

# **PortSIP VoIP SDK Manual for Android**

Version 16.5  
Fri Jul 26 2019

# Table of Contents

Welcome to the PortSIP VoIP SDK .....	3
Getting Started .....	3
Contents .....	3
SDK User Manual .....	3
Website .....	3
Support .....	3
Installation Prerequisites .....	3
Frequently Asked Questions .....	3
1. Where can I download the PortSIP VoIP SDK for trial? .....	3
2. How can I compile the sample project? .....	4
3. How can I create a new project with PortSIP VoIP SDK? .....	4
4. How can I test the P2P call (without SIP server)? .....	4
5. Is the SDK thread safe? .....	4
Module Index .....	5
Hierarchical Index .....	6
Class Index .....	7
Module Documentation .....	8
SDK Callback events .....	8
Register events .....	8
Call events .....	9
Refer events .....	12
Signaling events .....	13
MWI events .....	14
DTMF events .....	15
INFO/OPTIONS message events .....	16
Presence events .....	16
Play audio and video file finished events .....	19
RTP callback events .....	20
SDK functions .....	21
Initialize and register functions .....	22
Audio and video codecs functions .....	25
Additional settings functions .....	28
Access SIP message header functions .....	33
Audio and video functions .....	35
Call functions .....	38
Refer functions .....	42
Send audio and video stream functions .....	44
RTP packets, Audio stream and video stream callback .....	46
Record functions .....	47
Play audio and video file to remote functions .....	48
Conference functions .....	49
RTP and RTCP QOS functions .....	51
RTP statistics functions .....	52
Audio effect functions .....	54
Send OPTIONS/INFO/MESSAGE functions .....	55
Class Documentation .....	60
com.portsip.PortSipEnumDefine.AUDIODECODEC .....	60
com.portsip.PortSipEnumDefine.AudioDevice .....	61
com.portsip.OnPortSIPEvent .....	62
com.portsip.PortSIPCameraCapturer .....	64
com.portsip.PortSipEnumDefine .....	65
com.portsip.PortSipErrorcode .....	68
com.portsip.PortSipSdk .....	71
com.portsip.PortSIPVideoRenderer .....	75
Index .....	76

# Welcome to the PortSIP VoIP SDK

Create your SIP-based application for multiple platforms (iOS, Android, Windows, Mac OS and Linux) with our SDK.

The rewarding PortSIP VoIP SDK is a powerful and versatile set of tools that dramatically accelerate SIP application development. It includes a suite of stacks, SDKs, and some Sample projects, with each of them enables developers to combine all the necessary components to create an ideal development environment for every application's specific needs.

The PortSIP VoIP SDK complies with IETF and 3GPP standards, and is IMS-compliant (3GPP/3GPP2, TISPAN and PacketCable 2.0). These high performance SDKs provide unified API layers for full user control and flexibility.

## Getting Started

You can download PortSIP VoIP SDK Sample projects at our [Website](#). Samples include demos for VC++, C#, VB.NET, Delphi XE, XCode (for iOS and Mac OS), Eclipse (Java for Android) with the sample project source code provided (with SDK source code exclusive). The sample projects demonstrate how to create a powerful SIP application with our SDK easily and quickly.

## Contents

The sample package for downloading contains almost all of materials for PortSIP SDK: documentation, Dynamic/Static libraries, sources, headers, datasheet, and everything else a SDK user might need!

## SDK User Manual

To be started with, it is recommended to read the documentation of PortSIP VoIP SDK, [SDK User Manual page](#), which gives a brief description of each API functions.

## Website

Some general interest or often changing PortSIP SDK information will be posted on the [PortSIP website](#) in real time. The release contains links to the site, so while browsing you may see occasional broken links if you are not connected to the Internet. To be sure everything needed for using the PortSIP VoIP SDK has been contained within the release.

## Support

Please send email to our [Support team](#) if you need any help.

## Installation Prerequisites

To use PortSIP VoIP/IMS SDK for Android for development, SDK version with later than API-9 is required.

## Frequently Asked Questions

### 1. Where can I download the PortSIP VoIP SDK for trial?

All sample projects of the PortSIP VoIP SDK can be found and downloaded at:  
<https://www.portsip.com/download-portsip-voip-sdk/>  
<https://www.PortSIP.com/portsip-voip-sdk>.

## 2. How can I compile the sample project?

1. Download the sample project from PortSIP website.
2. Extract the .zip file.
3. Open the project by your Eclipse or Android studio:
4. Compile the sample project directly. The trial version SDK allows you a 2-3 minutes conversation.

## 3. How can I create a new project with PortSIP VoIP SDK?

1. Download the sample project and evaluation SDK and extract it to a specified directory
  2. Run Eclipse and create a new Android Application Project
  3. Copy all files from libs directory under extracted directory to the libs directory of your new application.
  4. Import the dependent class from the SDK. For example:

```
import com.portsip.OnPortSIPEvent;  
import com.portsip.PortSipSdk;
```
  5. Inherit the interface OnPortSIPEvent to process the callback events.
  6. Initialize SDK. For example:

```
mPortSIPSDK = new PortSipSdk();  
mPortSIPSDK.setOnPortSIPEvent(instanceofOnPortSIPEvent);  
mPortSIPSDK.CreateCallManager(context);  
mPortSIPSDK.initialize(...);
```
- For more details please refer to the Sample project source code.

## 4. How can I test the P2P call (without SIP server)?

1. Download and extract the SDK sample project ZIP file into local. Compile and run the "P2PSample" project.
2. Run the P2PSample on two devices. For example, run it on device A and device B, and IP address for A is 192.168.1.10, IP address for B is 192.168.1.11.
3. Enter a user name and password on A. For example, enter user name 111, and password aaa (you can enter anything for the password as the SDK will ignore it). Enter a user name and password on B. For example, enter user name 222, and password aaa.
4. Click the "Initialize" button on A and B. If the default port 5060 is already in use, the P2PSample will prompt "Initialize failure". In case of this, please click the "Uninitialize" button and change the local port, and click the "Initialize" button again.
5. The log box will appear "Initialized" if the SDK is successfully initialized.
6. To make call from A to B, enter "sip:222@192.168.1.11" and click "Dial" button; while to make call from B to A, enter "sip:111@192.168.1.10".

Note: If the local sip port is changed to other port, for example, A is using local port 5080, and B is using local port 6021, to make call from A to B, please enter "sip:222@192.168.1.11:6021" and dial; while to make call from B to A, enter "sip:111@192.168.1.10:5080".

## 5. Is the SDK thread safe?

Yes, the SDK is thread safe. You can call any of the API functions without the need to consider the multiple threads. Note: the SDK allows to call API functions in callback events directly - except for the "onAudioRawCallback", "onVideoRawCallback", "onReceivedRtpPacket", "onSendingRtpPacket" callbacks.

# Module Index

## Modules

Here is a list of all modules:

SDK Callback events .....	8
Register events .....	8
Call events .....	9
Refer events.....	12
Signaling events .....	13
MWI events.....	14
DTMF events .....	15
INFO/OPTIONS message events.....	16
Presence events .....	16
Play audio and video file finished events.....	19
RTP callback events .....	20
SDK functions .....	21
Initialize and register functions .....	22
Audio and video codecs functions.....	25
Additional settings functions.....	28
Access SIP message header functions.....	33
Audio and video functions .....	35
Call functions .....	38
Refer functions .....	42
Send audio and video stream functions .....	44
RTP packets, Audio stream and video stream callback.....	46
Record functions .....	47
Play audio and video file to remote functions .....	48
Conference functions .....	49
RTP and RTCP QOS functions .....	51
RTP statistics functions .....	52
Audio effect functions.....	54
Send OPTIONS/INFO/MESSAGE functions .....	55

# Hierarchical Index

## Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

com.portsip.PortSipEnumDefine.AUDIOCODEC .....	60
com.portsip.PortSipEnumDefine.AudioDevice .....	61
CapturerObserver	
com.portsip.PortSIPCameraCapturer .....	64
com.portsip.OnPortSIPEvent .....	62
com.portsip.PortSipEnumDefine .....	65
com.portsip.PortSipErrorCode .....	68
com.portsip.PortSipSdk .....	71
SurfaceViewRenderer	
com.portsip.PortSIPVideoRenderer .....	75

# Class Index

## Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

<a href="#"><u>com.portsip.PortSipEnumDefine.AUDIOCODEC</u></a> .....	60
<a href="#"><u>com.portsip.PortSipEnumDefine.AudioDevice</u></a> .....	61
<a href="#"><u>com.portsip.OnPortSIPEvent</u></a> .....	62
<a href="#"><u>com.portsip.PortSIPCameraCapturer</u></a> .....	64
<a href="#"><u>com.portsip.PortSipEnumDefine</u></a> .....	65
<a href="#"><u>com.portsip.PortSipErrorcode</u></a> .....	68
<a href="#"><u>com.portsip.PortSipSdk</u></a> .....	71
<a href="#"><u>com.portsip.PortSIPVideoRenderer</u></a> .....	75

# Module Documentation

## SDK Callback events

### Modules

- [Register events](#)
  - [Call events](#)
  - [Refer events](#)
  - [Signaling events](#)
  - [MWI events](#)
  - [DTMF events](#)
  - [INFO/OPTIONS message events](#)
  - [Presence events](#)
  - [Play audio and video file finished events](#)
  - [RTP callback events](#)
- 

### Detailed Description

SDK Callback events

## Register events

### Functions

- void [com.portsip.OnPortSIPEvent.onRegisterSuccess](#) (String *reason*, int *code*, String *sipMessage*)
  - void [com.portsip.OnPortSIPEvent.onRegisterFailure](#) (String *reason*, int *code*, String *sipMessage*)
- 

### Detailed Description

Register events

---

## Function Documentation

**void com.portsip.OnPortSIPEvent.onRegisterSuccess (String *reason*, int *code*, String *sipMessage*)**

When successfully registered to server, this event will be triggered.

#### Parameters:

<i>reason</i>	The status text.
<i>code</i>	The status code.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onRegisterFailure (String *reason*, int *code*, String *sipMessage*)**

If failed to register to SIP server, this event will be triggered.

#### Parameters:

<i>reason</i>	The status text.
<i>code</i>	The status code.
<i>sipMessage</i>	The SIP message received.

## Call events

### Functions

- void [com.portsip.OnPortSIPEvent.onInviteIncoming](#) (long sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteTrying](#) (long sessionId)
- void [com.portsip.OnPortSIPEvent.onInviteSessionProgress](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsEarlyMedia, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteRinging](#) (long sessionId, String statusText, int statusCode, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteAnswered](#) (long sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteFailure](#) (long sessionId, String reason, int code, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteUpdated](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteConnected](#) (long sessionId)
- void [com.portsip.OnPortSIPEvent.onInviteBeginningForward](#) (String forwardTo)
- void [com.portsip.OnPortSIPEvent.onInviteClosed](#) (long sessionId)
- void [com.portsip.OnPortSIPEvent.onDialogStateUpdated](#) (String BLFMonitoredUri, String BLFDialogState, String BLFDialogId, String BLFDialogDirection)
- void [com.portsip.OnPortSIPEvent.onRemoteHold](#) (long sessionId)
- void [com.portsip.OnPortSIPEvent.onRemoteUnHold](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo)

---

### Detailed Description

---

### Function Documentation

**void com.portsip.OnPortSIPEvent.onInviteIncoming (long *sessionId*, String *callerDisplayName*, String *caller*, String *calleeDisplayName*, String *callee*, String *audioCodecs*, String *videoCodecs*, boolean *existsAudio*, boolean *existsVideo*, String *sipMessage*)**

When a call is coming, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>callerDisplayName</i>	The display name of caller
<i>caller</i>	The caller.
<i>calleeDisplayName</i>	The display name of callee.
<i>callee</i>	The callee.
<i>audioCodecs</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecs</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	By setting to true, it means that this call include the audio.

<i>existsVideo</i>	By setting to true, it means that this call include the video.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteTrying (long *sessionId*)**

If the outgoing call is being processed, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onInviteSessionProgress (long *sessionId*, String *audioCodecs*, String *videoCodecs*, boolean *existsEarlyMedia*, boolean *existsAudio*, boolean *existsVideo*, String *sipMessage*)**

Once the caller received the "183 session progress" message, this event would be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>audioCodecs</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecs</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsEarlyMedia</i>	By setting to true it means the call has early media.
<i>existsAudio</i>	By setting to true it means this call include the audio.
<i>existsVideo</i>	By setting to true it means this call include the video.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteRinging (long *sessionId*, String *statusText*, int *statusCode*, String *sipMessage*)**

If the outgoing call is ringing, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>statusText</i>	The status text.
<i>statusCode</i>	The status code.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteAnswered (long *sessionId*, String *callerDisplayName*, String *caller*, String *calleeDisplayName*, String *callee*, String *audioCodecs*, String *videoCodecs*, boolean *existsAudio*, boolean *existsVideo*, String *sipMessage*)**

If the remote party answered the call, this event would be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>callerDisplayName</i>	The display name of caller
<i>caller</i>	The caller.
<i>calleeDisplayName</i>	The display name of callee.
<i>callee</i>	The callee.
<i>audioCodecs</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecs</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	By setting to true, this call includes the audio.
<i>existsVideo</i>	By setting to true, this call includes the video.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteFailure (long *sessionId*, String *reason*, int *code*, String *sipMessage*)**

This event will be triggered if the outgoing call fails.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>reason</i>	The failure reason.
<i>code</i>	The failure code.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteUpdated (long *sessionId*, String *audioCodecs*, String *videoCodecs*, boolean *existsAudio*, boolean *existsVideo*, String *sipMessage*)**

This event will be triggered when remote party updates the call.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>audioCodecs</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecs</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	By setting to true, this call includes the audio.
<i>existsVideo</i>	By setting to true, this call includes the video.
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onInviteConnected (long *sessionId*)**

This event will be triggered when UAC sent/UAS received ACK (the call is connected). Some functions (hold, updateCall etc..) can be called only after the call connected, otherwise the functions will return error.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onInviteBeginingForward (String *forwardTo*)**

If the enableCallForward method is called and a call is incoming, the call will be forwarded automatically and this event will be triggered.

**Parameters:**

<i>forwardTo</i>	The target SIP URI of the call forwarding.
------------------	--

**void com.portsip.OnPortSIPEvent.onInviteClosed (long *sessionId*)**

This event is triggered once remote side ends the call.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onDialogStateUpdated (String *BLFMonitoredUri*, String *BLFDialogState*, String *BLFDialogId*, String *BLFDialogDirection*)**

If a user subscribed and his dialog status monitored, when the monitored user is holding a call or is being rang, this event will be triggered.

**Parameters:**

<i>BLFMonitoredUri</i>	the monitored user's URI
<i>BLFDialogState</i>	- the status of the call
<i>BLFDialogId</i>	- the id of the call
<i>BLFDialogDirection</i>	- the direction of the call

**void com.portsip.OnPortSIPEvent.onRemoteHold (long *sessionId*)**

If the remote side places the call on hold, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onRemoteUnHold (long *sessionId*, String *audioCodecs*, String *videoCodecs*, boolean *existsAudio*, boolean *existsVideo*)**

If the remote side un-holds the call, this event will be triggered

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>audioCodecs</i>	The matched audio codecs. It's separated by "#" if there are more than one codec.
<i>videoCodecs</i>	The matched video codecs. It's separated by "#" if there are more than one codec.
<i>existsAudio</i>	By setting to true, this call includes the audio.
<i>existsVideo</i>	By setting to true, this call includes the video.

## Refer events

### Functions

- void [com.portsip.OnPortSIPEvent.onReceivedRefer](#) (long *sessionId*, long *referId*, String *to*, String *from*, String *referSipMessage*)
- void [com.portsip.OnPortSIPEvent.onReferAccepted](#) (long *sessionId*)
- void [com.portsip.OnPortSIPEvent.onReferRejected](#) (long *sessionId*, String *reason*, int *code*)
- void [com.portsip.OnPortSIPEvent.onTransferTrying](#) (long *sessionId*)
- void [com.portsip.OnPortSIPEvent.onTransferRinging](#) (long *sessionId*)
- void [com.portsip.OnPortSIPEvent.onACTVTransferSuccess](#) (long *sessionId*)
- void [com.portsip.OnPortSIPEvent.onACTVTransferFailure](#) (long *sessionId*, String *reason*, int *code*)

## Detailed Description

## Function Documentation

**void com.portsip.OnPortSIPEvent.onReceivedRefer (long *sessionId*, long *referId*, String *to*, String *from*, String *referSipMessage*)**

This event will be triggered once received a REFER message.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>referId</i>	The ID of the REFER message. Pass it to <code>acceptRefer</code> or <code>rejectRefer</code>
<i>to</i>	The refer target.
<i>from</i>	The sender of REFER message.
<i>referSipMessage</i>	The SIP message of "REFER". Pass it to "acceptRefer" function.

**void com.portsip.OnPortSIPEvent.onReferAccepted (long *sessionId*)**

This callback will be triggered once remote side calls "acceptRefer" to accept the REFER

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onReferRejected (long *sessionId*, String *reason*, int *code*)**

This callback will be triggered once remote side calls "rejectRefer" to reject the REFER

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>reason</i>	Reject reason.
<i>code</i>	Reject code.

**void com.portsip.OnPortSIPEvent.onTransferTrying (long *sessionId*)**

When the refer call is being processed, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onTransferRinging (long *sessionId*)**

When the refer call is ringing, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onACTVTransferSuccess (long *sessionId*)**

When the refer call succeeds, this event will be triggered. The ACTV means Active. For example, A establishes the call with B, A transfers B to C, C accepts the refer call, and A will receive this event.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

**void com.portsip.OnPortSIPEvent.onACTVTransferFailure (long *sessionId*, String *reason*, int *code*)**

When the refer call fails, this event will be triggered. The ACTV means Active. For example, A establish the call with B, A transfers B to C, C rejects this refer call, and A will receive this event.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>reason</i>	The error reason.
<i>code</i>	The error code.

## Signaling events

### Functions

- void [com.portsip.OnPortSIPEvent.onReceivedSignaling](#) (long sessionId, String message)
- void [com.portsip.OnPortSIPEvent.onSendingSignaling](#) (long sessionId, String message)

### Detailed Description

---

## Function Documentation

**void com.portsip.OnPortSIPEvent.onReceivedSignaling (long *sessionId*, String *message*)**

This event will be triggered when receiving a SIP message. This event is disabled by default. To enable, use `enableCallbackSignaling`.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>message</i>	The received SIP message.

**void com.portsip.OnPortSIPEvent.onSendingSignaling (long *sessionId*, String *message*)**

This event will be triggered when sent a SIP message. This event is disabled by default. To enable, use `enableCallbackSignaling`.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>message</i>	The sent SIP message.

## MWI events

### Functions

- void [com.portsip.OnPortSIPEvent.onWaitingVoiceMessage](#) (String *messageAccount*, int *urgentNewMessageCount*, int *urgentOldMessageCount*, int *newMessageCount*, int *oldMessageCount*)
- void [com.portsip.OnPortSIPEvent.onWaitingFaxMessage](#) (String *messageAccount*, int *urgentNewMessageCount*, int *urgentOldMessageCount*, int *newMessageCount*, int *oldMessageCount*)

---

## Detailed Description

---

## Function Documentation

**void com.portsip.OnPortSIPEvent.onWaitingVoiceMessage (String *messageAccount*, int *urgentNewMessageCount*, int *urgentOldMessageCount*, int *newMessageCount*, int *oldMessageCount*)**

If there is the waiting voice message (MWI), this event will be triggered.

**Parameters:**

<i>messageAccount</i>	Voice message account
<i>urgentNewMessageCount</i>	Count of new urgent messages.
<i>urgentOldMessageCount</i>	Count of history urgent message.
<i>newMessageCount</i>	Count of new messages.
<i>oldMessageCount</i>	Count of history messages.

**void com.portsip.OnPortSIPEvent.onWaitingFaxMessage (String *messageAccount*, int *urgentNewMessageCount*, int *urgentOldMessageCount*, int *newMessageCount*, int *oldMessageCount*)**

If there is waiting fax message (MWI), this event will be triggered.

**Parameters:**

<i>messageAccount</i>	Fax message account
<i>urgentNewMessageCount</i>	Count of new urgent messages.
<i>urgentOldMessageCount</i>	Count of history urgent messages.
<i>newMessageCount</i>	Count of new messages.
<i>oldMessageCount</i>	Count of old messages.

## DTMF events

### Functions

- void [com.portsip.OnPortSIPEvent.onRecvDtmfTone](#) (long *sessionId*, int *tone*)

### Detailed Description

### Function Documentation

**void com.portsip.OnPortSIPEvent.onRecvDtmfTone (long *sessionId*, int *tone*)**

This event will be triggered when receiving a DTMF tone from remote side.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
<i>tone</i>	

code	Description
0	The DTMF tone 0.
1	The DTMF tone 1.
2	The DTMF tone 2.
3	The DTMF tone 3.
4	The DTMF tone 4.
5	The DTMF tone 5.
6	The DTMF tone 6.
7	The DTMF tone 7.
8	The DTMF tone 8.
9	The DTMF tone 9.
10	The DTMF tone *.
11	The DTMF tone #.
12	The DTMF tone A.
13	The DTMF tone B.
14	The DTMF tone C.
15	The DTMF tone D.
16	The DTMF tone FLASH.

## INFO/OPTIONS message events

### Functions

- void [com.portsip.OnPortSIPEvent.onRecvOptions](#) (String optionsMessage)
- void [com.portsip.OnPortSIPEvent.onRecvInfo](#) (String infoMessage)
- void [com.portsip.OnPortSIPEvent.onRecvNotifyOfSubscription](#) (long subscribeId, String notifyMessage, byte[] messageData, int messageDataLength)

---

### Detailed Description

---

### Function Documentation

#### **void com.portsip.OnPortSIPEvent.onRecvOptions (String optionsMessage)**

This event will be triggered when receiving the OPTIONS message.

##### Parameters:

<i>optionsMessage</i>	The received whole OPTIONS message in text format.
-----------------------	--

#### **void com.portsip.OnPortSIPEvent.onRecvInfo (String infoMessage)**

This event will be triggered when receiving the INFO message.

##### Parameters:

<i>infoMessage</i>	The whole INFO message received in text format.
--------------------	---

#### **void com.portsip.OnPortSIPEvent.onRecvNotifyOfSubscription (long subscribeId, String notifyMessage, byte [] messageData, int messageDataLength)**

This event will be triggered when receiving a NOTIFY message of the subscription.

##### Parameters:

<i>subscribeId</i>	The ID of SUBSCRIBE request.
<i>notifyMessage</i>	The received INFO message in text format.
<i>messageData</i>	The received message body. It's can be either text or binary data.
<i>messageDataLength</i>	The length of "messageData".

## Presence events

### Functions

- void [com.portsip.OnPortSIPEvent.onPresenceRecvSubscribe](#) (long subscribeId, String fromDisplayName, String from, String subject)
- void [com.portsip.OnPortSIPEvent.onPresenceOnline](#) (String fromDisplayName, String from, String stateText)
- void [com.portsip.OnPortSIPEvent.onPresenceOffline](#) (String fromDisplayName, String from)
- void [com.portsip.OnPortSIPEvent.onRecvMessage](#) (long sessionId, String mimeType, String subMimeType, byte[] messageData, int messageDataLength)

- void [com.portsip.OnPortSIPEvent.onRecvOutOfDialogMessage](#) (String fromDisplayName, String from, String toDisplayName, String to, String mimeType, String subMimeType, byte[] messageData, int messageDataLength, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onSendMessageSuccess](#) (long sessionId, long messageId)
- void [com.portsip.OnPortSIPEvent.onSendMessageFailure](#) (long sessionId, long messageId, String reason, int code)
- void [com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageSuccess](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to)
- void [com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageFailure](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String reason, int code)
- void [com.portsip.OnPortSIPEvent.onSubscriptionFailure](#) (long subscribeId, int statusCode)
- void [com.portsip.OnPortSIPEvent.onSubscriptionTerminated](#) (long subscribeId)

## Detailed Description

## Function Documentation

**void com.portsip.OnPortSIPEvent.onPresenceRecvSubscribe (long *subscribeId*, String *fromDisplayName*, String *from*, String *subject*)**

This event will be triggered when receiving the SUBSCRIBE request from a contact.

**Parameters:**

<i>subscribeId</i>	The ID of SUBSCRIBE request.
<i>fromDisplayName</i>	The display name of contact.
<i>from</i>	The contact who sends the SUBSCRIBE request.
<i>subject</i>	The subject of the SUBSCRIBE request.

**void com.portsip.OnPortSIPEvent.onPresenceOnline (String *fromDisplayName*, String *from*, String *stateText*)**

When the contact is online or changes presence status, this event will be triggered.

**Parameters:**

<i>fromDisplayName</i>	The display name of contact.
<i>from</i>	The contact who sends the SUBSCRIBE request.
<i>stateText</i>	The presence status text.

**void com.portsip.OnPortSIPEvent.onPresenceOffline (String *fromDisplayName*, String *from*)**

When the contact is offline, this event will be triggered.

**Parameters:**

<i>fromDisplayName</i>	The display name of contact.
<i>from</i>	The contact who sends the SUBSCRIBE request

**void com.portsip.OnPortSIPEvent.onRecvMessage (long *sessionId*, String *mimeType*, String *subMimeType*, byte [] *messageData*, int *messageDataLength*)**

This event will be triggered when receiving a MESSAGE message in dialog.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>mimeType</i>	The message mime type.
<i>subMimeType</i>	The message sub mime type.
<i>messageData</i>	The received message body. It can be text or binary data. Use the mimeType

	and subMimeType to differentiate them. For example, if the mimeType is "text" and subMimeType is "plain", "messageData" is text message body. If the mimeType is "application" and subMimeType is "vnd.3gpp.sms", "messageData" is binary message body.
<i>messageDataLength</i>	The length of "messageData".

**void com.portsip.OnPortSIPEvent.onRecvOutOfDialogMessage (String fromDisplayName, String from, String toDisplayName, String to, String mimeType, String subMimeType, byte [] messageData, int messageDataLength, String sipMessage)**

This event will be triggered when receiving a MESSAGE message out of dialog. For example pager message.

**Parameters:**

<i>fromDisplayName</i>	The display name of sender.
<i>from</i>	The message sender.
<i>toDisplayName</i>	The display name of receiver.
<i>to</i>	The receiver.
<i>mimeType</i>	The message mime type.
<i>subMimeType</i>	The message sub mime type.
<i>messageData</i>	The received message body. It can be text or binary data. Use the mimeType and subMimeType to differentiate them. For example, if the mimeType is "text" and subMimeType is "plain", "messageData" is text message body. If the mimeType is "application" and subMimeType is "vnd.3gpp.sms", "messageData" is binary message body.
<i>messageDataLength</i>	The length of "messageData".
<i>sipMessage</i>	The SIP message received.

**void com.portsip.OnPortSIPEvent.onSendMessageSuccess (long sessionId, long messageId)**

If the message is sent successfully in dialog, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>messageId</i>	The message ID. It's equal to the return value of sendMessage function.

**void com.portsip.OnPortSIPEvent.onSendMessageFailure (long sessionId, long messageId, String reason, int code)**

If the message is failed to be sent out of dialog, this event will be triggered.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>messageId</i>	The message ID. It's equal to the return value of sendMessage function.
<i>reason</i>	The failure reason.
<i>code</i>	Failure code.

**void com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageSuccess (long messageId, String fromDisplayName, String from, String toDisplayName, String to)**

If the message is sent successfully out of dialog, this event will be triggered.

**Parameters:**

<i>messageId</i>	The message ID. It's equal to the return value of SendOutOfDialogMessage function.
<i>fromDisplayName</i>	The display name of message sender.
<i>from</i>	The message sender.

<i>toDisplayName</i>	The display name of message receiver.
<i>to</i>	The message receiver.

**void com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageFailure (long *messageId*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*, String *reason*, int *code*)**

If the message failed to be sent out of dialog, this event would be triggered.

**Parameters:**

<i>messageId</i>	The message ID. It's equal to the return value of SendOutOfDialogMessage function.
<i>fromDisplayName</i>	The display name of message sender
<i>from</i>	The message sender.
<i>toDisplayName</i>	The display name of message receiver.
<i>to</i>	The message receiver.
<i>reason</i>	The failure reason.
<i>code</i>	The failure code.

**void com.portsip.OnPortSIPEvent.onSubscriptionFailure (long *subscribeId*, int *statusCode*)**

This event will be triggered on sending SUBSCRIBE failure.

**Parameters:**

<i>subscribeId</i>	The ID of SUBSCRIBE request.
<i>statusCode</i>	The status code.

**void com.portsip.OnPortSIPEvent.onSubscriptionTerminated (long *subscribeId*)**

This event will be triggered when a SUBSCRIPTION is terminated or expired.

**Parameters:**

<i>subscribeId</i>	The ID of SUBSCRIBE request.
--------------------	------------------------------

## Play audio and video file finished events

### Functions

- void [com.portsip.OnPortSIPEvent.onPlayAudioFileFinished](#) (long *sessionId*, String *fileName*)
- void [com.portsip.OnPortSIPEvent.onPlayVideoFileFinished](#) (long *sessionId*)

## Detailed Description

### Function Documentation

**void com.portsip.OnPortSIPEvent.onPlayAudioFileFinished (long *sessionId*, String *fileName*)**

If called playAudioFileToRemote function with no loop mode, this event will be triggered once the file play finished.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

<i>fileName</i>	The play file name.
-----------------	---------------------

**void com.portsip.OnPortSIPEvent.onPlayVideoFileFinished (long *sessionId*)**

If called playVideoFileToRemote function with no loop mode, this event will be triggered once the file play finished.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
------------------	-----------------------------

## RTP callback events

### Functions

- void [com.portsip.OnPortSIPEvent.onReceivedRTPPacket](#) (long *sessionId*, boolean *isAudio*, byte[] *RTPPacket*, int *packetSize*)
- void [com.portsip.OnPortSIPEvent.onSendingRTPPacket](#) (long *sessionId*, boolean *isAudio*, byte[] *RTPPacket*, int *packetSize*)
- void [com.portsip.OnPortSIPEvent.onAudioRawCallback](#) (long *sessionId*, int *enum\_audioCallbackMode*, byte[] *data*, int *dataLength*, int *samplingFreqHz*)
- void [com.portsip.OnPortSIPEvent.onVideoRawCallback](#) (long *sessionId*, int *enum\_videoCallbackMode*, int *width*, int *height*, byte[] *data*, int *dataLength*)

### Detailed Description

### Function Documentation

**void com.portsip.OnPortSIPEvent.onReceivedRTPPacket (long *sessionId*, boolean *isAudio*, byte [] *RTPPacket*, int *packetSize*)**

If setRTPCallback function is called to enable the RTP callback, this event will be triggered once receiving a RTP packet.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>isAudio</i>	If the received RTP packet is of audio, this parameter returns true; otherwise false.
<i>RTPPacket</i>	The memory of whole RTP packet.
<i>packetSize</i>	The size of received RTP Packet. Remarks

Don't call any SDK API functions in this event directly. If you want to call the API functions or other code which is time-consuming, you should post a message to another thread and execute SDK API functions or other code in another thread.

**void com.portsip.OnPortSIPEvent.onSendingRTPPacket (long *sessionId*, boolean *isAudio*, byte [] *RTPPacket*, int *packetSize*)**

If setRTPCallback function is called to enable the RTP callback, this event will be triggered once sending a RTP packet.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>isAudio</i>	If the received RTP packet is of audio, this parameter returns true; otherwise false.

<i>RTPPacket</i>	The memory of whole RTP packet.
<i>packetSize</i>	The size of received RTP Packet. Remarks

Don't call any SDK API functions in this event directly. If you want to call the API functions or other code which is time-consuming, you should post a message to another thread and execute SDK API functions or other code in another thread.

**void com.portsip.OnPortSIPEvent.onAudioRawCallback (long *sessionId*, int *enum\_audioCallbackMode*, byte [] *data*, int *dataLength*, int *samplingFreqHz*)**

This event will be triggered once receiving the audio packets if called [enableAudioStreamCallback](#) function.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>enum_audioCallbackMode</i>	The type passed in enableAudioStreamCallback function. Below types allowed: <a href="#">ENUM_AUDIOSTREAM_NONE</a> , <a href="#">ENUM_AUDIOSTREAM_LOCAL_MIX</a> , <a href="#">ENUM_AUDIOSTREAM_LOCAL_PER_CHANNEL</a> , <a href="#">ENUM_AUDIOSTREAM_REMOTE_MIX</a> , <a href="#">ENUM_AUDIOSTREAM_REMOTE_PER_CHANNEL</a> .
<i>data</i>	The memory of audio stream. It's in PCM format.
<i>dataLength</i>	The data size.
<i>samplingFreqHz</i>	The audio stream sample in HZ. For example, 8000 or 16000. Remarks

Don't call any SDK API functions in this event directly. If you want to call the API functions or other code which is time-consuming, you should post a message to another thread and execute SDK API functions or other code in another thread.

**See also:**

[PortSipSdk::enableAudioStreamCallback](#)

**void com.portsip.OnPortSIPEvent.onVideoRawCallback (long *sessionId*, int *enum\_videoCallbackMode*, int *width*, int *height*, byte [] *data*, int *dataLength*)**

This event will be triggered once receiving the video packets if [enableVideoStreamCallback](#) function is called.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>enum_videoCallbackMode</i>	The type which is passed in enableVideoStreamCallback function. Below types allowed: <a href="#">ENUM_VIDEOSTREAM_NONE</a> , <a href="#">ENUM_VIDEOSTREAM_LOCAL</a> , <a href="#">ENUM_VIDEOSTREAM_REMOTE</a> , <a href="#">ENUM_VIDEOSTREAM_BOTH</a> .
<i>width</i>	The width of video image.
<i>height</i>	The height of video image.
<i>data</i>	The memory of video stream. It's in YUV420 format, YV12.
<i>dataLength</i>	The data size.

**See also:**

[PortSipSdk::enableVideoStreamCallback](#)

## SDK functions

### Modules

- [Initialize and register functions](#)
- [Audio and video codecs functions](#)
- [Additional settings functions](#)
- [Access SIP message header functions](#)

- [Audio and video functions](#)
- [Call functions](#)
- [Refer functions](#)
- [Send audio and video stream functions](#)
- [RTP packets, Audio stream and video stream callback](#)
- [Record functions](#)
- [Play audio and video file to remote functions](#)
- [Conference functions](#)
- [RTP and RTCP QOS functions](#)
- [RTP statistics functions](#)
- [Audio effect functions](#)
- [Send OPTIONS/INFO/MESSAGE functions](#)

## Detailed Description

### Initialize and register functions

#### Functions

- void [com.portsip.PortSipSdk.CreateCallManager](#) (Context context)
- void **com.portsip.PortSipSdk.setAudioManagerEvents** (AppRTCAudioManager.AudioManagerEvents audioManagerEvents)
- PortSipEnumDefine.AudioDevice **com.portsip.PortSipSdk.getSelectedAudioDevice** ()
- void [com.portsip.PortSipSdk.DeleteCallManager](#) ()
- int [com.portsip.PortSipSdk.initialize](#) (int enum\_transport, String localIP, int localSIPPort, int enum\_LogLevel, String LogPath, int maxLines, String agent, int audioDeviceLayer, int videoDeviceLayer, String TLSCertificatesRootPath, String TLSCipherList, boolean verifyTLSCertificate, String dnsServers)
- int [com.portsip.PortSipSdk.setInstanceId](#) (String instanceId)
- int [com.portsip.PortSipSdk.setUser](#) (String userName, String displayName, String authName, String password, String userDomain, String SIPServer, int SIPServerPort, String STUNServer, int STUNServerPort, String outboundServer, int outboundServerPort)
- void [com.portsip.PortSipSdk.removeUser](#) ()  
*remove user account info.*
- int [com.portsip.PortSipSdk.registerServer](#) (int expires, int retryTimes)
- int [com.portsip.PortSipSdk.refreshRegistration](#) (int expires)
- int [com.portsip.PortSipSdk.unregisterServer](#) ()
- int [com.portsip.PortSipSdk.setLicenseKey](#) (String key)

## Detailed Description

### Function Documentation

**void com.portsip.PortSipSdk.CreateCallManager (Context context)**

Create the callback handlers.

#### Parameters:

<i>context</i>	The context of application.
----------------	-----------------------------

### void com.portsip.PortSipSdk.DeleteCallManager ()

Release the callback Handlers.

### int com.portsip.PortSipSdk.initialize (int *enum\_transport*, String *localIP*, int *localSIPPort*, int *enum\_LogLevel*, String *LogPath*, int *maxLines*, String *agent*, int *audioDeviceLayer*, int *videoDeviceLayer*, String *TLSCertificatesRootPath*, String *TLSCipherList*, boolean *verifyTLSCertificate*, String *dnsServers*)

Initialize the SDK.

#### Parameters:

<i>enum_transport</i>	Transport for SIP signaling, which can be set as: <a href="#">ENUM_TRANSPORT_UDP</a> , <a href="#">ENUM_TRANSPORT_TCP</a> , <a href="#">ENUM_TRANSPORT_TLS</a> , <a href="#">ENUM_TRANSPORT_PERS</a> . The ENUM_TRANSPORT_PERS_UDP is the private PortSIP transport for anti-SIP blocking, which must work with the PERS. <a href="#">ENUM_TRANSPORT_PERS</a> . The ENUM_TRANSPORT_PERS_TCP is the private PortSIP transport for anti-SIP blocking, which must work with the PERS.
<i>localIP</i>	The local PC IP address (for example: 192.168.1.108). It will be used for sending and receiving SIP messages and RTP packets. If the local IP is provided in IPv6 format, the SDK will use IPv6. If you want the SDK to choose correct network interface (IP) automatically, please use "0.0.0.0" for IPv4, or ":::" for IPv6.
<i>localSIPPort</i>	The listening port for SIP message transmission, for example 5060.
<i>enum_LogLevel</i>	Set the application log level. The SDK will generate "PortSIP_Log_datatime.log" file if the log is enabled. <a href="#">ENUM_LOG_LEVEL_NONE</a> <a href="#">ENUM_LOG_LEVEL_DEBUG</a> <a href="#">ENUM_LOG_LEVEL_ERROR</a> <a href="#">ENUM_LOG_LEVEL_WARNING</a> <a href="#">ENUM_LOG_LEVEL_INFO</a> <a href="#">ENUM_LOG_LEVEL_DEBUG</a>
<i>LogPath</i>	The path for storing log file. The path (folder) specified MUST be existent.
<i>maxLines</i>	Theoretically, unlimited count of lines are supported depending on the device capability. For SIP client, it is recommended to limit it as ranging 1 - 100.
<i>agent</i>	The User-Agent header to be inserted in to SIP messages.
<i>audioDeviceLayer</i>	Specifies the audio device layer that should be using: 0 = Use the OS defaulted device. 1 = Virtual device, usually use this for the device that has no sound device installed.
<i>videoDeviceLayer</i>	Specifies the video device layer that should be using: 0 = Use the OS defaulted device. 1 = Use Virtual device, usually use this for the device that has no camera installed.
<i>TLSCertificatesRootPath</i>	Specify the TLS certificate path, from which the SDK will load the certificates automatically. Note: On Windows, this path will be ignored, and SDK will read the certificates from Windows certificates stored area instead.
<i>TLSCipherList</i>	Specify the TLS cipher list. This parameter is usually passed as empty so that the SDK will offer all available ciphers.
<i>verifyTLSCertificate</i>	Indicate if SDK will verify the TLS certificate or not. By setting to false, the SDK will not verify the validity of TLS certificate.
<i>dnsServers</i>	Additional Nameservers DNS servers. Value null indicates system DNS Server. Multiple servers will be split by ";", e.g "8.8.8.8;8.8.4.4"

#### Returns:

If the function succeeds, it returns value 0. If the function fails, it will return a specific error code

### int com.portsip.PortSipSdk.setInstanceId (String *instanceId*)

Set the instance Id, the outbound instanceId((RFC5626) ) used in contact headers.

**Parameters:**

<i>instanceId</i>	The SIP instance ID. If this function is not called, the SDK will generate an instance ID automatically. The instance ID MUST be unique on the same device (device ID or IMEI ID is recommended). Recommend to call this function to set the ID on Android devices.
-------------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setUser (String *userName*, String *displayName*, String *authName*, String *password*, String *userDomain*, String *SIPServer*, int *SIPServerPort*, String *STUNServer*, int *STUNServerPort*, String *outboundServer*, int *outboundServerPort*)**

Set user account info.

**Parameters:**

<i>userName</i>	Account (username) of the SIP, usually provided by an IP-Telephony service provider.
<i>displayName</i>	The name displayed. You can set it as your like, such as "James Kend". It's optional.
<i>authName</i>	Authorization user name (usually equal to the username).
<i>password</i>	User's password. It's optional.
<i>userDomain</i>	User domain; this parameter is optional, which allows to transfer an empty string if you are not using the domain.
<i>SIPServer</i>	SIP proxy server IP or domain, for example xx.xxx.xx.x or sip.xxx.com.
<i>SIPServerPort</i>	Port of the SIP proxy server, for example 5060.
<i>STUNServer</i>	Stun server for NAT traversal. It's optional and can be used to transfer empty string to disable STUN.
<i>STUNServerPort</i>	STUN server port. It will be ignored if the <i>outboundServer</i> is empty.
<i>outboundServer</i>	Outbound proxy server, for example sip.domain.com. It's optional and allows to transfer an empty string if not using the outbound server.
<i>outboundServerPort</i>	Outbound proxy server port, it will be ignored if the <i>outboundServer</i> is empty.

**Returns:**

If this function succeeds, it will return value 0. If it fails, it will return a specific error code.

**int com.portsip.PortSipSdk.registerServer (int *expires*, int *retryTimes*)**

Register to SIP proxy server (login to server)

**Parameters:**

<i>expires</i>	Time interval for registration refreshment, in seconds. The maximum of supported value is 3600. It will be inserted into SIP REGISTER message headers.
<i>retryTimes</i>	The maximum of retry attempts if failed to refresh the registration. By setting to $\leq 0$ , the attempt will be disabled and <i>onRegisterFailure</i> callback will be triggered when facing retry failure.

**Returns:**

If this function succeeds, it will return value 0. If fails, it will return a specific error code.

If the registration to server succeeds, *onRegisterSuccess* will be triggered; otherwise *onRegisterFailure* will be triggered.

**int com.portsip.PortSipSdk.refreshRegistration (int *expires*)**

Refresh the registration manually after successfully registered.

**Parameters:**

<i>expires</i>	Time interval for registration refreshment, in seconds. The maximum of supported value is 3600. It will be inserted into SIP REGISTER message
----------------	---

	headers.
--	----------

**Returns:**

If this function succeeds, it will return value 0. If fails, it will return a specific error code.  
 If the registration to server succeeds, onRegisterSuccess will be triggered; otherwise onRegisterFailure will be triggered.

**int com.portsip.PortSipSdk.unregisterServer ()**

Un-register from the SIP proxy server.

**Returns:**

If this function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setLicenseKey (String key)**

Set the license key. It must be called before setUser function.

**Parameters:**

key	The SDK license key. Please purchase from PortSIP.
-----	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Audio and video codecs functions

### Functions

- int [com.portsip.PortSipSdk.addAudioCodec](#) (int enum\_audiocodec)
- int [com.portsip.PortSipSdk.addVideoCodec](#) (int enum\_videocodec)
- boolean [com.portsip.PortSipSdk.isAudioCodecEmpty](#) ()
- boolean [com.portsip.PortSipSdk.isVideoCodecEmpty](#) ()
- int [com.portsip.PortSipSdk.setAudioCodecPayloadType](#) (int enum\_audiocodec, int payloadType)
- int [com.portsip.PortSipSdk.setVideoCodecPayloadType](#) (int enum\_videocodec, int payloadType)
- void [com.portsip.PortSipSdk.clearAudioCodec](#) ()
- void [com.portsip.PortSipSdk.clearVideoCodec](#) ()
- int [com.portsip.PortSipSdk.setAudioCodecParameter](#) (int enum\_audiocodec, String sdpParameter)
- int [com.portsip.PortSipSdk.setVideoCodecParameter](#) (int enum\_videocodec, String sdpParameter)

---

### Detailed Description

---

### Function Documentation

**int com.portsip.PortSipSdk.addAudioCodec (int enum\_audiocodec)**

Enable an audio codec, and it will be shown in SDP.

**Parameters:**

enum_audiocodec	Audio codec type, including: <a href="#">ENUM_AUDIOCODEC_G729</a> , <a href="#">ENUM_AUDIOCODEC_PCMA</a> , <a href="#">ENUM_AUDIOCODEC_PCMU</a> , <a href="#">ENUM_AUDIOCODEC_GSM</a> , <a href="#">ENUM_AUDIOCODEC_G722</a> , <a href="#">ENUM_AUDIOCODEC_ILBC</a> , <a href="#">ENUM_AUDIOCODEC_AMR</a> , <a href="#">ENUM_AUDIOCODEC_AMRWB</a> , <a href="#">ENUM_AUDIOCODEC_SPEEX</a> ,
-----------------	---

	<a href="#">ENUM_AUDIOCODEC_SPEEXWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACSWB</a> , <a href="#">ENUM_AUDIOCODEC_OPUS</a> , <a href="#">ENUM_AUDIOCODEC_DTMF</a> .
--	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.addVideoCodec (int *enum\_videocodec*)**

Enable a video codec, and it will be shown in SDP.

**Parameters:**

<i>enum_videocodec</i>	Video codec type. Supported types include <a href="#">ENUM_VIDEOCODEC_H264</a> , <a href="#">ENUM_VIDEOCODEC_VP8</a> . <a href="#">ENUM_VIDEOCODEC_VP9</a> .
------------------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**boolean com.portsip.PortSipSdk.isAudioCodecEmpty ()**

Detect if the audio codecs are enabled.

**Returns:**

If no audio codec enabled, it will return value true; otherwise it returns false.

**boolean com.portsip.PortSipSdk.isVideoCodecEmpty ()**

Detect if the video codecs are enabled.

**Returns:**

If no video codec enabled, it will return value true; otherwise it returns false.

**int com.portsip.PortSipSdk.setAudioCodecPayloadType (int *enum\_audiocodec*, int *payloadType*)**

Set the RTP payload type for dynamic audio codec.

**Parameters:**

<i>enum_audiocodec</i>	Audio codec types. Supported types include: <a href="#">ENUM_AUDIOCODEC_G729</a> , <a href="#">ENUM_AUDIOCODEC_PCMA</a> , <a href="#">ENUM_AUDIOCODEC_PCMU</a> , <a href="#">ENUM_AUDIOCODEC_GSM</a> , <a href="#">ENUM_AUDIOCODEC_G722</a> , <a href="#">ENUM_AUDIOCODEC_ILBC</a> , <a href="#">ENUM_AUDIOCODEC_AMR</a> , <a href="#">ENUM_AUDIOCODEC_AMRWB</a> , <a href="#">ENUM_AUDIOCODEC_SPEEX</a> , <a href="#">ENUM_AUDIOCODEC_SPEEXWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACSWB</a> , <a href="#">ENUM_AUDIOCODEC_OPUS</a> , <a href="#">ENUM_AUDIOCODEC_DTMF</a>
<i>payloadType</i>	The new RTP payload type that you want to set.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoCodecPayloadType (int *enum\_videocodec*, int *payloadType*)**

Set the RTP payload type for dynamic video codec.

**Parameters:**

<i>enum_videocodec</i>	Video codec type. Supported types include: <a href="#">ENUM_VIDEOCODEC_H264</a> , <a href="#">ENUM_VIDEOCODEC_VP8</a> . <a href="#">ENUM_VIDEOCODEC_VP9</a> .
<i>payloadType</i>	The new RTP payload type that you want to set.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.clearAudioCodec ()**

Remove all the enabled audio codecs.

**void com.portsip.PortSipSdk.clearVideoCodec ()**

Remove all the enabled video codecs.

**int com.portsip.PortSipSdk.setAudioCodecParameter (int *enum\_audiocodec*, String *sdpParameter*)**

Set the codec parameter for audio codec.

**Parameters:**

<i>enum_audiocodec</i>	Audio codec type. Supported types include: <a href="#">ENUM_AUDIOCODEC_G729</a> , <a href="#">ENUM_AUDIOCODEC_PCMA</a> , <a href="#">ENUM_AUDIOCODEC_PCMU</a> , <a href="#">ENUM_AUDIOCODEC_GSM</a> , <a href="#">ENUM_AUDIOCODEC_G722</a> , <a href="#">ENUM_AUDIOCODEC_ILBC</a> , <a href="#">ENUM_AUDIOCODEC_AMR</a> , <a href="#">ENUM_AUDIOCODEC_AMRWB</a> , <a href="#">ENUM_AUDIOCODEC_SPEEX</a> , <a href="#">ENUM_AUDIOCODEC_SPEEXWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACWB</a> , <a href="#">ENUM_AUDIOCODEC_ISACSWB</a> , <a href="#">ENUM_AUDIOCODEC_OPUS</a> , <a href="#">ENUM_AUDIOCODEC_DTMF</a>
<i>sdpParameter</i>	The parameter is in string format.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**See also:**

[PortSipEnumDefine](#)

**Remarks:**

Example:

```
setAudioCodecParameter(AUDIOCODEC_AMR, "mode-set=0; octet-align=1; robust-sorting=0")
```

**int com.portsip.PortSipSdk.setVideoCodecParameter (int *enum\_videocodec*, String *sdpParameter*)**

Set the codec parameter for video codec.

**Parameters:**

<i>enum_videocodec</i>	Video codec types. Supported types include: <a href="#">ENUM_VIDEOCODEC_H264</a> , <a href="#">ENUM_VIDEOCODEC_VP8</a> , <a href="#">ENUM_VIDEOCODEC_VP9</a> .
<i>sdpParameter</i>	The parameter is in string format.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

Example:

```
setVideoCodecParameter(PortSipEnumDefine.ENUM_VIDEOCODEC_H264, "profile-level-id=420033; packetization-mode=0");
```

## Additional settings functions

### Functions

- String [com.portsip.PortSipSdk.getVersion](#) ()
- int [com.portsip.PortSipSdk.enableRport](#) (boolean enable)
- int [com.portsip.PortSipSdk.enableEarlyMedia](#) (boolean enable)  
*Enable/disable rport(RFC3581).*
- int [com.portsip.PortSipSdk.enableReliableProvisional](#) (boolean enable)
- int [com.portsip.PortSipSdk.enable3GppTags](#) (boolean enable)
- void [com.portsip.PortSipSdk.enableCallbackSignaling](#) (boolean enableSending, boolean enableReceived)
- void [com.portsip.PortSipSdk.setSrtpPolicy](#) (int enum\_srtpolicy)
- int [com.portsip.PortSipSdk.setRtpPortRange](#) (int minimumRtpAudioPort, int maximumRtpAudioPort, int minimumRtpVideoPort, int maximumRtpVideoPort)
- int [com.portsip.PortSipSdk.setRtcpPortRange](#) (int minimumRtcpAudioPort, int maximumRtcpAudioPort, int minimumRtcpVideoPort, int maximumRtcpVideoPort)
- int [com.portsip.PortSipSdk.enableCallForward](#) (boolean forBusyOnly, String forwardTo)
- int [com.portsip.PortSipSdk.disableCallForward](#) ()
- int [com.portsip.PortSipSdk.enableSessionTimer](#) (int timerSeconds)
- void [com.portsip.PortSipSdk.disableSessionTimer](#) ()
- void [com.portsip.PortSipSdk.setDoNotDisturb](#) (boolean state)
- void [com.portsip.PortSipSdk.enableAutoCheckMwi](#) (boolean state)
- int [com.portsip.PortSipSdk.setRtpKeepAlive](#) (boolean state, int keepAlivePayloadType, int deltaTransmitTimeMS)
- int [com.portsip.PortSipSdk.setKeepAliveTime](#) (int keepAliveTime)
- int [com.portsip.PortSipSdk.setAudioSamples](#) (int ptime, int maxptime)
- int [com.portsip.PortSipSdk.addSupportedMimeType](#) (String methodName, String mimeType, String subMimeType)

---

## Detailed Description

---

### Function Documentation

#### String [com.portsip.PortSipSdk.getVersion](#) ()

Get the version number of the current SDK.

##### Returns:

String with version description

#### int [com.portsip.PortSipSdk.enableRport](#) (boolean *enable*)

Enable/Disable rport(RFC3581).

##### Parameters:

<i>enable</i>	enable Set to true to enable the SDK to support rport. By default it is enabled.
---------------	--

##### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

#### int [com.portsip.PortSipSdk.enableEarlyMedia](#) (boolean *enable*)

Enable/disable rport(RFC3581).

**Parameters:**

<i>enable</i>	Set to true to enable the SDK to support rport. By default it is enabled.
---------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code. Enable/Disable Early Media.

**Parameters:**

<i>enable</i>	Set to true to enable the SDK support Early Media. By default the Early Media is disabled.
---------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.enableReliableProvisional (boolean *enable*)**

Enable/Disable PRACK.

**Parameters:**

<i>enable</i>	Set to true to enable the SDK support PRACK. In default the PRACK is disabled.
---------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.enable3GppTags (boolean *enable*)**

Enable/disable the 3Gpp tags, including "ims.icsi.mmtel" and "g.3gpp.smsip".

**Parameters:**

<i>enable</i>	Set to true to enable 3Gpp tags for SDK.
---------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.enableCallbackSignaling (boolean *enableSending*, boolean *enableReceived*)**

Enable/disable the callback of the SIP messages.

**Parameters:**

<i>enableSending</i>	Set as true to enable to callback the sent SIP messages, or false to disable. Once enabled, the "onSendingSignaling" event will be triggered when the SDK sends a SIP message.
<i>enableReceived</i>	Set as true to enable to callback the received SIP messages, or false to disable. Once enabled, the "onReceivedSignaling" event will be triggered when the SDK receives a SIP message.

**void com.portsip.PortSipSdk.setSrtpPolicy (int *enum\_srtpolicy*)**

Set the SRTP policy.

**Parameters:**

<i>enum_srtpolicy</i>	The SRTP policy.allow: <a href="#">ENUM_SRTPPOLICY_NONE</a> , <a href="#">ENUM_SRTPPOLICY_FORCE</a> , <a href="#">ENUM_SRTPPOLICY_PREFER</a> .
-----------------------	--

**int com.portsip.PortSipSdk.setRtpPortRange (int *minimumRtpAudioPort*, int *maximumRtpAudioPort*, int *minimumRtpVideoPort*, int *maximumRtpVideoPort*)**

This function allows to set the RTP port range for audio and video streaming.

**Parameters:**

<i>minimumRtpAudioPort</i>	The minimum RTP port for audio stream.
<i>maximumRtpAudioPort</i>	The maximum RTP port for audio stream.
<i>minimumRtpVideoPort</i>	The minimum RTP port for video stream.
<i>maximumRtpVideoPort</i>	The maximum RTP port for video stream.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

The port range ((max - min) % maxCallLines) should be greater than 4.

**int com.portsip.PortSipSdk.setRtcpPortRange (int *minimumRtcpAudioPort*, int *maximumRtcpAudioPort*, int *minimumRtcpVideoPort*, int *maximumRtcpVideoPort*)**

This function allows to set the RTCP port range for audio and video streaming.

**Parameters:**

<i>minimumRtcpAudioPort</i>	The minimum RTCP port for audio stream.
<i>maximumRtcpAudioPort</i>	The maximum RTCP port for audio stream.
<i>minimumRtcpVideoPort</i>	The minimum RTCP port for video stream.
<i>maximumRtcpVideoPort</i>	The maximum RTCP port for video stream.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

The port range ((max - min) % maxCallLines) should be greater than 4.

**int com.portsip.PortSipSdk.enableCallForward (boolean *forBusyOnly*, String *forwardTo*)**

Enable call forwarding.

**Parameters:**

<i>forBusyOnly</i>	If this parameter is set to true, the SDK will forward incoming calls when the user is currently busy. If set it to false, SDK will forward all incoming calls.
<i>forwardTo</i>	The target to which the call will be forwarded. It must be in the format of sip: <a href="mailto:xxx@sip.portsip.com">xxx@sip.portsip.com</a> .

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.disableCallForward ()**

Disable the call forwarding. The SDK will not forward any incoming call when this function is called.

**Returns:**

If the function succeeds, it will not return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.enableSessionTimer (int *timerSeconds*)**

This function allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending repeated INVITE requests.

**Parameters:**

<i>timerSeconds</i>	The value of the refresh interval in seconds. A minimum of 90 seconds required.
---------------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

The repeated INVITE requests, or re-INVITES, are sent during an active call log to allow user agents (UA) or proxies to determine the status of a SIP session. Without this keep-alive mechanism, proxies that remember incoming and outgoing requests (stateful proxies) may continue to retain call state in vain. If a UA fails to send a BYE message at the end of a session, or if the BYE message is lost due to network problems, a stateful proxy will not know that the session has ended. The re-INVITES ensure that active sessions stay active and completed sessions are terminated.

**void com.portsip.PortSipSdk.disableSessionTimer ()**

Disable the session timer.

**void com.portsip.PortSipSdk.setDoNotDisturb (boolean *state*)**

Enable/disable the "Do not disturb" status.

**Parameters:**

<i>state</i>	If it is set to true, the SDK will reject all incoming calls.
--------------	---

**void com.portsip.PortSipSdk.enableAutoCheckMwi (boolean *state*)**

Enable/disable the "Auto Check MWI" status.

**Parameters:**

<i>state</i>	If it is set to true, the SDK will check Mwi automatically.
--------------	---

**int com.portsip.PortSipSdk.setRtpKeepAlive (boolean *state*, int *keepAlivePayloadType*, int *deltaTransmitTimeMS*)**

Enable or disable to send RTP keep-alive packet when the call is ongoing.

**Parameters:**

<i>state</i>	When it's set to true, it's allowed to send the keep-alive packet during the conversation;
<i>keepAlivePayloadType</i>	The payload type of the keep-alive RTP packet. It's usually set to 126.
<i>deltaTransmitTimeMS</i>	The interval for sending keep-alive RTP packet, in millisecond. Recommended value ranges 15000 - 300000.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setKeepAliveTime (int *keepAliveTime*)**

Enable or disable to send SIP keep-alive packet.

**Parameters:**

<i>keepAliveTime</i>	This is the time interval for SIP keep-alive, in seconds. When it is set to 0, the SIP keep-alive will be disabled. Recommended value is 30 or 50.
----------------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setAudioSamples (int *ptime*, int *maxptime*)**

Set the audio capture sample, which will be present in the SDP of INVITE and 200 OK message as "ptime and "maxptime" attribute.

**Parameters:**

<i>ptime</i>	It should be a multiple of 10 between 10 - 60 (included 10 and 60).
<i>maxptime</i>	The "maxptime" attribute should be a multiple of 10 between 10 - 60 (included 10 and 60). It can't be less than "ptime".

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.addSupportedMimeType (String *methodName*, String *mimeType*, String *subMimeType*)**

Set the SDK to receive SIP messages that include special mime type.

**Parameters:**

<i>methodName</i>	Method name of the SIP message, such as INVITE, OPTION, INFO, MESSAGE, UPDATE, ACK etc. For more details please refer to RFC3261.
<i>mimeType</i>	The mime type of SIP message.
<i>subMimeType</i>	The sub mime type of SIP message.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

In default, PortSIP VoIP SDK supports media types (mime types) included in the below incoming SIP messages:

```
"message/sipfrag" in NOTIFY message.
"application/simple-message-summary" in NOTIFY message.
"text/plain" in MESSAGE message. "application/dtmf-relay" in INFO
message. <br> "application/media_control+xml" in INFO message.
```

The SDK allows to receive SIP messages that include above mime types. Now if remote side send an INFO SIP message with its "Content-Type" header value "text/plain", SDK will reject this INFO message, because "text/plain" of INFO message is not included in the default type list. How should we enable the SDK to receive SIP INFO messages that include "text/plain" mime type? The answer is addSupportedMimyType:

```
addSupportedMimeType("INFO", "text", "plain");
```

If the user wishes to receive the NOTIFY message with "application/media\_control+xml", it should be set as below:

```
addSupportedMimeType("NOTIFY", "application", "media_control+xml");
```

For more details about the mime type, please visit:  
<http://www.iana.org/assignments/media-types/>

## Access SIP message header functions

### Functions

- String [com.portsip.PortSipSdk.getSipMessageHeaderValue](#) (String sipMessage, String headerName)
- long [com.portsip.PortSipSdk.addSipMessageHeader](#) (long sessionId, String methodName, int msgType, String headerName, String headerValue)
- int [com.portsip.PortSipSdk.removeAddedSipMessageHeader](#) (long addedSipMessageId)
- void [com.portsip.PortSipSdk.clearAddedSipMessageHeaders](#) ()
- long [com.portsip.PortSipSdk.modifySipMessageHeader](#) (long sessionId, String methodName, int msgType, String headerName, String headerValue)
- int [com.portsip.PortSipSdk.removeModifiedSipMessageHeader](#) (long modifiedSipMessageId)
- void [com.portsip.PortSipSdk.clearModifiedSipMessageHeaders](#) ()

---

### Detailed Description

---

### Function Documentation

#### **String com.portsip.PortSipSdk.getSipMessageHeaderValue (String sipMessage, String headerName)**

Access the SIP header of SIP message.

##### Parameters:

<i>sipMessage</i>	The SIP message.
<i>headerName</i>	The header of which user wishes to access the SIP message.

##### Returns:

String. The SIP header of SIP message.

#### **long com.portsip.PortSipSdk.addSipMessageHeader (long sessionId, String methodName, int msgType, String headerName, String headerValue)**

Add the SIP Message header into the specified outgoing SIP message.

##### Parameters:

<i>sessionId</i>	Add the header to the SIP message with the specified session Id only. By setting to -1, it will be added to all messages.
<i>methodName</i>	Add the header to the SIP message with specified method name only. For example: "INVITE", "REGISTER", "INFO" etc. If "ALL" specified, it will add all SIP messages.
<i>msgType</i>	1 refers to apply to the request message, 2 refers to apply to the response message, 3 refers to apply to both request and response.
<i>headerName</i>	The header name which will appear in SIP message.
<i>headerValue</i>	The custom header value.

##### Returns:

If the function succeeds, it will return the addedSipMessageId , which is greater than 0. If the function fails, it will return a specific error code.

#### **int com.portsip.PortSipSdk.removeAddedSipMessageHeader (long addedSipMessageId)**

Remove the headers (custom header) added by addSipMessageHeader.

**Parameters:**

<i>addedSipMessageId</i>	The addedSipMessageId return by addSipMessageHeader.
--------------------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.clearAddedSipMessageHeaders ()**

Clear the added extension headers (custom headers)

**Remarks:**

For example, we have added two custom headers into every outgoing SIP message and want to have them removed.

```
addSipMessageHeader(-1,"ALL",3,"Billing", "usd100.00");
addSipMessageHeader(-1,"ALL",3,"ServiceId", "8873456");
clearAddedSipMessageHeaders();
```

If this function is called, the added extension headers will no longer appear in outgoing SIP message.

**long com.portsip.PortSipSdk.modifySipMessageHeader (long sessionId, String methodName, int msgType, String headerName, String headerValue)**

Modify the special SIP header value for every outgoing SIP message.

**Parameters:**

<i>sessionId</i>	The header to the SIP message with the specified session Id. By setting to -1, it will be added to all messages.
<i>methodName</i>	Modify the header to the SIP message with specified method name only. For example: "INVITE", "REGISTER", "INFO" etc. If "ALL" specified, it will add all SIP messages.
<i>msgType</i>	1 refers to apply to the request message, 2 refers to apply to the response message, 3 refers to apply to both request and response.
<i>headerName</i>	The SIP header name of which the value will be modified.
<i>headerValue</i>	The header value to be modified.

**Returns:**

If the function succeeds, it will return modifiedSipMessageId, which is greater than 0. If the function fails, it will return a specific error code.

**Remarks:**

Example: modify "Expires" header and "User-Agent" header value for every outgoing SIP message:

```
modifySipMessageHeader(-1,"ALL",3, "Expires", "1000");
modifySipMessageHeader(-1,"ALL",3, "User-Agent", "MyTest Softphone 1.0");
```

**int com.portsip.PortSipSdk.removeModifiedSipMessageHeader (long modifiedSipMessageId)**

Remove the headers (custom header) added by modifiedSipMessageId.

**Parameters:**

<i>modifiedSipMessageId</i>	The modifiedSipMessageId return by modifySipMessageHeader.
-----------------------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.clearModifiedSipMessageHeaders ()**

Clear the modify headers value. Once cleared, it will no longer modify every outgoing SIP message header values.

## Remarks:

Example: modify two headers value for every outgoing SIP message and then clear it:

```
modifySipMessageHeader(-1,"ALL",3, "Expires", "1000");
modifySipMessageHeader(-1,"ALL",3, "User-Agent", "MyTest Softphone 1.0");
clearModifyHeaders();
```

## Audio and video functions

### Functions

- int [com.portsip.PortSipSdk.setVideoDeviceId](#) (int deviceId)
- int [com.portsip.PortSipSdk.enableVideoHardwareCodec](#) (boolean enableHWEncoder, boolean enableHWDDecoder)
- int [com.portsip.PortSipSdk.setVideoResolution](#) (int width, int height)
- int [com.portsip.PortSipSdk.setVideoCropAndScale](#) (boolean enable)
- int [com.portsip.PortSipSdk.setAudioBitrate](#) (long sessionId, int enum\_audiocodec, int bitrateKbps)
- int [com.portsip.PortSipSdk.setVideoBitrate](#) (long sessionId, int bitrateKbps)
- int [com.portsip.PortSipSdk.setVideoFrameRate](#) (long sessionId, int frameRate)
- int [com.portsip.PortSipSdk.sendVideo](#) (long sessionId, boolean send)
- void [com.portsip.PortSipSdk.setLocalVideoWindow](#) ([PortSIPVideoRenderer](#) renderer)
- int [com.portsip.PortSipSdk.setRemoteVideoWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- void [com.portsip.PortSipSdk.displayLocalVideo](#) (boolean state)
- int [com.portsip.PortSipSdk.setVideoNackStatus](#) (boolean state)
- int [com.portsip.PortSipSdk.setChannelOutputVolumeScaling](#) (long sessionId, int scaling)
- int [com.portsip.PortSipSdk.setChannelInputVolumeScaling](#) (long sessionId, int scaling)
- Set< PortSipEnumDefine.AudioDevice > [com.portsip.PortSipSdk.getAudioDevices](#) ()
- int [com.portsip.PortSipSdk.setAudioDevice](#) (PortSipEnumDefine.AudioDevice defaultDevice)

---

## Detailed Description

---

### Function Documentation

#### int com.portsip.PortSipSdk.setVideoDeviceId (int *deviceId*)

Set the video device that will be used for video call.

##### Parameters:

<i>deviceId</i>	Device ID (index) for video device (camera).
-----------------	--

##### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

#### int com.portsip.PortSipSdk.enableVideoHardwareCodec (boolean *enableHWEncoder*, boolean *enableHWDDecoder*)

Set enable/disable video Hardware codec.

##### Parameters:

<i>enableHWEncoder</i>	If it is set to true, the SDK will use video hardware encoder when available. By default it is true.
<i>enableHWDDecoder</i>	If it is set to true, the SDK will use video hardware decoder when available. By default it is true.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoResolution (int *width*, int *height*)**

Set the video capturing resolution.

**Parameters:**

<i>width</i>	Video resolution, width
<i>height</i>	Video resolution, height

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoCropAndScale (boolean *enable*)**

When the camera does not support specified resolution, enable or disable SDK to crop and scale to the specified resolution.

**Parameters:**

<i>enable</i>	Enable or disable to crop or scale the video to fit in specified resolution. By default it is disabled.
---------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setAudioBitrate (long *sessionId*, int *enum\_audiocodec*, int *bitrateKbps*)**

Set the audio bitrate.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>enum_audiocodec</i>	Audio codec type allowed: <a href="#">ENUM_AUDIOCODEC_OPUS</a>
<i>bitrateKbps</i>	The Audio bitrate in KBPS.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoBitrate (long *sessionId*, int *bitrateKbps*)**

Set the video bitrate.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>bitrateKbps</i>	The video bitrate in KBPS.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoFrameRate (long *sessionId*, int *frameRate*)**

Set the video frame rate. Usually you do not need to call this function to set the frame rate since the SDK uses default frame rate.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>frameRate</i>	The frame rate value, with its minimum of 5, and maximum value of 30. The greater the value is, the better video quality enabled and more bandwidth required;

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.sendVideo (long *sessionId*, boolean *send*)**

Send the video to remote side.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>send</i>	Set to true to send the video, or false to stop sending.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.setLocalVideoWindow ([PortSIPVideoRenderer](#) *renderer*)**

Set the window that is used for displaying the local video image.

**Parameters:**

<i>renderer</i>	<a href="#">SurfaceView</a> a SurfaceView for displaying local video image from camera.
-----------------	---

**int com.portsip.PortSipSdk.setRemoteVideoWindow (long *sessionId*, [PortSIPVideoRenderer](#) *renderer*)**

Set the window for a session that is used for displaying the received remote video image.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>renderer</i>	<a href="#">SurfaceView</a> a SurfaceView for displaying the received remote video image.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.displayLocalVideo (boolean *state*)**

Start/stop displaying the local video image.

**Parameters:**

<i>state</i>	Set to true to display local video image.
--------------	---

**int com.portsip.PortSipSdk.setVideoNackStatus (boolean *state*)**

Enable/disable the NACK feature (rfc6642) which helps to improve the video quality.

**Parameters:**

<i>state</i>	Set to true to enable.
--------------	------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setChannelOutputVolumeScaling (long *sessionId*, int *scaling*)**

Set a volume |scaling| to be applied to the outgoing signal of a specific audio channel.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>scaling</i>	Valid scale ranges [0, 1000]. Default is 100.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setChannellInputVolumeScaling (long *sessionId*, int *scaling*)**

Set a volume |scaling| to be applied to the microphone signal of a specific audio channel.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>scaling</i>	Valid scale ranges [0, 1000]. Default is 100.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Set<PortSipEnumDefine.AudioDevice> com.portsip.PortSipSdk.getAudioDevices ()**

Get current set of available/selectable audio devices.

**Returns:**

Current set of available/selectable audio devices.

**int com.portsip.PortSipSdk.setAudioDevice (PortSipEnumDefine.AudioDevice *defaultDevice*)**

Set the audio device that will used for audio call. For Android and iOS, switch between earphone and Loudspeaker allowed.

**Parameters:**

<i>defaultDevice</i>	Set to true the SDK use loudspeaker for audio call, this just available for mobile platform only.
----------------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Call functions

### Functions

- long [com.portsip.PortSipSdk.call](#) (String callee, boolean sendSdp, boolean videoCall)
- int [com.portsip.PortSipSdk.rejectCall](#) (long sessionId, int code)
- int [com.portsip.PortSipSdk.hangUp](#) (long sessionId)
- int [com.portsip.PortSipSdk.answerCall](#) (long sessionId, boolean videoCall)
- int [com.portsip.PortSipSdk.updateCall](#) (long sessionId, boolean enableAudio, boolean enableVideo)
- int [com.portsip.PortSipSdk.hold](#) (long sessionId)
- int [com.portsip.PortSipSdk.unHold](#) (long sessionId)
- int [com.portsip.PortSipSdk.muteSession](#) (long sessionId, boolean muteIncomingAudio, boolean muteOutgoingAudio, boolean muteIncomingVideo, boolean muteOutgoingVideo)
- int [com.portsip.PortSipSdk.forwardCall](#) (long sessionId, String forwardTo)
- long [com.portsip.PortSipSdk.pickupBLFCall](#) (String replaceDialogId, boolean videoCall)
- int [com.portsip.PortSipSdk.sendDtmf](#) (long sessionId, int enum\_dtmfMethod, int code, int dtmfDuration, boolean playDtmfTone)

## Detailed Description

---

### Function Documentation

**long com.portsip.PortSipSdk.call (String *callee*, boolean *sendSdp*, boolean *videoCall*)**

Make a call

**Parameters:**

<i>callee</i>	The callee. It can be a name only or full SIP URI, for example: user001 or sip: <a href="mailto:user001@sip.iptel.org">user001@sip.iptel.org</a> or sip: <a href="mailto:user002@sip.yourdomain.com">user002@sip.yourdomain.com</a> :5068
<i>sendSdp</i>	If it is set to false, the outgoing call will not include the SDP in INVITE message.
<i>videoCall</i>	If it is set to true and at least one video codec was added, the outgoing call will include the video codec into SDP. Otherwise no video codec will be added into outgoing SDP.

**Returns:**

If the function succeeds, it will return the session ID of the call, which is greater than 0. If the function fails, it will return a specific error code.

Note: the function success just means the outgoing call is processing, you need to detect the call final state in onInviteTrying, onInviteRinging, onInviteFailure callback events.

**int com.portsip.PortSipSdk.rejectCall (long *sessionId*, int *code*)**

rejectCall Reject the incoming call.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>code</i>	Reject code, for example, 486, 480 etc.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.hangUp (long *sessionId*)**

hangUp Hang up the call.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.answerCall (long *sessionId*, boolean *videoCall*)**

answerCall Answer the incoming call.

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>videoCall</i>	If the incoming call is a video call and the video codec is matched, set to true to answer the video call. If set to false, the answer call does not include video codec answer anyway.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.updateCall (long *sessionId*, boolean *enableAudio*, boolean *enableVideo*)**

updateCall Use the re-INVITE to update the established call.

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>enableAudio</i>	Set to true to allow the audio in updated call, or false to disable audio in updated call.
<i>enableVideo</i>	Set to true to allow the video in update call, or false to disable video in updated call.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return specific error code.

**Remarks:**

Example usage:

Example 1: A called B with the audio only, and B answered A, there would be an audio conversation between A and B. Now A want to see B through video, A could use these functions to fulfill it.

```
clearVideoCodec();
addVideoCodec(VIDEOCODEC_H264);
updateCall(sessionId, true, true);
```

Example 2: Remove video stream from the current conversation.

```
updateCall(sessionId, true, false);
```

**int com.portsip.PortSipSdk.hold (long *sessionId*)**

To place a call on hold.

**Parameters:**

<i>sessionId</i>	The session ID of call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.unHold (long *sessionId*)**

Take off hold.

**Parameters:**

<i>sessionId</i>	The session ID of call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.muteSession (long *sessionId*, boolean *muteIncomingAudio*, boolean *muteOutgoingAudio*, boolean *muteIncomingVideo*, boolean *muteOutgoingVideo*)**

Mute the specified audio or video session.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>muteIncomingAudio</i>	Set it to true to mute incoming audio stream. Once set, remote side audio cannot be heard.
<i>muteOutgoingAudio</i>	Set it to true to mute outgoing audio stream. Once set, the remote side cannot hear the audio.
<i>muteIncomingVideo</i>	Set it to true to mute incoming video stream. Once set, remote side video

<i>o</i>	cannot be seen.
<i>muteOutgoingVide</i>	Set it to true to mute outgoing video stream, the remote side cannot see the video.
<i>o</i>	

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.forwardCall (long *sessionId*, String *forwardTo*)**

Forward call to another one when receiving the incoming call.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>forwardTo</i>	Target of the forward. It can be either "sip:number@sipserver.com" or "number".

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return value a specific error code.

**long com.portsip.PortSipSdk.pickupBLFCall (String *replaceDialogId*, boolean *videoCall*)**

This function will be used for picking up a call based on the BLF (Busy Lamp Field) status.

**Parameters:**

<i>replaceDialogId</i>	The ID of the call which will be pickup. It comes with onDialogStateUpdated callback.
<i>videoCall</i>	Indicates pickup video call or audio call

**Returns:**

If the function succeeds, it will return a session ID that is greater than 0 to the new call, otherwise returns a specific error code that is less than 0.

**Remarks:**

The scenario is:

1. User 101 subscribed the user 100's call status: sendSubscription(mSipLib, "100", "dialog");
2. When 100 holds a call or 100 is ringing, onDialogStateUpdated callback will be triggered, and 101 will receive this callback. Now 101 can use pickupBLFCall function to pick the call rather than 100 to talk with caller.

**int com.portsip.PortSipSdk.sendDtmf (long *sessionId*, int *enum\_dtmfMethod*, int *code*, int *dtmfDuration*, boolean *playDtmfTone*)**

Send DTMF tone.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>enum_dtmfMethod</i>	DTMF tone could be sent via two methods: DTMF_RFC2833 or DTMF_INFO. The DTMF_RFC2833 is recommend.
<i>code</i>	The DTMF tone. Values include:

code	Description
0	The DTMF tone 0.
1	The DTMF tone 1.
2	The DTMF tone 2.
3	The DTMF tone 3.
4	The DTMF tone 4.
5	The DTMF tone 5.

6	The DTMF tone 6.
7	The DTMF tone 7.
8	The DTMF tone 8.
9	The DTMF tone 9.
10	The DTMF tone *.
11	The DTMF tone #.
12	The DTMF tone A.
13	The DTMF tone B.
14	The DTMF tone C.
15	The DTMF tone D.
16	The DTMF tone FLASH.

**Parameters:**

<i>dtmfDuration</i>	The DTMF tone samples. Recommended value 160.
<i>playDtmfTone</i>	Set to true the SDK play local DTMF tone sound during send DTMF.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Refer functions

### Functions

- int [com.portsip.PortSipSdk.refer](#) (long sessionId, String referTo)
- int [com.portsip.PortSipSdk.attendedRefer](#) (long sessionId, long replaceSessionId, String referTo)
- int [com.portsip.PortSipSdk.attendedRefer2](#) (long sessionId, long replaceSessionId, String replaceMethod, String target, String referTo)
- int [com.portsip.PortSipSdk.outOfDialogRefer](#) (long replaceSessionId, String replaceMethod, String target, String referTo)
- long [com.portsip.PortSipSdk.acceptRefer](#) (long referId, String referSignaling)
- int [com.portsip.PortSipSdk.rejectRefer](#) (long referId)

## Detailed Description

### Function Documentation

**int com.portsip.PortSipSdk.refer (long sessionId, String referTo)**

Transfer the current call to another callee.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>referTo</i>	Target callee of the transfer. It can be either "sip:number@sipserver.com" or "number".

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

```
refer(sessionId, "sip:testuser12@sip.portsip.com");
```

You can refer to the video on Youtube at:

<https://www.youtube.com/watch?v=2w9EGgr3FY>, which will demonstrate how to complete the transfer.

### **int com.portsip.PortSipSdk.attendedRefer (long *sessionId*, long *replaceSessionId*, String *referTo*)**

Make an attended refer.

#### **Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>replaceSessionId</i>	Session ID of the replace call.
<i>referTo</i>	Target callee of the refer. It can be either "sip:number@sipserver.com" or "number".

#### **Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

#### **Remarks:**

Please read the sample project source code to get more details, or you can refer to the video on YouTube at:

<https://www.youtube.com/watch?v=2w9EGgr3FY>

Note: Please use Windows Media Player to play the AVI file, which demonstrates how to complete the transfer.

### **int com.portsip.PortSipSdk.attendedRefer2 (long *sessionId*, long *replaceSessionId*, String *replaceMethod*, String *target*, String *referTo*)**

Make an attended refer.

#### **Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>replaceSessionId</i>	The session ID of the session to be replaced.
<i>replaceMethod</i>	The SIP method name to be added in the "Refer-To" header, usually INVITE or BYE.
<i>target</i>	The target to which the REFER message will be sent.
<i>referTo</i>	The URI to be added into the "Refer-To" header.

#### **Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

### **int com.portsip.PortSipSdk.outOfDialogRefer (long *replaceSessionId*, String *replaceMethod*, String *target*, String *referTo*)**

Make an attended refer.

#### **Parameters:**

<i>replaceSessionId</i>	The session ID of the session which will be replaced.
<i>replaceMethod</i>	The SIP method name which will be added in the "Refer-To" header, usually INVITE or BYE.
<i>target</i>	The target to which the REFER message will be sent.
<i>referTo</i>	The URI which will be added into the "Refer-To" header.

#### **Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

### **long com.portsip.PortSipSdk.acceptRefer (long *referId*, String *referSignaling*)**

By accepting the REFER request, a new call will be made if this function is called. The function is usually called after onReceivedRefer callback event.

**Parameters:**

<i>referId</i>	The ID of REFER request that comes from onReceivedRefer callback event.
<i>referSignaling</i>	The SIP message of REFER request that comes from onReceivedRefer callback event.

**Returns:**

If the function succeeds, it will return a session ID greater than 0 to the new call for REFER; otherwise it will return a specific error code less than 0;

**int com.portsip.PortSipSdk.rejectRefer (long *referId*)**

Reject the REFER request.

**Parameters:**

<i>referId</i>	The ID of REFER request that comes from onReceivedRefer callback event.
----------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Send audio and video stream functions

### Functions

- int [com.portsip.PortSipSdk.enableSendPcmStreamToRemote](#) (long sessionId, boolean state, int streamSamplesPerSec)
- int [com.portsip.PortSipSdk.sendPcmStreamToRemote](#) (long sessionId, byte[] data, int dataLength)
- int [com.portsip.PortSipSdk.enableSendVideoStreamToRemote](#) (long sessionId, boolean state)
- int [com.portsip.PortSipSdk.sendVideoStreamToRemote](#) (long sessionId, byte[] data, int dataLength, int width, int height)

---

## Detailed Description

---

### Function Documentation

**int com.portsip.PortSipSdk.enableSendPcmStreamToRemote (long *sessionId*, boolean *state*, int *streamSamplesPerSec*)**

Enable the SDK send PCM stream data to remote side from another source instead of microphone. This function MUST be called first to send the PCM stream data to another side.

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>state</i>	Set to true to enable the send stream, or false to disable.
<i>streamSamplesPerSec</i>	The PCM stream data sample, in seconds. For example 8000 or 16000.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.sendPcmStreamToRemote (long *sessionId*, byte [] *data*, int *dataLength*)**

Send the audio stream in PCM format from another source instead of audio device capturing (microphone).

**Parameters:**

<i>sessionId</i>	Session ID of the call conversation.
<i>data</i>	The PCM audio stream data. It must be 16bit, mono.
<i>dataLength</i>	The size of data.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

Usually we should use it like below:

```
enableSendPcmStreamToRemote(sessionId, true, 16000);
sendPcmStreamToRemote(sessionId, data, dataSize);
```

You can't have too much audio data at one time as we have 100ms audio buffer only. Once you put too much, data will be lost. It is recommended to send 20ms audio data every 20ms.

**int com.portsip.PortSipSdk.enableSendVideoStreamToRemote (long *sessionId*, boolean *state*)**

Enable the SDK to send video stream data to remote side from another source instead of camera.

This function MUST be called first to send the video stream data to another side.

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>state</i>	Set to true to enable the send stream, or false to disable.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.sendVideoStreamToRemote (long *sessionId*, byte [] *data*, int *dataLength*, int *width*, int *height*)**

Send the video stream in i420 from another source instead of video device capturing (camera).

Before calling this function, you MUST call the enableSendVideoStreamToRemote function.

**Parameters:**

<i>sessionId</i>	Session ID of the call conversation.
<i>data</i>	The video stream data. It must be in i420 format.
<i>dataLength</i>	The size of data.
<i>width</i>	The width of the video image.
<i>height</i>	The height of video image.

**Returns:**

If the function succeeds, it will return value is 0. If the function fails, it will return a specific error code.

# RTP packets, Audio stream and video stream callback

## Functions

- void [com.portsip.PortSipSdk.setRtpCallback](#) (boolean enable)
- void [com.portsip.PortSipSdk.enableAudioStreamCallback](#) (long sessionId, boolean enable, int enum\_audioCallbackMode)
- void [com.portsip.PortSipSdk.enableVideoStreamCallback](#) (long sessionId, int enum\_videoCallbackMode)

---

## Detailed Description

functions

---

## Function Documentation

### void com.portsip.PortSipSdk.setRtpCallback (boolean *enable*)

Set the RTP callbacks to allow access to the sent and received RTP packets.

#### Parameters:

<i>enable</i>	Set to true to enable the RTP callback for receiving and sending RTP packets. The onSendingRtpPacket and onReceivedRtpPacket events will be triggered.
---------------	--

### void com.portsip.PortSipSdk.enableAudioStreamCallback (long *sessionId*, boolean *enable*, int *enum\_audioCallbackMode*)

Enable/disable the audio stream callback. The onAudioRawCallback event will be triggered if the callback is enabled.

#### Parameters:

<i>sessionId</i>	The session ID of call.
<i>enable</i>	Set to true to enable audio stream callback, or false to stop the callback.
<i>enum_audioCallbackMode</i>	The audio stream callback mode. Supported modes include <a href="#">ENUM_AUDIOSTREAM_NONE</a> , <a href="#">ENUM_AUDIOSTREAM_LOCAL_PER_CHANNEL</a> , <a href="#">ENUM_AUDIOSTREAM_REMOTE_PER_CHANNEL</a> , <a href="#">ENUM_AUDIOSTREAM_BOTH_PER_CHANNEL</a> .

### void com.portsip.PortSipSdk.enableVideoStreamCallback (long *sessionId*, int *enum\_videoCallbackMode*)

Enable/disable the video stream callback, the onVideoRawCallback event will be triggered if the callback is enabled.

#### Parameters:

<i>sessionId</i>	The session ID of call.
<i>enum_videoCallbackMode</i>	The video stream callback mode. Supported modes include <a href="#">ENUM_VIDEOSTREAM_NONE</a> , <a href="#">ENUM_VIDEOSTREAM_LOCAL</a> , <a href="#">ENUM_VIDEOSTREAM_REMOTE</a> , <a href="#">ENUM_VIDEOSTREAM_BOTH</a> .

## Record functions

### Functions

- int [com.portsip.PortSipSdk.startRecord](#) (long sessionId, String recordFilePath, String recordFileName, boolean appendTimeStamp, int enum\_audioFileFormat, int enum\_audioRecordMode, int enum\_videocodec, int enum\_videoRecordMode)
- int [com.portsip.PortSipSdk.stopRecord](#) (long sessionId)

---

### Detailed Description

---

### Function Documentation

**int com.portsip.PortSipSdk.startRecord (long sessionId, String recordFilePath, String recordFileName, boolean appendTimeStamp, int enum\_audioFileFormat, int enum\_audioRecordMode, int enum\_videocodec, int enum\_videoRecordMode)**

Start recording the call.

#### Parameters:

<i>sessionId</i>	The session ID of call conversation.
<i>recordFilePath</i>	The file path to save record file. It must be existent.
<i>recordFileName</i>	The file name of record file. For example audiorecord.wav or videorecord.avi.
<i>appendTimeStamp</i>	Set to true to append the timestamp to the name of the recording file.
<i>enum_audioFileFormat</i>	The audio record file format, allow below values: FILEFORMAT_WAVE = 1, /// The record audio file is WAVE format. FILEFORMAT_AMR, /// The record audio file is in AMR format with all voice data compressed by AMR codec.
<i>enum_audioRecordMode</i>	The audio record mode, allow below values: RECORD_NONE = 0, /// Not Record. RECORD_RECV = 1, /// Only record the received data. RECORD_SEND, /// Only record send data. RECORD_BOTH /// The record audio file is WAVE format.
<i>enum_videocodec</i>	The codec used for compressing the video data to save into video record file.
<i>enum_videoRecordMode</i>	Allow to set video record mode. Support to record received and/or sent video.

#### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.stopRecord (long sessionId)**

Stop recording.

#### Parameters:

<i>sessionId</i>	The session ID of call conversation.
------------------	--------------------------------------

#### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Play audio and video file to remote functions

### Functions

- int [com.portsip.PortSipSdk.playVideoFileToRemote](#) (long sessionId, String aviFile, boolean loop, boolean playAudio)
- int [com.portsip.PortSipSdk.stopPlayVideoFileToRemote](#) (long sessionId)
- int [com.portsip.PortSipSdk.playAudioFileToRemote](#) (long sessionId, String filename, int fileSamplesPerSec, boolean loop)
- int [com.portsip.PortSipSdk.stopPlayAudioFileToRemote](#) (long sessionId)
- int [com.portsip.PortSipSdk.playAudioFileToRemoteAsBackground](#) (long sessionId, String filename, int fileSamplesPerSec)
- int [com.portsip.PortSipSdk.stopPlayAudioFileToRemoteAsBackground](#) (long sessionId)
- void [com.portsip.PortSipSdk.audioPlayLoopbackTest](#) (boolean enable)

---

### Detailed Description

---

### Function Documentation

**int com.portsip.PortSipSdk.playVideoFileToRemote (long sessionId, String aviFile, boolean loop, boolean playAudio)**

Play an AVI file to remote party.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
<i>aviFile</i>	The full filepath, such as "/mnt/sdcard/test.avi".
<i>loop</i>	Set to false to stop playing video file when it is ended, or true to play it repeatedly.
<i>playAudio</i>	If set to true, audio and video will be played together; or false to play the video only.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.stopPlayVideoFileToRemote (long sessionId)**

Stop play video file to remote side.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.playAudioFileToRemote (long sessionId, String filename, int fileSamplesPerSec, boolean loop)**

Play a wave file to remote party.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
<i>filename</i>	The full filepath, such as "/mnt/sdcard/test.wav".

<i>fileSamplesPerSec</i>	The wave file sample in seconds. It could be 8000, 16000 or 32000.
<i>loop</i>	Set to false to stop playing audio file when it is ended, or true to play it repeatedly.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.stopPlayAudioFileToRemote (long *sessionId*)**

Stop playing wave file to remote side.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.playAudioFileToRemoteAsBackground (long *sessionId*, String *filename*, int *fileSamplesPerSec*)**

Play a wave file to remote party as conversation background sound.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
<i>filename</i>	The full filepath, such as "/mnt/sdcard/test.wav".
<i>fileSamplesPerSec</i>	The wave file sample, in seconds. It should be 8000, 16000 or 32000.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.stopPlayAudioFileToRemoteAsBackground (long *sessionId*)**

Stop playing a wave file to remote party as background sound for the conversation.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**void com.portsip.PortSipSdk.audioPlayLoopbackTest (boolean *enable*)**

Used for testing loopback for the audio device.

**Parameters:**

<i>enable</i>	Set to true to start testing audio loopback test; or set to false to stop.
---------------	--

## Conference functions

### Functions

- int [com.portsip.PortSipSdk.createAudioConference](#) ()
- int [com.portsip.PortSipSdk.createVideoConference](#) ([PortSIPVideoRenderer](#) conferenceVideoWindow, int videoWidth, int videoHeight, boolean displayLocalVideoInConference)
- void [com.portsip.PortSipSdk.destroyConference](#) ()

- int [com.portsip.PortSipSdk.setConferenceVideoWindow](#) ([PortSIPVideoRenderer](#) conferenceVideoWindow)
- int [com.portsip.PortSipSdk.joinToConference](#) (long sessionId)
- int [com.portsip.PortSipSdk.removeFromConference](#) (long sessionId)

## Detailed Description

## Function Documentation

### int com.portsip.PortSipSdk.createAudioConference ()

Create an audio conference. It will fail if the existing conference is not ended yet.

#### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

### int com.portsip.PortSipSdk.createVideoConference ([PortSIPVideoRenderer](#) conferenceVideoWindow, int videoWidth, int videoHeight, boolean displayLocalVideoInConference)

Create a video conference. It will fail if the existing conference is not ended yet.

#### Parameters:

<i>conferenceVideoWindow</i>	<a href="#">SurfaceView</a> The window used for displaying the conference video.
<i>videoWidth</i>	Width of conference video resolution
<i>videoHeight</i>	Height of conference video resolution
<i>displayLocalVideoInConference</i>	Display local video during conference

#### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

### void com.portsip.PortSipSdk.destroyConference ()

End the exist conference.

### int com.portsip.PortSipSdk.setConferenceVideoWindow ([PortSIPVideoRenderer](#) conferenceVideoWindow)

Set the window for a conference that is used for displaying the received remote video image.

#### Parameters:

<i>conferenceVideoWindow</i>	<a href="#">SurfaceView</a> The window which is used for displaying the conference video
------------------------------	--

#### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

### int com.portsip.PortSipSdk.joinToConference (long sessionId)

Join a session into existing conference. If the call is in hold, it will be un-hold automatically.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.removeFromConference (long sessionId)**

Remove a session from an existing conference.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
------------------	-------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## RTP and RTCP QOS functions

### Functions

- int [com.portsip.PortSipSdk.setAudioRtcpBandwidth](#) (long sessionId, int BitsRR, int BitsRS, int KBitsAS)
- int [com.portsip.PortSipSdk.setVideoRtcpBandwidth](#) (long sessionId, int BitsRR, int BitsRS, int KBitsAS)
- int [com.portsip.PortSipSdk.enableAudioQos](#) (boolean state)
- int [com.portsip.PortSipSdk.enableVideoQos](#) (boolean state)
- int [com.portsip.PortSipSdk.setVideoMTU](#) (int mtu)

### Detailed Description

### Function Documentation

**int com.portsip.PortSipSdk.setAudioRtcpBandwidth (long sessionId, int BitsRR, int BitsRS, int KBitsAS)**

Set the audio RTCP bandwidth parameters as RFC3556.

**Parameters:**

<i>sessionId</i>	Set the audio RTCP bandwidth parameters as RFC3556.
<i>BitsRR</i>	The bits for the RR parameter.
<i>BitsRS</i>	The bits for the RS parameter.
<i>KBitsAS</i>	The Kbits for the AS parameter.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoRtcpBandwidth (long sessionId, int BitsRR, int BitsRS, int KBitsAS)**

Set the video RTCP bandwidth parameters as the RFC3556.

**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>BitsRR</i>	The bits for the RR parameter.
<i>BitsRS</i>	The bits for the RS parameter.
<i>KBitsAS</i>	The Kbits for the AS parameter.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.enableAudioQos (boolean state)**

Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for audio channel.

**Parameters:**

<i>state</i>	Set to YES to enable audio QoS and DSCP value will be 46; or NO to disable audio QoS and DSCP value will be 0.
--------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.enableVideoQos (boolean state)**

Set the DSCP(differentiated services code point) value of QoS(Quality of Service) for video channel.

**Parameters:**

<i>state</i>	Set to YES to enable video QoS and DSCP value will be 34; or NO to disable video QoS and DSCP value will be 0.
--------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setVideoMTU (int mtu)**

Set the MTU size for video RTP packet.

**Parameters:**

<i>mtu</i>	Set MTU value. Allow values range 512 - 65507. Default is 14000.
------------	--

**Returns:**

If the function succeeds, the return value is 0. If the function fails, the return value is a specific error code.

## RTP statistics functions

### Functions

- int [com.portsip.PortSipSdk.getAudioStatistics](#) (long sessionId, int[] statistics)
- int [com.portsip.PortSipSdk.getVideoStatistics](#) (long sessionId, int[] statistics)

## Detailed Description

## Function Documentation

**int com.portsip.PortSipSdk.getAudioStatistics (long *sessionId*, int [] *statistics*)**

Obtain the statistics of audio channel.

**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>statistics</i>	Return audio statistics statistics[0] - The number of sent bytes. statistics[1] - The number of sent packets. statistics[2] - The number of sent but lost packet. statistics[3] - Fraction of sent but lost packet in percentage. statistics[4] - The round-trip time of the session, in milliseconds. statistics[5] - Sent Audio codec type. statistics[6] - The sent jitter, in milliseconds. statistics[7] - The sent audio level.It ranges 0 - 9. statistics[8] - The number of received bytes. statistics[9] - The number of received packets. statistics[10] - The number of received but lost packets. statistics[11] - Fraction of received but lost packet in percentage. statistics[12] - Received Audio codec type. statistics[13] - The received jitter, in milliseconds. statistics[14] - The received audio level.It ranges 0 - 9.

**Returns:**

If the function succeeds, it will return value is 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.getVideoStatistics (long *sessionId*, int [] *statistics*)**

Obtain the statistics of video channel.

**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>statistics</i>	Return Video statistics statistics[0] - The number of sent bytes. statistics[1] - The number of sent packets. statistics[2] - The number of sent but lost packet. statistics[3] - Fraction of sent lost in percentage. statistics[4] - The round-trip time of the session, in milliseconds. statistics[5] - Send Video codec type. statistics[6] - Frame width for the sent video. statistics[7] - Frame height for the sent video. statistics[8] - Bitrate in BPS for the sent video. statistics[9] - Frame rate for the sent video. statistics[10] - The number of received bytes. statistics[11] - The number of received packets. statistics[12] - The number of received but lost packet. statistics[13] - Fraction of received but lost packet in percentage. statistics[14] - Received Video codec type. statistics[15] - Frame width for the received video. statistics[16] - Frame height for the received video. statistics[17] - (This parameter is not implemented yet)Bitrate in BPS for the received video. statistics[18] - Framerate for the received video. statistics[19] - The number of sent bytes.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

## Audio effect functions

### Functions

- void [com.portsip.PortSipSdk.enableVAD](#) (boolean state)
- void [com.portsip.PortSipSdk.enableAEC](#) (boolean state)
- void [com.portsip.PortSipSdk.enableCNG](#) (boolean state)
- void [com.portsip.PortSipSdk.enableAGC](#) (boolean state)
- void [com.portsip.PortSipSdk.enableANS](#) (boolean state)

### Detailed Description

### Function Documentation

**void com.portsip.PortSipSdk.enableVAD (boolean state)**

Enable/disable Voice Activity Detection(VAD).

**Parameters:**

<i>state</i>	Set to true to enable VAD, or false to disable.
--------------	---

**void com.portsip.PortSipSdk.enableAEC (boolean state)**

Enable/disable AEC (Acoustic Echo Cancellation).

**Parameters:**

<i>state</i>	Set to true to enable AEC, or false to disable.
--------------	---

**void com.portsip.PortSipSdk.enableCNG (boolean state)**

Enable/disable Comfort Noise Generator(CNG).

**Parameters:**

<i>state</i>	Set to true to enable CNG, or false to disable.
--------------	---

**void com.portsip.PortSipSdk.enableAGC (boolean state)**

Enable/disable Automatic Gain Control(AGC).

**Parameters:**

<i>state</i>	Set to true to enable AEC, or false to disable.
--------------	---

**void com.portsip.PortSipSdk.enableANS (boolean state)**

Enable/disable Audio Noise Suppression(ANS).

**Parameters:**

<i>state</i>	Set to true to enable ANS, or false to disable.
--------------	---

## Send OPTIONS/INFO/MESSAGE functions

### Functions

- int [com.portsip.PortSipSdk.sendOptions](#) (String to, String sdp)
- int [com.portsip.PortSipSdk.sendInfo](#) (long sessionId, String mimeType, String subMimeType, String infoContents)
- long [com.portsip.PortSipSdk.sendMessage](#) (long sessionId, String mimeType, String subMimeType, byte[] message, int messageLength)
- long [com.portsip.PortSipSdk.sendOutOfDialogMessage](#) (String to, String mimeType, String subMimeType, boolean isSMS, byte[] message, int messageLength)
- long [com.portsip.PortSipSdk.setPresenceMode](#) (int mode)
- long [com.portsip.PortSipSdk.setDefaultSubscriptionTime](#) (int secs)
- long [com.portsip.PortSipSdk.setDefaultPublicationTime](#) (int secs)
- long [com.portsip.PortSipSdk.presenceSubscribe](#) (String contact, String subject)
- int [com.portsip.PortSipSdk.presenceTerminateSubscribe](#) (long subscribeId)
- int [com.portsip.PortSipSdk.presenceAcceptSubscribe](#) (long subscribeId)
- int [com.portsip.PortSipSdk.presenceRejectSubscribe](#) (long subscribeId)
- int [com.portsip.PortSipSdk.setPresenceStatus](#) (long subscribeId, String statusText)
- long [com.portsip.PortSipSdk.sendSubscription](#) (String to, String eventName)  
*Send a SUBSCRIBE message to subscribe an event.*
- int [com.portsip.PortSipSdk.terminateSubscription](#) (long subscribeId)

---

### Detailed Description

---

### Function Documentation

#### **int com.portsip.PortSipSdk.sendOptions (String to, String sdp)**

Send OPTIONS message.

##### Parameters:

<i>to</i>	The recipient of OPTIONS message.
<i>sdp</i>	The SDP of OPTIONS message. It's optional if user does not want to send the SDP with OPTIONS message.

##### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return value a specific error code.

#### **int com.portsip.PortSipSdk.sendInfo (long sessionId, String mimeType, String subMimeType, String infoContents)**

Send a INFO message to remote side in dialog.

##### Parameters:

<i>sessionId</i>	The session ID of call.
<i>mimeType</i>	The mime type of INFO message.
<i>subMimeType</i>	The sub mime type of INFO message.
<i>infoContents</i>	The contents that will be sent with INFO message.

##### Returns:

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**long com.portsip.PortSipSdk.sendMessage (long *sessionId*, String *contentType*, String *subMimeType*, byte [] *message*, int *messageLength*)**

Send a MESSAGE message to remote side in dialog.

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>contentType</i>	The mime type of MESSAGE message.
<i>subMimeType</i>	The sub mime type of MESSAGE message.
<i>message</i>	The contents that will be sent with MESSAGE message. Binary data allowed.
<i>messageLength</i>	The message size.

**Returns:**

If the function succeeds, it will return a message ID that allows to track the message sending state in `onSendMessageSuccess` and `onSendMessageFailure`. If the function fails, it will return a specific error code that is less than 0.

**Remarks:**

Example 1: Send a plain text message. Note: to send other languages text, please use the UTF8 to encode the message before sending.

```
sendMessage(sessionId, "text", "plain", "hello", 6);
```

Example 2: Send a binary message.

```
sendMessage(sessionId, "application", "vnd.3gpp.sms", binData, binDataSize);
```

**long com.portsip.PortSipSdk.sendOutOfDialogMessage (String *to*, String *contentType*, String *subMimeType*, boolean *isSMS*, byte [] *message*, int *messageLength*)**

Send a out of dialog MESSAGE message to remote side.

**Parameters:**

<i>to</i>	The message receiver. Likes sip: <a href="mailto:receiver@portsip.com">receiver@portsip.com</a>
<i>contentType</i>	The mime type of MESSAGE message.
<i>subMimeType</i>	The sub mime type of MESSAGE message.
<i>isSMS</i>	Set to YES to specify "messagetype=SMS" in the To line, or NO to disable.
<i>message</i>	The contents that will be sent with MESSAGE message. Binary data allowed.
<i>messageLength</i>	The message size.

**Returns:**

If the function succeeds, it will return a message ID that allows to track the message sending state in `onSendOutOfMessageSuccess` and `onSendOutOfMessageFailure`. If the function fails, it will return a specific error code that is less than 0.

**Remarks:**

Example 1: Send a plain text message. Note: to send other languages text, please use the UTF8 to encode the message before sending.

```
sendOutOfDialogMessage("sip:user1@sip.portsip.com", "text", "plain", "hello", 6);
```

Example 2: Send a binary message.

```
sendOutOfDialogMessage("sip:user1@sip.portsip.com", "application", "vnd.3gpp.sms", binData, binDataSize);
```

**long com.portsip.PortSipSdk.setPresenceMode (int *mode*)**

Indicate the SDK uses the P2P mode for presence or presence agent mode.

**Parameters:**

<i>mode</i>	0 - P2P mode; 1 - Presence Agent mode. Default is P2P mode.
-------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

Since presence agent mode requires the PBX/Server support the PUBLISH, please ensure you have your server and PortSIP PBX support this feature. For more details please visit: <https://www.portsip.com/portsip-pbx>

**long com.portsip.PortSipSdk.setDefaultSubscriptionTime (int secs)**

Set the default expiration time to be used when creating a subscription.

**Parameters:**

<i>secs</i>	The default expiration time of subscription.
-------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**long com.portsip.PortSipSdk.setDefaultPublicationTime (int secs)**

Set the default expiration time to be used when creating a publication.

**Parameters:**

<i>secs</i>	The default expiration time of publication.
-------------	---

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**long com.portsip.PortSipSdk.presenceSubscribe (String contact, String subject)**

Send a SUBSCRIBE message for presence to a contact.

**Parameters:**

<i>contact</i>	The target contact, it must be in the format of sip: <a href="mailto:contact001@sip.portsip.com">contact001@sip.portsip.com</a> .
<i>subject</i>	This subject text will be inserted into the SUBSCRIBE message. For example: "Hello, I'm Jason". The subject maybe is in UTF8 format. You should use UTF8 to decode it.

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.presenceTerminateSubscribe (long subscribeId)**

Terminate the given presence subscription.

**Parameters:**

<i>subscribeId</i>	The ID of the subscription.
--------------------	-----------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.presenceAcceptSubscribe (long subscribeId)**

Accept the presence SUBSCRIBE request which received from contact.

**Parameters:**

<i>subscribeId</i>	Subscription ID. When receiving a SUBSCRIBE request from contact, the event onPresenceRecvSubscribe will be triggered. The event includes the subscription ID.
--------------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.presenceRejectSubscribe (long *subscribeId*)**

Reject a presence SUBSCRIBE request received from contact.

**Parameters:**

<i>subscribeId</i>	Subscription ID. When receiving a SUBSCRIBE request from contact, the event onPresenceRecvSubscribe will be triggered. The event includes the subscription ID.
--------------------	--

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**int com.portsip.PortSipSdk.setPresenceStatus (long *subscribeId*, String *statusText*)**

Send a NOTIFY message to contact to notify that presence status is online/offline/changed.

**Parameters:**

<i>subscribeId</i>	Subscription ID. When receiving a SUBSCRIBE request from contact, the event onPresenceRecvSubscribe that includes the Subscription ID will be triggered.
<i>statusText</i>	The state text of presence status. For example: "I'm here", offline must use "offline"

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**long com.portsip.PortSipSdk.sendSubscription (String *to*, String *eventName*)**

Send a SUBSCRIBE message to subscribe an event.

**Parameters:**

<i>to</i>	The user/extension will be subscribed.
<i>eventName</i>	The event name to be subscribed.

**Returns:**

If the function succeeds, it will return the ID of that SUBSCRIBE which is greater than 0. If the function fails, it will return a specific error code which is less than 0.

**Remarks:**

Example 1, below code indicates that user/extension 101 is subscribed to MWI (Message Waiting notifications) for checking his voicemail: `int32 mwiSubId = sendSubscription("sip:101@test.com", "message-summary");`

Example 2, to monitor a user/extension call status, You can use code: `sendSubscription(mSipLib, "100", "dialog");` Extension 100 refers to the user/extension to be monitored. Once being monitored, when extension 100 hold a call or is ringing, the `onDialogStateUpdated` callback will be triggered.

**int com.portsip.PortSipSdk.terminateSubscription (long *subscribeId*)**

Terminate the given subscription.

**Parameters:**

<i>subscribeId</i>	The ID of the subscription.
--------------------	-----------------------------

**Returns:**

If the function succeeds, it will return value 0. If the function fails, it will return a specific error code.

**Remarks:**

For example, if you want stop check the MWI, use below code:

```
terminateSubscription(mwiSubId);
```

# Class Documentation

## **com.portsip.PortSipEnumDefine.AUDIOCODEC Interface Reference**

---

The documentation for this interface was generated from the following file:

- PortSipEnumDefine.java

## com.portsip.PortSipEnumDefine.AudioDevice Enum Reference

### Public Attributes

- SPEAKER\_PHONE
  - WIRED\_HEADSET
  - EARPIECE
  - BLUETOOTH
  - NONE
- 

### Detailed Description

[AudioDevice](#) lists possible audio devices that we currently support.

---

The documentation for this enum was generated from the following file:

- PortSipEnumDefine.java

## com.portsip.OnPortSIPEvent Interface Reference

### Public Member Functions

- void [onRegisterSuccess](#) (String reason, int code, String sipMessage)
- void [onRegisterFailure](#) (String reason, int code, String sipMessage)
- void [onInviteIncoming](#) (long sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [onInviteTrying](#) (long sessionId)
- void [onInviteSessionProgress](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsEarlyMedia, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [onInviteRinging](#) (long sessionId, String statusText, int statusCode, String sipMessage)
- void [onInviteAnswered](#) (long sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [onInviteFailure](#) (long sessionId, String reason, int code, String sipMessage)
- void [onInviteUpdated](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo, String sipMessage)
- void [onInviteConnected](#) (long sessionId)
- void [onInviteBeginingForward](#) (String forwardTo)
- void [onInviteClosed](#) (long sessionId)
- void [onDialogStateUpdated](#) (String BLFMonitoredUri, String BLFDialogState, String BLFDialogId, String BLFDialogDirection)
- void [onRemoteHold](#) (long sessionId)
- void [onRemoteUnHold](#) (long sessionId, String audioCodecs, String videoCodecs, boolean existsAudio, boolean existsVideo)
- void [onReceivedRefer](#) (long sessionId, long referId, String to, String from, String referSipMessage)
- void [onReferAccepted](#) (long sessionId)
- void [onReferRejected](#) (long sessionId, String reason, int code)
- void [onTransferTrying](#) (long sessionId)
- void [onTransferRinging](#) (long sessionId)
- void [onACTVTransferSuccess](#) (long sessionId)
- void [onACTVTransferFailure](#) (long sessionId, String reason, int code)
- void [onReceivedSignaling](#) (long sessionId, String message)
- void [onSendingSignaling](#) (long sessionId, String message)
- void [onWaitingVoiceMessage](#) (String messageAccount, int urgentNewMessageCount, int urgentOldMessageCount, int newMessageCount, int oldMessageCount)
- void [onWaitingFaxMessage](#) (String messageAccount, int urgentNewMessageCount, int urgentOldMessageCount, int newMessageCount, int oldMessageCount)
- void [onRecvDtmfTone](#) (long sessionId, int tone)
- void [onRecvOptions](#) (String optionsMessage)
- void [onRecvInfo](#) (String infoMessage)
- void [onRecvNotifyOfSubscription](#) (long subscribeId, String notifyMessage, byte[] messageData, int messageDataLength)
- void [onPresenceRecvSubscribe](#) (long subscribeId, String fromDisplayName, String from, String subject)
- void [onPresenceOnline](#) (String fromDisplayName, String from, String stateText)
- void [onPresenceOffline](#) (String fromDisplayName, String from)
- void [onRecvMessage](#) (long sessionId, String mimeType, String subMimeType, byte[] messageData, int messageDataLength)
- void [onRecvOutOfDialogMessage](#) (String fromDisplayName, String from, String toDisplayName, String to, String mimeType, String subMimeType, byte[] messageData, int messageDataLength, String sipMessage)
- void [onSendMessageSuccess](#) (long sessionId, long messageId)
- void [onSendMessageFailure](#) (long sessionId, long messageId, String reason, int code)
- void [onSendOutOfDialogMessageSuccess](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to)

- void [onSendOutOfDialogMessageFailure](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String reason, int code)
- void [onSubscriptionFailure](#) (long subscribeId, int statusCode)
- void [onSubscriptionTerminated](#) (long subscribeId)
- void [onPlayAudioFileFinished](#) (long sessionId, String fileName)
- void [onPlayVideoFileFinished](#) (long sessionId)
- void [onReceivedRTPPacket](#) (long sessionId, boolean isAudio, byte[] RTPPacket, int packetSize)
- void [onSendingRTPPacket](#) (long sessionId, boolean isAudio, byte[] RTPPacket, int packetSize)
- void [onAudioRawCallback](#) (long sessionId, int enum\_audioCallbackMode, byte[] data, int dataLength, int samplingFreqHz)
- void [onVideoRawCallback](#) (long sessionId, int enum\_videoCallbackMode, int width, int height, byte[] data, int dataLength)

---

The documentation for this interface was generated from the following file:

- OnPortSIPEvent.java

## com.portsip.PortSIPCameraCapturer Class Reference

Inherits CapturerObserver.

### Public Member Functions

- void **InitializeVideoCapturer** (Context applicationContext)
- void **UnInitializeVideoCapturer** ()
- void **switchCamera** (int newDeviceId)
- void **onCapturerStarted** (boolean success)
- void **onCapturerStopped** ()
- void **onByteBufferFrameCaptured** (byte[] data, int width, int height, int rotation, long timeStamp)
- void **onTextureFrameCaptured** (int width, int height, int oesTextureId, float[] transformMatrix, int rotation, long timestamp)
- void **onFrameCaptured** (VideoFrame frame)

### Public Attributes

- CameraVideoCapturer **capturer**

---

The documentation for this class was generated from the following file:

- PortSIPCameraCapturer.java

## com.portsip.PortSipEnumDefine Class Reference

### Classes

- interface [AUDIOCODEC](#)
- enum [AudioDevice](#)

### Static Public Attributes

- static final int **ENUM\_AUDIOCODEC\_G729** = 18
  - static final int **ENUM\_AUDIOCODEC\_PCMA** = 8
  - static final int **ENUM\_AUDIOCODEC\_PCMU** = 0
  - static final int **ENUM\_AUDIOCODEC\_GSM** = 3
  - static final int **ENUM\_AUDIOCODEC\_G722** = 9
  - static final int **ENUM\_AUDIOCODEC\_ILBC** = 97
  - static final int **ENUM\_AUDIOCODEC\_AMR** = 98
  - static final int **ENUM\_AUDIOCODEC\_AMRWB** = 99
  - static final int **ENUM\_AUDIOCODEC\_SPEEX** = 100
  - static final int **ENUM\_AUDIOCODEC\_SPEEXWB** = 102
  - static final int **ENUM\_AUDIOCODEC\_ISACWB** = 103
  - static final int **ENUM\_AUDIOCODEC\_ISACSWB** = 104
  - static final int **ENUM\_AUDIOCODEC\_OPUS** = 105
  - static final int **ENUM\_AUDIOCODEC\_DTMF** = 101
  - static final int [ENUM\\_VIDEOCODEC\\_NONE](#) = -1
  - static final int [ENUM\\_VIDEOCODEC\\_I420](#) = 133
  - static final int **ENUM\_VIDEOCODEC\_H264** = 125
  - static final int **ENUM\_VIDEOCODEC\_VP8** = 120
  - static final int **ENUM\_VIDEOCODEC\_VP9** = 122
  - static final int **ENUM\_SRTPPOLICY\_NONE** = 0
  - static final int **ENUM\_SRTPPOLICY\_FORCE** = 1
  - static final int **ENUM\_SRTPPOLICY\_PREFER** = 2
  - static final int **ENUM\_TRANSPORT\_UDP** = 0
  - static final int **ENUM\_TRANSPORT\_TLS** = 1
  - static final int **ENUM\_TRANSPORT\_TCP** = 2
  - static final int **ENUM\_TRANSPORT\_PERS\_UDP** = 3
  - static final int **ENUM\_TRANSPORT\_PERS\_TCP** = 4
  - static final int **ENUM\_LOG\_LEVEL\_NONE** = -1
  - static final int **ENUM\_LOG\_LEVEL\_ERROR** = 1
  - static final int **ENUM\_LOG\_LEVEL\_WARNING** = 2
  - static final int **ENUM\_LOG\_LEVEL\_INFO** = 3
  - static final int **ENUM\_LOG\_LEVEL\_DEBUG** = 4
  - static final int **ENUM\_DTMF\_MOTHOD\_RFC2833** = 0
  - static final int **ENUM\_DTMF\_MOTHOD\_INFO** = 1
  - static final int **ENUM\_AUDIOSTREAM\_NONE** = 0
  - static final int [ENUM\\_AUDIOSTREAM\\_LOCAL\\_PER\\_CHANNEL](#) = 1
  - static final int [ENUM\\_AUDIOSTREAM\\_REMOTE\\_PER\\_CHANNEL](#) = 2
  - static final int [ENUM\\_AUDIOSTREAM\\_BOTH\\_PER\\_CHANNEL](#) = 3
  - static final int [ENUM\\_VIDEOSTREAM\\_NONE](#) = 0
  - static final int [ENUM\\_VIDEOSTREAM\\_LOCAL](#) = 1
  - static final int [ENUM\\_VIDEOSTREAM\\_REMOTE](#) = 2
  - static final int [ENUM\\_VIDEOSTREAM\\_BOTH](#) = 3
  - static final int [ENUM\\_RECORD\\_MODE\\_NONE](#) = 0
  - static final int [ENUM\\_RECORD\\_MODE\\_RECV](#) = 1
  - static final int [ENUM\\_RECORD\\_MODE\\_SEND](#) = 2
  - static final int [ENUM\\_RECORD\\_MODE\\_BOTH](#) = 3
  - static final int **ENUM\_AUDIO\_FILE\_FORMAT\_WAVE** = 1
  - static final int **ENUM\_AUDIO\_FILE\_FORMAT\_AMR** = 2
-

## Member Data Documentation

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOCODEC\_NONE = -1 [static]**

Used in startRecord only

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOCODEC\_I420 = 133 [static]**

Used in startRecord only

**final int  
com.portsip.PortSipEnumDefine.ENUM\_AUDIOSTREAM\_LOCAL\_PER\_CHANNEL =  
1 [static]**

Callback the audio stream from microphone for one channel base on the session ID

**final int  
com.portsip.PortSipEnumDefine.ENUM\_AUDIOSTREAM\_REMOTE\_PER\_CHANNEL =  
2 [static]**

Callback the received audio stream for one channel base on the session ID.

**final int  
com.portsip.PortSipEnumDefine.ENUM\_AUDIOSTREAM\_BOTH\_PER\_CHANNEL =  
3 [static]**

Callback microphone and received audio stream for one channel based on the session ID.

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOSTREAM\_NONE = 0 [static]**

Disable video stream callback

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOSTREAM\_LOCAL = 1 [static]**

Local video stream callback

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOSTREAM\_REMOTE =  
2 [static]**

Remote video stream callback

**final int com.portsip.PortSipEnumDefine.ENUM\_VIDEOSTREAM\_BOTH = 3 [static]**

Both of local and remote video stream callback

**final int com.portsip.PortSipEnumDefine.ENUM\_RECORD\_MODE\_NONE = 0 [static]**

Not Recorded.

**final int com.portsip.PortSipEnumDefine.ENUM\_RECORD\_MODE\_RECV = 1 [static]**

Only record the received data.

**final int com.portsip.PortSipEnumDefine.ENUM\_RECORD\_MODE\_SEND = 2 [static]**

Only record the sent data.

**final int com.portsip.PortSipEnumDefine.ENUM\_RECORD\_MODE\_BOTH = 3 [static]**

Record both received and sent data.

**The documentation for this class was generated from the following file:**

- PortSipEnumDefine.java

## com.portsip.PortSipErrorcode Class Reference

### Static Public Attributes

- static final int **ECoreErrorNone** = 0
- static final int **INVALID\_SESSION\_ID** = -1
- static final int **ECoreAlreadyInitialized** = -60000
- static final int **ECoreNotInitialized** = -60001
- static final int **ECoreSDKObjectNull** = -60002
- static final int **ECoreArgumentNull** = -60003
- static final int **ECoreInitializeWinsocFailure** = -60004
- static final int **ECoreUserNameAuthNameEmpty** = -60005
- static final int **ECoreInitiazeStackFailure** = -60006
- static final int **ECorePortOutOfRange** = -60007
- static final int **ECoreAddTcpTransportFailure** = -60008
- static final int **ECoreAddTlsTransportFailure** = -60009
- static final int **ECoreAddUdpTransportFailure** = -60010
- static final int **ECoreNotSupportMediaType** = -60011
- static final int **ECoreNotSupportDTMFValue** = -60012
- static final int **ECoreAlreadyRegistered** = -60021
- static final int **ECoreSIPServerEmpty** = -60022
- static final int **ECoreExpiresValueTooSmall** = -60023
- static final int **ECoreCallIdNotFound** = -60024
- static final int **ECoreNotRegistered** = -60025
- static final int **ECoreCalleeEmpty** = -60026
- static final int **ECoreInvalidUri** = -60027
- static final int **ECoreAudioVideoCodecEmpty** = -60028
- static final int **ECoreNoFreeDialogSession** = -60029
- static final int **ECoreCreateAudioChannelFailed** = -60030
- static final int **ECoreSessionTimerValueTooSmall** = -60040
- static final int **ECoreAudioHandleNull** = -60041
- static final int **ECoreVideoHandleNull** = -60042
- static final int **ECoreCallsClosed** = -60043
- static final int **ECoreCallAlreadyHold** = -60044
- static final int **ECoreCallNotEstablished** = -60045
- static final int **ECoreCallNotHold** = -60050
- static final int **ECoreSipMessaegEmpty** = -60051
- static final int **ECoreSipHeaderNotExist** = -60052
- static final int **ECoreSipHeaderValueEmpty** = -60053
- static final int **ECoreSipHeaderBadFormed** = -60054
- static final int **ECoreBufferTooSmall** = -60055
- static final int **ECoreSipHeaderValueListEmpty** = -60056
- static final int **ECoreSipHeaderParserEmpty** = -60057
- static final int **ECoreSipHeaderValueListNull** = -60058
- static final int **ECoreSipHeaderNameEmpty** = -60059
- static final int **ECoreAudioSampleNotmultiple** = -60060
- static final int **ECoreAudioSampleOutOfRange** = -60061
- static final int **ECoreInviteSessionNotFound** = -60062
- static final int **ECoreStackException** = -60063
- static final int **ECoreMimeTypeUnknown** = -60064
- static final int **ECoreDataSizeTooLarge** = -60065
- static final int **ECoreSessionNumsOutOfRange** = -60066
- static final int **ECoreNotSupportCallbackMode** = -60067
- static final int **ECoreNotFoundSubscribeId** = -60068
- static final int **ECoreCodecNotSupport** = -60069
- static final int **ECoreCodecParameterNotSupport** = -60070
- static final int **ECorePayloadOutofRange** = -60071
- static final int **ECorePayloadHasExist** = -60072
- static final int **ECoreFixPayloadCantChange** = -60073

- static final int **ECoreCodecTypeInvalid** = -60074
- static final int **ECoreCodecWasExist** = -60075
- static final int **ECorePayloadTypeInvalid** = -60076
- static final int **ECoreArgumentTooLong** = -60077
- static final int **ECoreMiniRtpPortMustIsEvenNum** = -60078
- static final int **ECoreCallInHold** = -60079
- static final int **ECoreNotIncomingCall** = -60080
- static final int **ECoreCreateMediaEngineFailure** = -60081
- static final int **ECoreAudioCodecEmptyButAudioEnabled** = -60082
- static final int **ECoreVideoCodecEmptyButVideoEnabled** = -60083
- static final int **ECoreNetworkInterfaceUnavailable** = -60084
- static final int **ECoreWrongDTMFTone** = -60085
- static final int **ECoreWrongLicenseKey** = -60086
- static final int **ECoreTrialVersionLicenseKey** = -60087
- static final int **ECoreOutgoingAudioMuted** = -60088
- static final int **ECoreOutgoingVideoMuted** = -60089
- static final int **ECoreFailedCreateSdp** = -60090
- static final int **ECoreTrialVersionExpired** = -60091
- static final int **ECoreStackFailure** = -60092
- static final int **ECoreTransportExists** = -60093
- static final int **ECoreUnsupportTransport** = -60094
- static final int **ECoreAllowOnlyOneUser** = -60095
- static final int **ECoreUserNotFound** = -60096
- static final int **ECoreTransportsIncorrect** = -60097
- static final int **ECoreCreateTransportFailure** = -60098
- static final int **ECoreTransportNotSet** = -60099
- static final int **ECoreECreateSignalingFailure** = -60100
- static final int **ECoreArgumentIncorrect** = -60101
- static final int **ECoreIVRObjectNull** = -61001
- static final int **ECoreIVRIndexOutOfRange** = -61002
- static final int **ECoreIVRReferFailure** = -61003
- static final int **ECoreIVRWaitingTimeOut** = -61004
- static final int **EAudioFileNameEmpty** = -70000
- static final int **EAudioChannelNotFound** = -70001
- static final int **EAudioStartRecordFailure** = -70002
- static final int **EAudioRegisterRecodingFailure** = -70003
- static final int **EAudioRegisterPlaybackFailure** = -70004
- static final int **EAudioGetStatisticsFailure** = -70005
- static final int **EAudioPlayFileAlreadyEnable** = -70006
- static final int **EAudioPlayObjectNotExist** = -70007
- static final int **EAudioPlaySteamNotEnabled** = -70008
- static final int **EAudioRegisterCallbackFailure** = -70009
- static final int **EAudioCreateAudioConferenceFailure** = -70010
- static final int **EAudioOpenPlayFileFailure** = -70011
- static final int **EAudioPlayFileModeNotSupport** = -70012
- static final int **EAudioPlayFileFormatNotSupport** = -70013
- static final int **EAudioPlaySteamAlreadyEnabled** = -70014
- static final int **EAudioCreateRecordFileFailure** = -70015
- static final int **EAudioCodecNotSupport** = -70016
- static final int **EAudioPlayFileNotEnabled** = -70017
- static final int **EAudioPlayFileUnknowSeekOrigin** = -70018
- static final int **EAudioCantSetDeviceIdDuringCall** = -70019
- static final int **EAudioVolumeOutOfRange** = -70020
- static final int **EVideoFileNameEmpty** = -80000
- static final int **EVideoGetDeviceNameFailure** = -80001
- static final int **EVideoGetDeviceIdFailure** = -80002
- static final int **EVideoStartCaptureFailure** = -80003
- static final int **EVideoChannelNotFound** = -80004
- static final int **EVideoStartSendFailure** = -80005
- static final int **EVideoGetStatisticsFailure** = -80006

- static final int **EVideoStartPlayAviFailure** = -80007
- static final int **EVideoSendAviFileFailure** = -80008
- static final int **EVideoRecordUnknowCodec** = -80009
- static final int **EVideoCantSetDeviceIdDuringCall** = -80010
- static final int **EVideoUnsupportCaptureRotate** = -80011
- static final int **VideoUnsupportCaptureResolution** = -80012
- static final int **ECameraSwitchTooOften** = -80013
- static final int **EMTUOutOfRange** = -80014
- static final int **EDeviceGetDeviceNameFailure** = -90001

---

The documentation for this class was generated from the following file:

- PortSipErrorcode.java

# com.portsip.PortSipSdk Class Reference

## Classes

- class **MainHandler**

## Public Member Functions

- void [CreateCallManager](#) (Context context)
- PortSipEnumDefine.AudioDevice **getSelectedAudioDevice** ()
- void [DeleteCallManager](#) ()
- int [initialize](#) (int enum\_transport, String localIP, int localSIPPort, int enum\_LogLevel, String LogPath, int maxLines, String agent, int audioDeviceLayer, int videoDeviceLayer, String TLSCertificatesRootPath, String TLSCipherList, boolean verifyTLSCertificate, String dnsServers)
- int [setInstanceId](#) (String instanceId)
- int [setUser](#) (String userName, String displayName, String authName, String password, String userDomain, String SIPServer, int SIPServerPort, String STUNServer, int STUNServerPort, String outboundServer, int outboundServerPort)
- void [removeUser](#) ()  
*remove user account info.*
- int [registerServer](#) (int expires, int retryTimes)
- int [refreshRegistration](#) (int expires)
- int [unRegisterServer](#) ()
- int [setLicenseKey](#) (String key)
- int [addAudioCodec](#) (int enum\_audiocodec)
- int [addVideoCodec](#) (int enum\_videocodec)
- boolean [isAudioCodecEmpty](#) ()
- boolean [isVideoCodecEmpty](#) ()
- int [setAudioCodecPayloadType](#) (int enum\_audiocodec, int payloadType)
- int [setVideoCodecPayloadType](#) (int enum\_videocodec, int payloadType)
- void [clearAudioCodec](#) ()
- void [clearVideoCodec](#) ()
- int [setAudioCodecParameter](#) (int enum\_audiocodec, String sdpParameter)
- int [setVideoCodecParameter](#) (int enum\_videocodec, String sdpParameter)
- String [getVersion](#) ()
- int [enableRport](#) (boolean enable)
- int [enableEarlyMedia](#) (boolean enable)  
*Enable/disable rport(RFC3581).*
- int [enableReliableProvisional](#) (boolean enable)
- int [enable3GppTags](#) (boolean enable)
- void [enableCallbackSignaling](#) (boolean enableSending, boolean enableReceived)
- void [setSrtpPolicy](#) (int enum\_srtpolicy)
- int [setRtpPortRange](#) (int minimumRtpAudioPort, int maximumRtpAudioPort, int minimumRtpVideoPort, int maximumRtpVideoPort)
- int [setRtcpPortRange](#) (int minimumRtcpAudioPort, int maximumRtcpAudioPort, int minimumRtcpVideoPort, int maximumRtcpVideoPort)
- int [enableCallForward](#) (boolean forBusyOnly, String forwardTo)
- int [disableCallForward](#) ()
- int [enableSessionTimer](#) (int timerSeconds)
- void [disableSessionTimer](#) ()
- void [setDoNotDisturb](#) (boolean state)
- void [enableAutoCheckMwi](#) (boolean state)
- int [setRtpKeepAlive](#) (boolean state, int keepAlivePayloadType, int deltaTransmitTimeMS)
- int [setKeepAliveTime](#) (int keepAliveTime)
- int [setAudioSamples](#) (int ptime, int maxptime)
- int [addSupportedMimeType](#) (String methodName, String mimeType, String subMimeType)
- String [getSipMessageHeaderValue](#) (String sipMessage, String headerName)
- long [addSipMessageHeader](#) (long sessionId, String methodName, int msgType, String headerName, String headerValue)

- int [removeAddedSipMessageHeader](#) (long addedSipMessageId)
- void [clearAddedSipMessageHeaders](#) ()
- long [modifySipMessageHeader](#) (long sessionId, String methodName, int msgType, String headerName, String headerValue)
- int [removeModifiedSipMessageHeader](#) (long modifiedSipMessageId)
- void [clearModifiedSipMessageHeaders](#) ()
- int [setVideoDeviceId](#) (int deviceId)
- int [enableVideoHardwareCodec](#) (boolean enableHWEncoder, boolean enableHWDecoder)
- int [setVideoResolution](#) (int width, int height)
- int [setVideoCropAndScale](#) (boolean enable)
- int [setAudioBitrate](#) (long sessionId, int enum\_audiocodec, int bitrateKbps)
- int [setVideoBitrate](#) (long sessionId, int bitrateKbps)
- int [setVideoFrameRate](#) (long sessionId, int frameRate)
- int [sendVideo](#) (long sessionId, boolean send)
- void [setLocalVideoWindow](#) ([PortSIPVideoRenderer](#) renderer)
- int [setRemoteVideoWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- void [displayLocalVideo](#) (boolean state)
- int [setVideoNackStatus](#) (boolean state)
- int [setChannelOutputVolumeScaling](#) (long sessionId, int scaling)
- int [setChannelInputVolumeScaling](#) (long sessionId, int scaling)
- Set< PortSipEnumDefine.AudioDevice > [getAudioDevices](#) ()
- int [setAudioDevice](#) (PortSipEnumDefine.AudioDevice defaultDevice)
- long [call](#) (String callee, boolean sendSdp, boolean videoCall)
- int [rejectCall](#) (long sessionId, int code)
- int [hangUp](#) (long sessionId)
- int [answerCall](#) (long sessionId, boolean videoCall)
- int [updateCall](#) (long sessionId, boolean enableAudio, boolean enableVideo)
- int [hold](#) (long sessionId)
- int [unHold](#) (long sessionId)
- int [muteSession](#) (long sessionId, boolean muteIncomingAudio, boolean muteOutgoingAudio, boolean muteIncomingVideo, boolean muteOutgoingVideo)
- int [forwardCall](#) (long sessionId, String forwardTo)
- long [pickupBLFCall](#) (String replaceDialogId, boolean videoCall)
- int [sendDtmf](#) (long sessionId, int enum\_dtmfMethod, int code, int dtmfDuration, boolean playDtmfTone)
- int [refer](#) (long sessionId, String referTo)
- int [attendedRefer](#) (long sessionId, long replaceSessionId, String referTo)
- int [attendedRefer2](#) (long sessionId, long replaceSessionId, String replaceMethod, String target, String referTo)
- int [outOfDialogRefer](#) (long replaceSessionId, String replaceMethod, String target, String referTo)
- long [acceptRefer](#) (long referId, String referSignaling)
- int [rejectRefer](#) (long referId)
- int [enableSendPcmStreamToRemote](#) (long sessionId, boolean state, int streamSamplesPerSec)
- int [sendPcmStreamToRemote](#) (long sessionId, byte[] data, int dataLength)
- int [enableSendVideoStreamToRemote](#) (long sessionId, boolean state)
- int [sendVideoStreamToRemote](#) (long sessionId, byte[] data, int dataLength, int width, int height)
- void [setRtpCallback](#) (boolean enable)
- void [enableAudioStreamCallback](#) (long sessionId, boolean enable, int enum\_audioCallbackMode)
- void [enableVideoStreamCallback](#) (long sessionId, int enum\_videoCallbackMode)
- int [startRecord](#) (long sessionId, String recordFilePath, String recordFileName, boolean appendTimeStamp, int enum\_audioFileFormat, int enum\_audioRecordMode, int enum\_videocodec, int enum\_videoRecordMode)
- int [stopRecord](#) (long sessionId)
- int [playVideoFileToRemote](#) (long sessionId, String aviFile, boolean loop, boolean playAudio)
- int [stopPlayVideoFileToRemote](#) (long sessionId)
- int [playAudioFileToRemote](#) (long sessionId, String filename, int fileSamplesPerSec, boolean loop)
- int [stopPlayAudioFileToRemote](#) (long sessionId)
- int [playAudioFileToRemoteAsBackground](#) (long sessionId, String filename, int fileSamplesPerSec)
- int [stopPlayAudioFileToRemoteAsBackground](#) (long sessionId)

- void [audioPlayLoopbackTest](#) (boolean enable)
- int [createAudioConference](#) ()
- int [createVideoConference](#) ([PortSIPVideoRenderer](#) conferenceVideoWindow, int videoWidth, int videoHeight, boolean displayLocalVideoInConference)
- void [destroyConference](#) ()
- int [setConferenceVideoWindow](#) ([PortSIPVideoRenderer](#) conferenceVideoWindow)
- int [joinToConference](#) (long sessionId)
- int [removeFromConference](#) (long sessionId)
- int [setAudioRtcpBandwidth](#) (long sessionId, int BitsRR, int BitsRS, int KBitsAS)
- int [setVideoRtcpBandwidth](#) (long sessionId, int BitsRR, int BitsRS, int KBitsAS)
- int [enableAudioQos](#) (boolean state)
- int [enableVideoQos](#) (boolean state)
- int [setVideoMTU](#) (int mtu)
- int [getAudioStatistics](#) (long sessionId, int[] statistics)
- int [getVideoStatistics](#) (long sessionId, int[] statistics)
- void [enableVAD](#) (boolean state)
- void [enableAEC](#) (boolean state)
- void [enableCNG](#) (boolean state)
- void [enableAGC](#) (boolean state)
- void [enableANS](#) (boolean state)
- int [sendOptions](#) (String to, String sdp)
- int [sendInfo](#) (long sessionId, String mimeType, String subMimeType, String infoContents)
- long [sendMessage](#) (long sessionId, String mimeType, String subMimeType, byte[] message, int messageLength)
- long [sendOutOfDialogMessage](#) (String to, String mimeType, String subMimeType, boolean isSMS, byte[] message, int messageLength)
- long [setPresenceMode](#) (int mode)
- long [setDefaultSubscriptionTime](#) (int secs)
- long [setDefaultPublicationTime](#) (int secs)
- long [presenceSubscribe](#) (String contact, String subject)
- int [presenceTerminateSubscribe](#) (long subscribeId)
- int [presenceAcceptSubscribe](#) (long subscribeId)
- int [presenceRejectSubscribe](#) (long subscribeId)
- int [setPresenceStatus](#) (long subscribeId, String statusText)
- long [sendSubscription](#) (String to, String eventName)  
*Send a SUBSCRIBE message to subscribe an event.*
- int [terminateSubscription](#) (long subscribeId)
- void [receiveSIPEvent](#) (long sipCommand)
- void [receivedRTPPacket](#) (long sessionId, boolean isAudio, byte[] RTPPacket, int packetSize)
- void [sendingRTPPacket](#) (long sessionId, boolean isAudio, byte[] RTPPacket, int packetSize)
- void [audioRawCallback](#) (long sessionId, int enum\_audioCallbackMode, byte[] data, int dataLength, int samplingFreqHz)
- void [videoRawCallback](#) (long sessionId, int enum\_videoCallbackMode, int width, int height, byte[] data, int dataLength)
- void [setOnPortSIPEvent](#) ([OnPortSIPEvent](#) l)

## Protected Member Functions

- void [setAudioManagerEvents](#) (AppRTCAudioManager.AudioManagerEvents audioManagerEvents)

## Detailed Description

### Author:

PortSIP Solutions, Inc. All rights reserved.

### Version:

16

---

The documentation for this class was generated from the following file:

- PortSipSdk.java

## com.portsip.PortSIPVideoRenderer Class Reference

Inherits SurfaceViewRenderer.

### Classes

- enum `ScalingType`

### Public Member Functions

- [PortSIPVideoRenderer](#) (Context context)
- [PortSIPVideoRenderer](#) (Context context, AttributeSet attrs)
- void `setScalingType` (ScalingType scalingType)
- void `release` ()
- void `surfaceDestroyed` (SurfaceHolder holder)
- void `renderFrame` (VideoRenderer.I420Frame frame)

### Public Attributes

- long `nativeVideoRenderer` = 0
- 

### Detailed Description

Display the video stream on a SurfaceView.

---

### Constructor & Destructor Documentation

#### **com.portsip.PortSIPVideoRenderer.PortSIPVideoRenderer (Context context)**

Standard View constructor. In order to render something, you must first call `init()`.

#### **com.portsip.PortSIPVideoRenderer.PortSIPVideoRenderer (Context context, AttributeSet attrs)**

Standard View constructor. In order to render something, you must first call `init()`.

---

### Member Function Documentation

#### **void com.portsip.PortSIPVideoRenderer.release ()**

Block until any pending frame is returned and all GL resources released, even if an interrupt occurs. If an interrupt occurs during [release\(\)](#), the interrupt flag will be set. This function should be called before the Activity is destroyed and the EGLContext is still valid. If you don't call this function, the GL resources might leak.

---

The documentation for this class was generated from the following file:

- PortSIPVideoRenderer.java

# Index

- acceptRefer
  - Refer functions, 43
- Access SIP message header functions, 33
  - addSipMessageHeader, 33
  - clearAddedSipMessageHeaders, 34
  - clearModifiedSipMessageHeaders, 34
  - getSipMessageHeaderValue, 33
  - modifySipMessageHeader, 34
  - removeAddedSipMessageHeader, 33
  - removeModifiedSipMessageHeader, 34
- addAudioCodec
  - Audio and video codecs functions, 25
- Additional settings functions, 28
  - addSupportedMimeType, 32
  - disableCallForward, 30
  - disableSessionTimer, 31
  - enable3GppTags, 29
  - enableAutoCheckMwi, 31
  - enableCallbackSignaling, 29
  - enableCallForward, 30
  - enableEarlyMedia, 28
  - enableReliableProvisional, 29
  - enableRport, 28
  - enableSessionTimer, 31
  - getVersion, 28
  - setAudioSamples, 32
  - setDoNotDisturb, 31
  - setKeepAliveTime, 31
  - setRtcpPortRange, 30
  - setRtpKeepAlive, 31
  - setRtpPortRange, 30
  - setSrtpPolicy, 29
- addSipMessageHeader
  - Access SIP message header functions, 33
- addSupportedMimeType
  - Additional settings functions, 32
- addVideoCodec
  - Audio and video codecs functions, 26
- answerCall
  - Call functions, 39
- attendedRefer
  - Refer functions, 43
- attendedRefer2
  - Refer functions, 43
- Audio and video codecs functions, 25
  - addAudioCodec, 25
  - addVideoCodec, 26
  - clearAudioCodec, 27
  - clearVideoCodec, 27
  - isAudioCodecEmpty, 26
  - isVideoCodecEmpty, 26
  - setAudioCodecParameter, 27
  - setAudioCodecPayloadType, 26
  - setVideoCodecParameter, 27
  - setVideoCodecPayloadType, 26
- Audio and video functions, 35
  - displayLocalVideo, 37
  - enableVideoHardwareCodec, 35
  - getAudioDevices, 38
  - sendVideo, 37
  - setAudioBitrate, 36
  - setAudioDevice, 38
  - setChannelInputVolumeScaling, 38
  - setChannelOutputVolumeScaling, 37
  - setLocalVideoWindow, 37
  - setRemoteVideoWindow, 37
  - setVideoBitrate, 36
  - setVideoCropAndScale, 36
  - setVideoDeviceId, 35
  - setVideoFrameRate, 36
  - setVideoNackStatus, 37
  - setVideoResolution, 36
- Audio effect functions, 54
  - enableAEC, 54
  - enableAGC, 54
  - enableANS, 54
  - enableCNG, 54
  - enableVAD, 54
- audioPlayLoopbackTest
  - Play audio and video file to remote functions, 49
- call
  - Call functions, 39
- Call events, 9
  - onDialogStateUpdated, 11
  - onInviteAnswered, 10
  - onInviteBeginingForward, 11
  - onInviteClosed, 11
  - onInviteConnected, 11
  - onInviteFailure, 11
  - onInviteIncoming, 9
  - onInviteRinging, 10
  - onInviteSessionProgress, 10
  - onInviteTrying, 10
  - onInviteUpdated, 11
  - onRemoteHold, 12
  - onRemoteUnHold, 12
- Call functions, 38
  - answerCall, 39
  - call, 39
  - forwardCall, 41
  - hangUp, 39
  - hold, 40
  - muteSession, 40
  - pickupBLFCall, 41
  - rejectCall, 39
  - sendDtmf, 41
  - unHold, 40
  - updateCall, 40
- clearAddedSipMessageHeaders
  - Access SIP message header functions, 34
- clearAudioCodec

- Audio and video codecs functions, 27
- clearModifiedSipMessageHeaders
  - Access SIP message header functions, 34
- clearVideoCodec
  - Audio and video codecs functions, 27
- com.portsip.OnPortSIPEvent, 62
- com.portsip.PortSIPCameraCapturer, 64
- com.portsip.PortSipEnumDefine, 65
- com.portsip.PortSipEnumDefine.AUDIOCODEC, 60
- com.portsip.PortSipEnumDefine.AudioDevice, 61
- com.portsip.PortSipErrorcode, 68
- com.portsip.PortSipSdk, 71
- com.portsip.PortSIPVideoRenderer, 75
- com::portsip::PortSipEnumDefine
  - ENUM\_AUDIOSTREAM\_BOTH\_PER\_CHANNEL, 66
  - ENUM\_AUDIOSTREAM\_LOCAL\_PER\_CHANNEL, 66
  - ENUM\_AUDIOSTREAM\_REMOTE\_PER\_CHANNEL, 66
  - ENUM\_RECORD\_MODE\_BOTH, 66
  - ENUM\_RECORD\_MODE\_NONE, 66
  - ENUM\_RECORD\_MODE\_RECV, 66
  - ENUM\_RECORD\_MODE\_SEND, 66
  - ENUM\_VIDEOCODEC\_I420, 66
  - ENUM\_VIDEOCODEC\_NONE, 66
  - ENUM\_VIDEOSTREAM\_BOTH, 66
  - ENUM\_VIDEOSTREAM\_LOCAL, 66
  - ENUM\_VIDEOSTREAM\_NONE, 66
  - ENUM\_VIDEOSTREAM\_REMOTE, 66
- com::portsip::PortSIPVideoRenderer
  - PortSIPVideoRenderer, 75
  - release, 75
- Conference functions, 49
  - createAudioConference, 50
  - createVideoConference, 50
  - destroyConference, 50
  - joinToConference, 50
  - removeFromConference, 51
  - setConferenceVideoWindow, 50
- createAudioConference
  - Conference functions, 50
- CreateCallManager
  - Initialize and register functions, 22
- createVideoConference
  - Conference functions, 50
- DeleteCallManager
  - Initialize and register functions, 23
- destroyConference
  - Conference functions, 50
- disableCallForward
  - Additional settings functions, 30
- disableSessionTimer
  - Additional settings functions, 31
- displayLocalVideo
  - Audio and video functions, 37
- DTMF events, 15
  - onRecvDtmfTone, 15
- enable3GppTags
  - Additional settings functions, 29
- enableAEC
  - Audio effect functions, 54
- enableAGC
  - Audio effect functions, 54
- enableANS
  - Audio effect functions, 54
- enableAudioQos
  - RTP and RTCP QOS functions, 52
- enableAudioStreamCallback
  - RTP packets, Audio stream and video stream callback, 46
- enableAutoCheckMwi
  - Additional settings functions, 31
- enableCallbackSignaling
  - Additional settings functions, 29
- enableCallForward
  - Additional settings functions, 30
- enableCNG
  - Audio effect functions, 54
- enableEarlyMedia
  - Additional settings functions, 28
- enableReliableProvisional
  - Additional settings functions, 29
- enableRport
  - Additional settings functions, 28
- enableSendPcmStreamToRemote
  - Send audio and video stream functions, 44
- enableSendVideoStreamToRemote
  - Send audio and video stream functions, 45
- enableSessionTimer
  - Additional settings functions, 31
- enableVAD
  - Audio effect functions, 54
- enableVideoHardwareCodec
  - Audio and video functions, 35
- enableVideoQos
  - RTP and RTCP QOS functions, 52
- enableVideoStreamCallback
  - RTP packets, Audio stream and video stream callback, 46
- ENUM\_AUDIOSTREAM\_BOTH\_PER\_CHANNEL
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_AUDIOSTREAM\_LOCAL\_PER\_CHANNEL
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_AUDIOSTREAM\_REMOTE\_PER\_CHANNEL
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_RECORD\_MODE\_BOTH
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_RECORD\_MODE\_NONE
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_RECORD\_MODE\_RECV
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_RECORD\_MODE\_SEND
  - com::portsip::PortSipEnumDefine, 66

- com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDECODEC\_I420
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDECODEC\_NONE
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDESTREAM\_BOTH
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDESTREAM\_LOCAL
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDESTREAM\_NONE
  - com::portsip::PortSipEnumDefine, 66
- ENUM\_VIDESTREAM\_REMOTE
  - com::portsip::PortSipEnumDefine, 66
- forwardCall
  - Call functions, 41
- getAudioDevices
  - Audio and video functions, 38
- getAudioStatistics
  - RTP statistics functions, 53
- getSipMessageHeaderValue
  - Access SIP message header functions, 33
- getVersion
  - Additional settings functions, 28
- getVideoStatistics
  - RTP statistics functions, 53
- hangUp
  - Call functions, 39
- hold
  - Call functions, 40
- INFO/OPTIONS message events, 16
  - onRecvInfo, 16
  - onRecvNotifyOfSubscription, 16
  - onRecvOptions, 16
- initialize
  - Initialize and register functions, 23
- Initialize and register functions, 22
  - CreateCallManager, 22
  - DeleteCallManager, 23
  - initialize, 23
  - refreshRegistration, 24
  - registerServer, 24
  - setInstanceId, 23
  - setLicenseKey, 25
  - setUser, 24
  - unRegisterServer, 25
- isAudioCodecEmpty
  - Audio and video codecs functions, 26
- isVideoCodecEmpty
  - Audio and video codecs functions, 26
- joinToConference
  - Conference functions, 50
- modifySipMessageHeader
  - Access SIP message header functions, 34
- muteSession
  - Call functions, 40
- MWI events, 14
  - onWaitingFaxMessage, 15
  - onWaitingVoiceMessage, 14
- onACTVTransferFailure
  - Refer events, 13
- onACTVTransferSuccess
  - Refer events, 13
- onAudioRawCallback
  - RTP callback events, 21
- onDialogStateUpdated
  - Call events, 11
- onInviteAnswered
  - Call events, 10
- onInviteBeginingForward
  - Call events, 11
- onInviteClosed
  - Call events, 11
- onInviteConnected
  - Call events, 11
- onInviteFailure
  - Call events, 11
- onInviteIncoming
  - Call events, 9
- onInviteRinging
  - Call events, 10
- onInviteSessionProgress
  - Call events, 10
- onInviteTrying
  - Call events, 10
- onInviteUpdated
  - Call events, 11
- onPlayAudioFileFinished
  - Play audio and video file finished events, 19
- onPlayVideoFileFinished
  - Play audio and video file finished events, 20
- onPresenceOffline
  - Presence events, 17
- onPresenceOnline
  - Presence events, 17
- onPresenceRecvSubscribe
  - Presence events, 17
- onReceivedRefer
  - Refer events, 12
- onReceivedRTPPacket
  - RTP callback events, 20
- onReceivedSignaling
  - Signaling events, 14
- onRecvDtmfTone
  - DTMF events, 15
- onRecvInfo
  - INFO/OPTIONS message events, 16
- onRecvMessage
  - Presence events, 17
- onRecvNotifyOfSubscription
  - INFO/OPTIONS message events, 16
- onRecvOptions
  - INFO/OPTIONS message events, 16
- onRecvOutOfDialogMessage
  - Presence events, 18
- onReferAccepted
  - Refer events, 12
- onReferRejected
  - Refer events, 13

- onRegisterFailure
  - Register events, 8
- onRegisterSuccess
  - Register events, 8
- onRemoteHold
  - Call events, 12
- onRemoteUnHold
  - Call events, 12
- onSendingRTPPacket
  - RTP callback events, 20
- onSendingSignaling
  - Signaling events, 14
- onSendMessageFailure
  - Presence events, 18
- onSendMessageSuccess
  - Presence events, 18
- onSendOutOfDialogMessageFailure
  - Presence events, 19
- onSendOutOfDialogMessageSuccess
  - Presence events, 18
- onSubscriptionFailure
  - Presence events, 19
- onSubscriptionTerminated
  - Presence events, 19
- onTransferRinging
  - Refer events, 13
- onTransferTrying
  - Refer events, 13
- onVideoRawCallback
  - RTP callback events, 21
- onWaitingFaxMessage
  - MWI events, 15
- onWaitingVoiceMessage
  - MWI events, 14
- outOfDialogRefer
  - Refer functions, 43
- pickupBLFCall
  - Call functions, 41
- Play audio and video file finished events, 19
  - onPlayAudioFileFinished, 19
  - onPlayVideoFileFinished, 20
- Play audio and video file to remote functions, 48
  - audioPlayLoopbackTest, 49
  - playAudioFileToRemote, 48
  - playAudioFileToRemoteAsBackground, 49
  - playVideoFileToRemote, 48
  - stopPlayAudioFileToRemote, 49
  - stopPlayAudioFileToRemoteAsBackground, 49
  - stopPlayVideoFileToRemote, 48
- playAudioFileToRemote
  - Play audio and video file to remote functions, 48
- playAudioFileToRemoteAsBackground
  - Play audio and video file to remote functions, 49
- playVideoFileToRemote
  - Play audio and video file to remote functions, 48
- PortSIPVideoRenderer
  - com::portsip::PortSIPVideoRenderer, 75
- Presence events, 16
  - onPresenceOffline, 17
  - onPresenceOnline, 17
  - onPresenceRecvSubscribe, 17
  - onRecvMessage, 17
  - onRecvOutOfDialogMessage, 18
  - onSendMessageFailure, 18
  - onSendMessageSuccess, 18
  - onSendOutOfDialogMessageFailure, 19
  - onSendOutOfDialogMessageSuccess, 18
  - onSubscriptionFailure, 19
  - onSubscriptionTerminated, 19
- presenceAcceptSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 57
- presenceRejectSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 58
- presenceSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 57
- presenceTerminateSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 57
- Record functions, 47
  - startRecord, 47
  - stopRecord, 47
- refer
  - Refer functions, 42
- Refer events, 12
  - onACTVTransferFailure, 13
  - onACTVTransferSuccess, 13
  - onReceivedRefer, 12
  - onReferAccepted, 12
  - onReferRejected, 13
  - onTransferRinging, 13
  - onTransferTrying, 13
- Refer functions, 42
  - acceptRefer, 43
  - attendedRefer, 43
  - attendedRefer2, 43
  - outOfDialogRefer, 43
  - refer, 42
  - rejectRefer, 44
- refreshRegistration
  - Initialize and register functions, 24
- Register events, 8
  - onRegisterFailure, 8
  - onRegisterSuccess, 8
- registerServer
  - Initialize and register functions, 24
- rejectCall
  - Call functions, 39
- rejectRefer
  - Refer functions, 44
- release

- com::portsip::PortSIPVideoRenderer, 75
- removeAddedSipMessageHeader
  - Access SIP message header functions, 33
- removeFromConference
  - Conference functions, 51
- removeModifiedSipMessageHeader
  - Access SIP message header functions, 34
- RTP and RTCP QOS functions, 51
  - enableAudioQos, 52
  - enableVideoQos, 52
  - setAudioRtcpBandwidth, 51
  - setVideoMTU, 52
  - setVideoRtcpBandwidth, 51
- RTP callback events, 20
  - onAudioRawCallback, 21
  - onReceivedRTPPacket, 20
  - onSendingRTPPacket, 20
  - onVideoRawCallback, 21
- RTP packets, Audio stream and video stream callback, 46
  - enableAudioStreamCallback, 46
  - enableVideoStreamCallback, 46
  - setRtpCallback, 46
- RTP statistics functions, 52
  - getAudioStatistics, 53
  - getVideoStatistics, 53
- SDK Callback events, 8
- SDK functions, 21
- Send audio and video stream functions, 44
  - enableSendPcmStreamToRemote, 44
  - enableSendVideoStreamToRemote, 45
  - sendPcmStreamToRemote, 45
  - sendVideoStreamToRemote, 45
- Send OPTIONS/INFO/MESSAGE functions, 55
  - presenceAcceptSubscribe, 57
  - presenceRejectSubscribe, 58
  - presenceSubscribe, 57
  - presenceTerminateSubscribe, 57
  - sendInfo, 55
  - sendMessage, 56
  - sendOptions, 55
  - sendOutOfDialogMessage, 56
  - sendSubscription, 58
  - setDefaultPublicationTime, 57
  - setDefaultSubscriptionTime, 57
  - setPresenceMode, 56
  - setPresenceStatus, 58
  - terminateSubscription, 58
- sendDtmf
  - Call functions, 41
- sendInfo
  - Send OPTIONS/INFO/MESSAGE functions, 55
- sendMessage
  - Send OPTIONS/INFO/MESSAGE functions, 56
- sendOptions
  - Send OPTIONS/INFO/MESSAGE functions, 55
- sendOutOfDialogMessage
  - Send OPTIONS/INFO/MESSAGE functions, 56
- sendPcmStreamToRemote
  - Send audio and video stream functions, 45
- sendSubscription
  - Send OPTIONS/INFO/MESSAGE functions, 58
- sendVideo
  - Audio and video functions, 37
- sendVideoStreamToRemote
  - Send audio and video stream functions, 45
- setAudioBitrate
  - Audio and video functions, 36
- setAudioCodecParameter
  - Audio and video codecs functions, 27
- setAudioCodecPayloadType
  - Audio and video codecs functions, 26
- setAudioDevice
  - Audio and video functions, 38
- setAudioRtcpBandwidth
  - RTP and RTCP QOS functions, 51
- setAudioSamples
  - Additional settings functions, 32
- setChannelInputVolumeScaling
  - Audio and video functions, 38
- setChannelOutputVolumeScaling
  - Audio and video functions, 37
- setConferenceVideoWindow
  - Conference functions, 50
- setDefaultPublicationTime
  - Send OPTIONS/INFO/MESSAGE functions, 57
- setDefaultSubscriptionTime
  - Send OPTIONS/INFO/MESSAGE functions, 57
- setDoNotDisturb
  - Additional settings functions, 31
- setInstanceId
  - Initialize and register functions, 23
- setKeepAliveTime
  - Additional settings functions, 31
- setLicenseKey
  - Initialize and register functions, 25
- setLocalVideoWindow
  - Audio and video functions, 37
- setPresenceMode
  - Send OPTIONS/INFO/MESSAGE functions, 56
- setPresenceStatus
  - Send OPTIONS/INFO/MESSAGE functions, 58
- setRemoteVideoWindow
  - Audio and video functions, 37
- setRtcpPortRange
  - Additional settings functions, 30
- setRtpCallback

- RTP packets, Audio stream and video stream callback, 46
- setRtpKeepAlive
  - Additional settings functions, 31
- setRtpPortRange
  - Additional settings functions, 30
- setSrtpPolicy
  - Additional settings functions, 29
- setUser
  - Initialize and register functions, 24
- setVideoBitrate
  - Audio and video functions, 36
- setVideoCodecParameter
  - Audio and video codecs functions, 27
- setVideoCodecPayloadType
  - Audio and video codecs functions, 26
- setVideoCropAndScale
  - Audio and video functions, 36
- setVideoDeviceId
  - Audio and video functions, 35
- setVideoFrameRate
  - Audio and video functions, 36
- setVideoMTU
  - RTP and RTCP QOS functions, 52
- setVideoNackStatus
  - Audio and video functions, 37
- setVideoResolution
  - Audio and video functions, 36
- setVideoRtcpBandwidth
  - RTP and RTCP QOS functions, 51
- Signaling events, 13
  - onReceivedSignaling, 14
  - onSendingSignaling, 14
- startRecord
  - Record functions, 47
- stopPlayAudioFileToRemote
  - Play audio and video file to remote functions, 49
- stopPlayAudioFileToRemoteAsBackground
  - Play audio and video file to remote functions, 49
- stopPlayVideoFileToRemote
  - Play audio and video file to remote functions, 48
- stopRecord
  - Record functions, 47
- terminateSubscription
  - Send OPTIONS/INFO/MESSAGE functions, 58
- unHold
  - Call functions, 40
- unRegisterServer
  - Initialize and register functions, 25
- updateCall
  - Call functions, 40