold();
    Boolean isRemoteHold();
    Object getUserData();
    void setUSerData(Object object);
    void hold(Boolean hold);
    void switchCamera();
    void showVideo(Activity);
    void showVideo(SurfaceViewRenderer localView,
    SurfaceViewRenderer remoteView);
    void stopVideo();
    void setLocalRenderPosition(int xPercentage, int yPercentage);
    void addSessionEventListener(AudioCodesSessionEventListener
    sessionEventListener);
    void removeSessionEventListener(AudioCodesSessionEventListener
    sessionEventListener);
    ACCallStatistics getStats();
    void redirect(RemoteContact contact, Hashtable<String,String>
    inviteHeaders);
    void reinviteWithVideo();
    Boolean transferCall(RemoteContact transferTo);
    Boolean transferCall(AudioCodesSession transferToSession);
    RemoteContact getTransferContact();
    CallTransferState getTransferState();
    }
```

3.2.1 Standard Methods

3.2.1.1 getSessionID

Retrieves the internal identifier for the session. This identifier can be used in case there is more than one session.

Parameters

N/A

Return Values

sessionID [*int,* ID of the session]

3.2.1.2 answer

Initiates the object and establishes the call. Only valid for incoming calls.

Parameters

- headers [*Hashtable<String, String>,* list of headers with a key/value where each key is added as a header to the SIP response with the specified value]
- **AnswerWithVideo** (Boolean):
  - **true:** The call is answered with video and video is unmuted. Both sides see each other
  - **false:** The call is answered with video, but video muted. This side will see the remote video, but the remote side cannot see the local side video

Return Values

N/A

3.2.1.3 reject

Rejects a call. Only valid for incoming calls.

Parameters

headers [*Hashtable<String, String>,* list of headers with a key/value where each key iadded as a header to the SIP response with the specified value]

Return Values

N/A

3.2.1.4 Terminate

Terminates an active call. Only valid for outgoing and established calls.

Parameters

N/A

Return Values

N/A

3.2.1.5 muteAudio

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

Parameters

- **MuteAudio** (Boolean):
  - **true:** To mute audio
  - **false:** To unmute audio

Return Values

N/A

3.2.1.6 muteVideo

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

Parameters

- **MuteVideo** (Boolean):
  - **true:** To mute video
  - **false:** To unmute video

Return Values

N/A

3.2.1.7 isAudioMuted

Checks audio mute status.

Parameters

N/A

Return Values

- **IsAudioMuted** (Boolean):
  - **true:** If muted
  - **false**: If not

3.2.1.8 isVideoMuted

Checks video mute status.

Parameters

N/A

Return Values

- **IsVideoMuted**(Boolean):
  - **true:** If muted
  - **false**: If not

3.2.1.9 sendDTMF

Sends a DTMF character

Parameters

- dtmf [*DTMF*, an enum containing a DTMF character]

Return Values

N/A

3.2.1.10 sendInfo

Sends a SIP info message

Parameters

- info [*String*, message in JSON format]

Return Values

N/A

3.2.1.11 isOutgoing

Checks if a call is outgoing.

Parameters

N/A

Return Values

- **IsCallOutgoing** (Boolean):
  - **true:** If outgoing
  - **false**: If incoming

3.2.1.12 hasVideo

Checks if a call has video.

Parameters

N/A

Return Values

- **IsCallWithVideo** (Boolean):
  - **true:** If the call has video
  - **false:** If the call is audio only

3.2.1.13 getCallState

Gets the call state of the session.

Parameters

N/A

Return Values

- callstate [*CallState*, enum containing the call state]

3.2.1.14 getTermination

Gets the termination reason of the session.

Parameters

N/A

Return Values

- termination [*CallTermination,* enum containing the termination reason]

3.2.1.15 duration

Defines the call duration, in seconds. It will be -1 if the call has still not been established.

Parameters

N/A

Return Values

- duration [*int,* call duration in seconds, -1 if the call is not yet established]

3.2.1.16 isLocalHold

Checks if this side of the call has put the call on hold.

Parameters

N/A

Return Values

- **isCallLocallyOnHold** (Boolean):
  - **true:** If call is on local hold
  - **false:** if otherwise

3.2.1.17 isRemoteHold

Checks if the remote side of the call has put the call on hold.

Parameters

N/A

Return Values

- **IsCallRemotelyOnHold** (Boolean):
  - **true:** If the call is on local hold
  - **false:** If otherwise

3.2.1.18 setUserData

Allows the optional setting of a user-created object / data to be attached to a specific session. The reference is removed from the session after the session has been terminated.

Parameters

userData [*Object,* Java object for option user data]

Return Values

N/A

3.2.1.19 getUserData

Allows the optional retrieval of a user-created object / data which was attached to a certain session.

Parameters

N/A

Return Values

- userData [*Object,* Java object for option user data]

3.2.1.20 hold

Set call on hold (or un-hold). The callProgress callback in.AudioCodesSessionEventListener indicates when the call has been placed on hold/unhold.

Parameters

Hold [*Boolean,* set call to hold]

Return Values

N/A

3.2.1.21 switchCamera

Switches the camera between front and back camera. This method requires the device to have two cameras. A successful camera switch is returned in the cameraSwitched callback in AudioCodesSessionEventListener.

Parameters

N/A

Return Values

N/A

3.2.1.22 showVideo (1)

Displays video during a call. This requires the UI element *ac_webrtc_video* to be included in the layout of the activity. This is the recommended method to be used when using video.
For a more advanced and customized option, see *showVideo(2)* in the section below.

Parameters

Activity [*Activity,* Android activity that is used to display the video. This activity must contain the *ac_webrtc_video* UI element in its content view.

Return Values

N/A

3.2.1.23 showVideo (2)

Displays video during a call. This does not require the UI element *ac_webrtc_video* to be included in the layout of the activity. The method is mutually exclusive to showVideo (1).

Parameters

- localView [*SurfaceViewRender,* Android SurfaceViewRenderer UI element; this element is used to show the local stream.
- remoteView [*SurfaceViewRender,* Android SurfaceViewRenderer UI element; this element is used to show the remote stream.

Return Values

N/A

3.2.1.24 stopVideo

Stops the capturing of video and removes the remote and local renderer. To start video again, showVideo needs to be called.

Parameters

N/A

Return Values

N/A

3.2.1.25 setLocalRenderPosition

Sets the local render position. Only relevant if showVideo(1) method is used.
This method should be called before showVideo.

Parameters

- xPercentage [*int*, the horizontal position of the top left side of the local render view. This is set as a percentage of the *ac_webrtc_video* screen. E.g., if the local render view has a width of 1000 pixels. Setting this item to 66 will place the top left side of the local render view at pixel 660 of the x-axis]
- yPercentage [*int*, the vertical position of the top left side of the local render view. This is set as a percentage of the *ac_webrtc_video* screen. For example, if the local render view has a height of 1500 pixels. Setting this item to 66 will place the top left side of the local render view at pixel 1000 of the y-axis]

Return Values

N/A

3.2.1.26 addSessionEventListener

Adds an event listener to listen for session events. The client application might add multiple listeners. The listeners will receive events until they are other removed or the session was terminated.

Parameters

sessionEventListener[*AudioCodesSessionEventListener*, implementation object of the AudioCodesSessionEventListener interface]

Return Values

N/A

3.2.1.27 removeSessionEventListener

Removes a sessionEventListener that was previously added in the addSessionEventListener method.

Parameters

sessionEventListener[*AudioCodesSessionEventListener*, implementation object of the AudioCodesSessionEventListener interface]

Return Values

N/A

3.2.1.28 getStats

Retrieves statistics on the entire call session.

Parameters

- N/A

Return Values

ACCallStatistics[*ACCallStatistics*, object containing the call statistics]

3.2.1.29 redirect

Redirects an incoming call to a new number. Indicates to the calling party that the caller should try to call the number specified in RemoteContact (see also under here).

Parameters

- contact [RemoteContact, redirect destination address/number]
- inviteHeaders [*Hashtable<String, String>,* list of headers with a key/value where each key is added as a header to the SIP INVITE with the specified value]

Return Values

- session [*AudioCodesSession*, a call session object defined here]

3.2.1.30 reinviteWithVideo

Enables video during a call. Sends a REINVITE with video enabled. Note that showVideo will need to be called (see code example delivered with the SDK).

Parameters

N/A

Return Values

N/A

3.2.1.31 transferCall (blind transfer)

Transfers the other side of the current AudioCodesSession to the remote contact supplied.

This is a blind transfer and should be used when there is only one session. Refer to the Democlient for an example.

Parameters

contact [RemoteContact, transfer destination address/number. Remote Contact to which the other side of the current call will be transferred to].

Return Values

TransferCall (Boolean):

- **true:** If the current call is in connected state
- **false:** In any other case

3.2.1.32 transferCall (attended transfer)

Transfers the other side of the current AudioCodesSession to the AudioCodesSession supplied. This is a attended transfer and should be used when there is more than one session. Refer to the Democlient for an example.

Parameters

transferToSession [AudioCodesSession - Transfers the destination call AudioCodesSession to which the other side of the current call will be transferred to. For example, the other side of the current AudioCodesSession will try to replace this call with a call to the number of the supplied AudioCodesSession.]

Return Values

- **TransferCall** (Boolean):
  - **true:** If the current call (AudioCodesSession) is on hold or connected and the supplied call (AudioCodesSession) is on hold
  - **false:** In any other case

3.2.1.33 getTransferContact

Gets the contact to which the call is being transferred to. After a successful transfer the transfer contact becomes the remote contact (see getRemoteContact).

Parameters

- NA

Return Values

- Transfer contact [RemoteContact, the transfer contact. This can be the contact to which the other side of the current call is being transferred to, or the remote contact to which this call is being transferred to.]

3.2.1.34 getTransferState

Gets the state of the transfer. This method is only applicable for calls that are in the process of transfer or being transferred. See getCallState to know when an AudioCodesSession is in the state of transfer.

Parameters

- N/A

Return Values

Transfer state [CallTransferState, the state of the transfer for this AudioCodesSession. Possible states:

- **NONE:** No transfer in progress.
- **TRANSFER_REQUEST_RECEIVED_IN_PROGRESS:** The other side has sent a transfer request, getTransferContact which returns the contact to whom this call is being transferred to.
- **TRANSFER_REQUEST_RECEIVED_FAILED:** The other side has sent a transfer request, but the transfer did not succeed.
- **TRANSFER_REQUEST_RECEIVED_SUCCEEDED:** The other side has sent a transfer request, and the transfer succeeded.
- **TRANSFER_REQUEST_SEND_IN_PROGRESS:** This side has sent a transfer request which is being processed. getTransferContact returns the contact to whom the other side of this call is being transferred to.
- **TRANSFER_REQUEST_SEND_FAILED:** This side has sent a transfer request which has failed.
- **TRANSFER_REQUEST_SEND_SUCCEEDED:** This side has sent a transfer request which has succeeded.
- **TRANSFER_REPLACED:** This AudioCodesSession is in a call with a remote party and the remote party has been replaced (side C in an attended transfer). getRemoteContact returns the new number to where the call has been transferred to.

3.3 WebRTCAudioManager

This defines WebRTC SDK Audio management. This class handles audio routing during WebRTC calls.

Syntax

```
class AudioCodesSession {
    WebRTCAudioManager getInstance();
    void setWebRTCAudioRouteListener(WebRTCAudioRouteListener
    listener);
    void setAudioRoute(AudioRoute route);
    AudioRoute getAudioRoute();
    List<AudioRoute> getAvailableAudioRoutes();
    }
```

3.3.1 Standard Methods

3.3.1.1 getInstance

Gets the singleton instance of the WebRTCAudioManager class.

Parameters

N/A

Return Values

- instance [*WebRTCAudioManager*, singleton object instance]

3.3.1.2 setWebRTCAudioRouteListener

Sets a listener for listening to updates in the current audio route and available audio routes.

There must always be only one listener; setting a new listener will overwrite the previous listener.

Parameters

listener [*WebRTCAudioRouteListener*, implementation instance of the WebRTCAudioRouteListener interface]

Return Values

- N/A

3.3.1.3 setAudioRoute

Sets the audio route. This method changes the audio route of the device. This generally should be used during a call. The audio will be only be routed if the new audio route is available.

For Tablet devices, it is recommended to change to speaker route if available.

Parameters

audioRoute [*AudioRoute*, enum describing the new audio route]

Return Values

- **NewAudioRouteFound** (Boolean):
  - **true:** If the new audio route was found
  - **false**: If the new audio route is unavailable

3.3.1.4 getAudioRoute

Gets the current audio route.

Parameters

N/A

Return Values

- audioRoute [*AudioRoute*, enum describing the new audio route]

3.3.1.5 getAvailableAudioRoutes

Gets the available audio routes.

For Tablet devices, it is recommended to check if a speaker route is available.

Parameters

N/A

Return Values

- audioRouteList [*List<AudioRoute>*, list of enums with the available audio routes]

3.4 ACConfiguration

Used to provide additional configuration options for the WebRTC SDK. Using this class is optional. The class is a singleton object. The configuration object can be retrieved through getConfiguration. Any changes in that object will be applied to the SDK. It is recommended to apply any changes before calling the AudioCodesUA login method.

```
Class ACConfiguration{
    ACConfiguration getConfiguration();
    String version();
    int getLocalServerPort();
    void setLocalServerPort(int port);
    void setLocalServerPort(LocalServerPortOptions
    localServerPortOptions);
    DTMFOptions getDTMFOptions();
    void setDTMFOptions(DTMFOptions dtmfOptions);
    VideoConfiguration getVideoConfiguration();
    void setVideoConfiguration(VideoConfiguration configuration);
    void setAutomaticCallOnRedirect(Boolean automaticRedirect);
    Boolean getAutomaticCallOnRedirect();
    void setRedirect(Boolean redirect, RemoteContact
    redirectContact);
    RemoteContact getRedirectContact();
    boolean getRedirectEnabled();
    }
```

3.4.1 Standard Methods

3.4.1.1 getConfiguration

Defines the static method that returns the current used configuration object.

Parameters

N/A

Return Values

- configuration [*ACConfiguration*, current used configuration object; see Section 3.4]

3.4.1.2 version

Defines the static method that returns the current version of the SDK.

Parameters

N/A

Return Values

- Version [*String*, version of the SDK, e.g., 1.x]

3.4.1.3 getLocalServerPort

Gets the current default local port used by the SIP stack.

Parameters

N/A

Return Values

- port [*integer*, default local user port (default randomized port from range 5000 - 7000)]

3.4.1.4 setLocalServerPort

The local SIP server port is randomly selected from the default range (5000 – 7000).

Changes the default local SIP server port.

Parameters

- port [*integer,* default local server port]

Return Values

N/A

Changes the range from which is the local SIP port selected randomly.

Parameters

- localServerPortOptions [LocalServerPortOptions, default local server port range)

Return Values

N/A

3.4.1.5 getDtmfOptions

Gets the current default local port used by the SIP stack

Parameters

N/A

Return Values

dtmfOptions [*DTMFOptions*, DTMFOptions class for setting the handling of DTMF tones; the default value is for the WebRTC to handle DTMF tones]

3.4.1.6 setDtmfOptions

Changes the DTMFOptions class used by the SDK. This allows for sending DTMF through either the WebRTC or SIP INFO. The class allows for changing DTMF duration and interval (if applicable for the chosen method). See Section 3.6 for more information.

Parameters

- dtmfOptions [*DTMFOptions*, DTMFOptions class for setting the handling of DTMF tones]

Return Values

N/A

3.4.1.7 getVideoConfiguration

Gets the current video configuration used by the SDK.

Parameters

N/A

Return Values

- configuration [*VideoConfiguration*, class containing video configuration options]

3.4.1.8 setVideoConfiguration

Changes the video configurations options used by the SDK. See also Section 3.5.

Parameters

- configuration [*VideoConfiguration*, class containing video configuration options]

Return Values

N/A

3.4.1.9 setAutomaticCallOnRedirect

Sets automatic call redirection for outgoing call redirection. If enabled, the SDK will automatically attempt to call the number supplied in the 3XX response on the INVITE.

If not, the call will be terminated, and the redirect number will be provided in the callProgress (redirect) callback. See the code example delivered with the SDK. In this case, the application using the SDK will be responsible for placing a new call to the new number, if necessary.

Parameters

- **CallRedirected** (Boolean):
  - **true:** automatic redirect is enabled
  - **false:** automatic redirect is disabled

Return Values

N/A

3.4.1.10 getAutomaticCallOnRedirect

Gets the current setting for automatic call redirection.

Parameters

N/A

Return Values

IsCallRedirected (Boolean):

- **true:** Automatic redirect is enabled
- **false:** Redirect is disabled

3.4.1.11 setRedirect

This method allows for the setting of incoming call redirection. If enabled, *all* incoming calls will be answered with a 302 response and the redirect contact provided in the redirectContact field. If the redirect field is enabled, there will be no incoming call callbacks

Parameters

- redirect [*Boolean*, if true, *all* incoming call redirection is enabled; if not, then redirect is disabled]
- redirectContact [*RemoteContact*, the contact to whom the call should be redirected. Only username is mandatory. The domain and scheme fields will be taken from the user account if not provided.

Return Values

N/A

3.4.1.12 getRedirectContact

Gets the current contact that was set for incoming call redirection

Parameters

N/A

Return Values

- Redirect contact [*RemoteContact*, current contact set for incoming call redirection]

3.4.1.13 getRedirectEnabled

Gets the current setting for incoming call redirection.

Parameters

N/A

Return Values

IncomingCallRedirected (Boolean):

- **true:** Incoming call redirect is enabled
- **false:** Incoming call redirect is disabled

3.5 VideoConfiguration

Used to provide additional configuration options for the WebRTC SDK. Using this class is optional. The class provides access to public parameters that can be changed if needed.

The configuration object can be retrieved through getVideoConfiguration in ACConfiguration class. Calling setVideoConfiguration in the ACConfiguration class will apply the changes. It is recommend to call setVideoConfiguration before showVideo is called.

```
Class VideoConfiguration{
    }
```

3.5.1 Camera Parameters

- **cameraWidth** – captures the width of the camera (default 640)
- **cameraHeight** - captures the height of the camera (default 480)
- **cameraFrameRate** - captures the framerate of the camera (default 15)

3.5.2 Rendering Views Parameters

- **LOCAL_X_CONNECTING** – Uppermost left corner (% of the screen width) of the local video screen when a remote video stream is not yet available.
- **LOCAL_Y_CONNECTING** - Uppermost left corner (% of the screen height) of the local video screen when a remote video stream is not yet available.
- **LOCAL_WIDTH_CONNECTING** – Lowermost right corner (% of the screen width) of the local video screen when a remote video stream is not yet available.
- **LOCAL_HEIGHT_CONNECTING** – Lowermost right corner (% of the screen height) of the local video screen when a remote video stream is not yet available.
- **LOCAL_X_CONNECTED** - Uppermost left corner (% of the screen width) of the local video screen when a remote video stream is available.
- **LOCAL_Y_CONNECTED** - Uppermost left corner (% of the screen height) of the local video screen when a remote video stream is available.
- **LOCAL_WIDTH_CONNECTED** – Lowermost right corner (% of the screen width) of the local video screen when a remote video stream is available.
- **LOCAL_HEIGHT_CONNECTED** – Lowermost right corner (% of the screen height) of the local video screen when a remote video stream is available.
- **REMOTE_X** - Uppermost left corner (% of the screen width) of the remote video screen when a remote video stream is available.
- **REMOTE_Y** - Uppermost left corner (% of the screen height) of the remote video screen when a remote video stream is available.
- **REMOTE_WIDTH** - Lowermost right corner (% of the screen width) of the remote video screen when a remote video stream is available
- **REMOTE_HEIGHT** - Lowermost right corner (% of the screen height) of the remote video screen when a remote video stream is available

3.6 DTMFOptions

Used to provide additional configuration options for the WebRTC SDK. Using this class is optional. The class provides access to public parameters that can be changed if needed. The class allows configuration of sending DTMF events.

```
Class DTMFOptions{
    }
```

3.6.1 DTMF Parameters

- **dtmfMethod** - DTMFMethod enum parameter that supports sending of DTMF through:
  - **WEBRTC** - DTMF is sent through media by telephone-event using the WebRTC engine. This is the default method.
  - **SIP_INFO** - DTMF events are sent through SIP_INFO events.
- **duration -** duration of the DTMF event, in milliseconds. When using SIP_INFO, the minimum is 100, which is the default value.
- **intervalGap** - the interval gap in milliseconds between sending DTMF events. This is only relevant for WEBRTC DTMF events. Default: 70.

3.7 RemoteContact

Used to represent a remote contact. This can be either a dialed number or a remote contact received through an incoming call.

```
Class RemoteContact{
    String getDisplayName();
    String getUserName();
    String getDomain();
    void setDisplayName(String displayName);
    void setUserName(String username);
    void setDomain(String domain);
}
```

3.7.1 Standard Methods

3.7.1.1 getDisplayName

Gets the contact display name. Note that this might not be available since the remote contact did not set a display name.

Parameters

N/A

Return Values

- displayName [*String*, display name of the remote contact]

3.7.1.2 getUserName

Gets the contact user name.

Parameters

N/A

Return Values

userName [*String*, user name of the remote contact]

3.7.1.3 getDomain

Gets the contact domain.

Parameters

N/A

Return Values

- domain [*String*, domain of the remote contact]

3.7.1.4 setDisplayName

(Optional) Sets the contact display name. Since this does not affect SIP signaling, it's optional; it allows for easier retrieval of the display name used in the call.

Parameters

- displayName [*String*, display name of the remote contact]

Return Values

N/A

3.7.1.5 setUsername

Sets the contact user name.

Parameters

- userName [*String*, user name of the remote contact]

Return Values

N/A

3.7.1.6 setDomain

(Optional) Sets the contact domain.

Parameters

domain [*String*, domain of the remote contact. This value doesn't usually have to be set as the remote contact is likely to reside in the same domain]

Return Values

N/A

3.8 TerminationInfo

This section describes Call Termination information.

```
Class TerminationInfo{
    CallTermination;
    int statusCode;
    String reason;
    String reasonHeader;
    String sipMessage;
```

3.8.1 TerminationInfo attribute

- **CallTermination** – enum indicating the reason for the termination. This is a simplified version of the status code and should be enough for most call cases
- **statusCode –** SIP status code of the termination (e.g. 404 or 500, etc.). This can be useful in specific use cases
- **reason –** SIP reason, e.g. Server Internal Error or User Not Found
- **reasonHeader –** SIP reason header – describes a specific reason for the call termination, e.g. in case of 500 Server Internal Error, the reason header could be: Reason: SIP; cause=500; tet=”General Routing Failure”
- **sipMessage –** the last sip message, this will be mostly be used for debugging or for future uses

3.9 InfoAlert

This section describes Call Termination information.

```
Class InfoAlert{
    boolean autoAnswer;
    int delay;
```

3.9.1 InfoAlert attribute

- **autoAnswer** – Boolean indicating whether the call should be answered automatically after the *delay* property value
- **delay** – The amount of time, in seconds, that needs to pass before the call is answered automatically.

4 API Callbacks/ Listeners Interfaces

The API provides capability to register in order to listen to different types of events. Here's a list of the interfaces that must be implemented to receive an event:

4.1 AudioCodesEventListener

Interface for receiving SDK events. This interface must be implemented and set through the AudioCodesUA class to receive these event.

4.1.1 Login state changed event

Triggered when the login state has been changed.

Syntax

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

Parameters

- IsLogin [*Boolean,* 'true' if logged in and 'false' if not logged in]
- cause [*String*, text describing the received SIP reason. This can be mostly used if more information on a login failure is required]

4.1.2 Incoming call event

Triggered when receiving an incoming call.

Syntax

```
void incomingCall(AudioCodesSession call, InfoAlert infoAlert);
```

Parameters

- Session [*AudioCodesSession*, the incoming call session object]
- infoAlert [InfoAlert, the incoming call auto answer object]

4.1.3 Incoming IM message event

Triggered when receiving an incoming message.

Syntax

```
void onIncomingInstantMessage(RemoteContact contact, String
message);
```

Parameters

- contact [*RemoteContact*, remote contact that send the message]
- message [String, the message that the remote contact send]

4.1.4 IM Message status event

When a message has been sent, the status of the message can be tracked using the callback onInstantMessageStatus.

Syntax

```
void onInstantMessageStatus(InstanceMessageStatus status, long
id);
```

Parameters

- status [*InstanceMessageStatus*, status of the message. This is an enum with the following values:
  - **SUCCESS** – the message has been delivered to the remote contact
  - **ACCEPTED** – the message has been delivered to the server (remote contact might not have received the message yet though)
  - **NOT_FOUND** – the remote contact does not exist or the server is not able to deliver the message to the remote contact
  - **UNKNOWN_ERROR** – an unknown error occurred, the message was not delivered
- Message ID [long, the ID of the message, this corresponds to the ID used in sendInstantMessage]

4.2 AudioCodesSessionEventListener

4.2.1 callTerminated

Callback for when the session is terminated by the local or the remote side.

Syntax

```
void callTerminated(AudioCodesSession session, Terminationinfo
info);
```

Parameters

- session [*AudioCodesSession* refers to the call session object that was terminated. The object will be removed at the end of the callTerminated method]
- info [TerminationInfo – Refers to the object containing information about the termination reason. e.g., termination reason, status code,. See TerminationInfo.

4.2.2 callProgress

Callback for changes in the state of the call. The call progress state can be retrieved by getCallState in the AudioCodesSession object.

Syntax

```
void callProgress(AudioCodesSession call);
```

Parameters

- session [*AudioCodesSession*, the call session object]

4.2.3 cameraSwitched

Callback for when the camera has been switched between the front or the back camera.

Syntax

```
void cameraSwitched(Boolean frontCamera);
```

Parameters

FrontCamera (Boolean):

- **true:** The camera has switched to the front camera
- **false:** The camera has switched to the back camera

Return Values

- N/A

4.2.4 reinviteWithvideoCallback

Callback for when a video is added during a call. This callback can be used to call showVideo (from the UI thread). See the code example delivered with the SDK.

Syntax

```
void reinviteWithVideoCallback(AudiocodesSession session);
```

Parameters

- session [*AudioCodesSession*, the call session object]

4.2.5 mediaFailed

Callback for a failure in the media. This callback is for information purposes only and can be used to inform the user that there currently is no media. This can be most useful in the case of a network change, where a media fail might be expected.

Syntax

```
void mediaFailed(AudioCodesSession call);
```

Parameters

- session [*AudioCodesSession*, the call session object]

4.2.6 incoming Notify

Callback for when an incoming notify state is received, video is added during the call. This callback can be used to execute the incoming notify state (from the UI thread if needed). See the code example delivered with the SDK.

Syntax

```
void incomingNotify(NotifyEvent notifyEvent, String dtmfValue);
```

Parameters

- notifyEvent [NotifyEvent, the call notify event type, This is an enum with the following values:
  - **TALK**– If the call is incoming and is not answered yet, then the client application is required to answer the call. If the call is already active and on hold, then the client application is required to un-hold it.
  - **HOLD**– If the call is active, then the client application is required to put the call on hold.
  - **DTMF**– The client application is required to send DTMF characters provided in the *dtmf* string parameter. This should be performed using calls to the sendDtmf method consecutively for each character in the DTMF string, and in a way that is non-blocking to the current thread. The interval between sending each DTMF character should be MAX(DTMFOptions.intervalGap, DTMFOptions.duration).
  - **CONFERENCE** – Currently not supported
- **MESSAGE – Incoming notification for a SIP message**dtmfValue [String, the call dtmf value or message body, null in case notifyEvent is not DTMF or **MESSAGE**]

4.3 WebRTCAudioRouteListener

Interface for receiving audio routes events. The interface must be implemented and set through the WebRTCAudioManager class to receive these events.

4.3.1 audioRoutesChanged

Callback for when the list of available audio routes has been changed, for example, if the user is connected to a Bluetooth audio device

Syntax

```
void audioRoutesChanged(List<WebRTCAudioManager.AudioRoute>
audioRouteList);
```

Parameters

- audioRouteList *[List<WebRTCAudioManager.AudioRoute>,* a list of available audio routes]

4.3.2 currentAudioRouteChanged

Callback for when the currently used audio route has been changed. If the user adds a Bluetooth audio device, for example, the SDK routes the audio to the Bluetooth device and this callback will be called.

Syntax

```
void currentAudioRouteChanged(WebRTCAudioManager.AudioRoute
newAudioRoute);
```

Parameters

newAudioRoute [WebRTCAudioManager.AudioRoute, the new audio route where the audio is being routed to]

5 Use Examples

Here are some user examples for your reference.

5.1 User Agent: Create Instance, Set Server and Account

```
AudioCodesUA phone = new AudioCodesUA(); // phone API
ArrayList<PeerConnection.IceServer> iceServerList = new
ArrayList<PeerConnection.IceServer>();
phone.setServerConfig("webrtclab.audiocodes.com", 5080,
"example.com",Transport.TCP, iceServerList);
phone.setAccount("John", "******", "John Smit", "jsmit");
```

5.2 User Agent: Set Listeners (Callbacks)

```
phone.setListener(new AudioCodesEventListener() {
    @Override
    public void loginStateChanged(Boolean isLogin, String cause) {
// place your code here (remember that this is being called
//on the WebRTC SDK thread, not on the UI thread)
    }

    @Override
    public void incomingCall(AudioCodesSession session, InfoAlert
infoAlert) {
// place your code here (remember that this is being called
//on the WebRTC SDK thread, not on the UI thread)
    }
});
```

5.3 User Agent login: Connection to SBC Server and Login

```
phone.login(getApplicationContext()); // getApplicationContext is
an Android method available for Activities and Services
```

5.4 Make a Call

```
Boolean withVideo = true;
AudioCodesSession activeCall = phone.call("jane", withVideo,
null);
```

5.5 Send DTMF During a Call

```
activeCall.sendDTMF(DTMF.NINE);
```

5.6 Send SIP Message During a Call

```
activeCall.sendInfo(String);
```

5.7 Mute / Unmute During a Call

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

5.8 Accept Incoming Call (with video)

```
incomingCall.answer(null, true);
```

5.9 Reject Incoming Call

```
incomingCall.reject(null);
```

5.10 Terminate a Call

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

5.11 Use of Video

Include the *ac_webrtc_video* UI element in the XML file for your call activity:

```
<include
    android:id="@+id/my_ac_webrtc_video"
    layout="@layout/ac_webrtc_video"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

Update the position of your local render screen:

```
activeCall.setLocalRenderPosition(70,60);
```

Call showVideo with your call activity; the WebRTC SDK will locate the *ac_webrtc_video* UI element and use it to display the remote and the local video:

```
activeCall.showVideo(CallActivity.this);
```

International Headquarters
Naimi Park
6 Ofra Haza Street
Or Yehuda, 6032303, Israel
Tel: +972-3-976-4000
Fax: +972-3-976-4040

AudioCodes Inc.
80 Kingsbridge Rd
Piscataway, NJ 08854, USA
Tel: +1-732-469-0880
Fax: +1-732-469-2298

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

©2026 AudioCodes Ltd. All rights reserved. AudioCodes, AC, HD VoIP, HD VoIP Sounds Better, IPmedia, Mediant, MediaPack, What's Inside Matters, OSN, SmartTAP, User Management Pack, VMAS, VoIPerfect, VoIPerfectHD, Your Gateway To VoIP, 3GX, AudioCodes One Voice, AudioCodes Meeting Insights, and AudioCodes Room Experience are trademarks or registered trademarks of AudioCodes Limited. All other products or trademarks are property of their respective owners. Product specifications are subject to change without notice.

Document #: LTRT-14098