

# **PortSIP VoIP SDK Manual for Android**

Version 19.4  
5/21/24

# Table of Contents

Welcome to PortSIP VoIP SDK For Android .....	2
Getting Started .....	2
Contents .....	2
SDK User Manual.....	2
Website .....	2
Support.....	2
Installation Prerequisites .....	2
Frequently Asked Questions .....	3
1. Where can I download the PortSIP VoIP SDK for test? .....	3
2. How can I compile the sample project?.....	3
3. How can I create a new project with PortSIP VoIP SDK? .....	3
4. How can I test the P2P call (without SIP server)? .....	3
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 .....	9
Call events.....	10
Refer events .....	14
Signaling events.....	16
MWI events.....	17
DTMF events .....	18
INFO/OPTIONS message events .....	19
Presence events .....	20
audio device changed,Play audio and video file finished events.....	23
RTP callback events.....	25
SDK functions.....	27
Initialize and register functions.....	28
Audio and video codecs functions .....	32
Additional settings functions .....	35
Access SIP message header functions .....	41
Audio and video functions .....	44
Call functions .....	48
Refer functions.....	52
Send audio and video stream functions .....	55
RTP packets, Audio stream and video stream callback.....	57
Record functions .....	58
Play audio and video file and RTMP/RTSP stream functions.....	60
Conference functions .....	62
RTP and RTCP QOS functions .....	64
RTP statistics functions .....	66
Audio effect functions.....	67
Send OPTIONS/INFO/MESSAGE functions.....	68
Class Documentation .....	73
com.portsip.PortSipEnumDefine.AUDIOCODEC.....	73
com.portsip.PortSipEnumDefine.AudioDevice.....	74
com.portsip.OnPortSIPEvent.....	75
com.portsip.PortSIPCameraCapturer.....	77
com.portsip.PortSipEnumDefine .....	78
com.portsip.PortSipErrorcode.....	80
com.portsip.PortSipSdk .....	83
com.portsip.PortSIPVideoRenderer.....	87
Index .....	88



# Welcome to PortSIP VoIP SDK For Android

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), Android Studio 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 VoIP 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 function.

## 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-16 is required.

# Frequently Asked Questions

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

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 Android studio:
4. Compile the sample project directly.

## 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 Android Studio 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", "onRTPPacketCallback" callbacks.

# Module Index

## Modules

Here is a list of all modules:

SDK Callback events.....	8
Register events.....	9
Call events .....	10
Refer events .....	14
Signaling events.....	16
MWI events .....	17
DTMF events.....	18
INFO/OPTIONS message events .....	19
Presence events.....	20
audio device changed,Play audio and video file finished events .....	23
RTP callback events .....	25
 SDK functions .....	27
Initialize and register functions .....	28
Audio and video codecs functions.....	32
Additional settings functions.....	35
Access SIP message header functions.....	41
Audio and video functions.....	44
Call functions.....	48
Refer functions .....	52
Send audio and video stream functions .....	55
RTP packets, Audio stream and video stream callback .....	57
Record functions.....	58
Play audio and video file and RTMP/RTSP stream functions .....	60
Conference functions.....	62
RTP and RTCP QOS functions .....	64
RTP statistics functions .....	66
Audio effect functions .....	67
Send OPTIONS/INFO/MESSAGE functions .....	68

# Hierarchical Index

## Class Hierarchy

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

com.portsip.PortSipEnumDefine.AUDIOCODEC .....	73
com.portsip.PortSipEnumDefine.AudioDevice.....	74
AppRTCAudioManager.AudioManagerEvents	
com.portsip.PortSipSdk .....	83
com.portsip.OnPortSIPEvent.....	75
com.portsip.PortSipEnumDefine .....	78
com.portsip.PortSipErrorcode .....	80
CapturerObserver	
com.portsip.PortSIPCameraCapturer .....	77
SurfaceViewRenderer	
com.portsip.PortSIPVideoRenderer .....	87



# Class Index

## Class List

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

<a href="#"><u>com.portsip.PortSipEnumDefine.AUDIOCODEC</u></a>	73
<a href="#"><u>com.portsip.PortSipEnumDefine.AudioDevice</u></a>	74
<a href="#"><u>com.portsip.OnPortSIPEvent</u></a>	75
<a href="#"><u>com.portsip.PortSIPCameraCapturer</u></a>	77
<a href="#"><u>com.portsip.PortSipEnumDefine</u></a>	78
<a href="#"><u>com.portsip.PortSipErrorcode</u></a>	80
<a href="#"><u>com.portsip.PortSipSdk</u></a>	83
<a href="#"><u>com.portsip.PortSIPVideoRenderer</u></a>	87

# Module Documentation

## SDK Callback events

### Modules

- [Register events](#)
  - [Call events](#)
  - [Refer events](#)
  - [Signaling events](#)
  - [MWI events](#)
  - [DTMF events](#)
  - [INFO/OPTIONS message events](#)
  - [Presence events](#)
  - [audio device changed, 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 callerDisplayName, String caller, String calleeDisplayName, String callee, String reason, int code, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteUpdated](#) (long sessionId, String audioCodecs, String videoCodecs, String screenCodecs, boolean existsAudio, boolean existsVideo, boolean existsScreen, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onInviteConnected](#) (long sessionId)
- void [com.portsip.OnPortSIPEvent.onInviteBeginingForward](#) (String forwardTo)
- void [com.portsip.OnPortSIPEvent.onInviteClosed](#) (long sessionId, String sipMessage)
- 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 *callerDisplayName*, String *caller*, String *calleeDisplayName*, String *callee*, String *reason*, int *code*, String *sipMessage*)**

This event will be triggered if the outgoing or incoming call fails.

**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>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*, String *screenCodecs*, boolean *existsAudio*, boolean *existsVideo*, boolean *existsScreen*, 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>screenCodecs</i>	The matched screen 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>existsScreen</i>	By setting to true, this call includes the screen shared.
<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*, String *sipMessage*)**

This event is triggered once remote side ends the call.

#### Parameters

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

**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 acceptRefer or rejectRefer
<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, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onSendMessageFailure](#) (long sessionId, long messageId, String reason, int code, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageSuccess](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String sipMessage)
- void [com.portsip.OnPortSIPEvent.onSendOutOfDialogMessageFailure](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String reason, int code, String sipMessage)
- 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*, String *sipMessage*)**

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.
<i>sipMessage</i>	The SIP message received.

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

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.
<i>sipMessage</i>	The SIP message received.

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

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.
<i>sipMessage</i>	The SIP message received.

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

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.
<i>sipMessage</i>	The SIP message received.

**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.
--------------------	------------------------------



## audio device changed, Play audio and video file finished events

### Functions

- void [com.portsip.OnPortSIPEvent.onPlayFileFinished](#) (long sessionId, String fileName)
- void [com.portsip.OnPortSIPEvent.onStatistics](#) (long sessionId, String statistics)
- void [com.portsip.OnPortSIPEvent.onAudioDeviceChanged](#) (PortSipEnumDefine.AudioDevice audioDevice, Set< PortSipEnumDefine.AudioDevice > devices)
- void [com.portsip.OnPortSIPEvent.onAudioFocusChange](#) (int focusChange)

---

### Detailed Description

---

### Function Documentation

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

If called startPlayingFileToRemote 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.onStatistics (long *sessionId*, String *statistics*)**

If called getStatistics function, this event will be triggered once the statistics get finished.

#### Parameters

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

**void com.portsip.OnPortSIPEvent.onAudioDeviceChanged (PortSipEnumDefine.AudioDevice *audioDevice*, Set< PortSipEnumDefine.AudioDevice > *devices*)**

fired When available audio devices changed or audio device currently in use changed.

#### Parameters

<i>audioDevice</i>	device currently in use
<i>devices</i>	devices useable. If a wired headset is connected, it should be the only possible option. When no wired headset connected, the devices set may contain speaker, earpiece, Bluetooth devices. can be set by <a href="#">PortSipSdk#setAudioDevice</a> to switch to current device.

**void com.portsip.OnPortSIPEvent.onAudioFocusChange (int *focusChange*)**

fired when the audio focus has been changed.

#### Parameters

<i>focusChange</i>	the type of focus change, one of AudioManager::AUDIOFOCUS_GAIN, AudioManager::AUDIOFOCUS_LOSS, AudioManager::AUDIOFOCUS_LOSS_TRANSIENT and AudioManager::AUDIOFOCUS_LOSS_TRANSIENT_CAN_DUCK.
--------------------	--



## RTP callback events

### Functions

- void [com.portsip.OnPortSIPEvent.onRTPPacketCallback](#) (long sessionId, int mediaType, int enum\_direction, byte[] RTPPacket, int packetSize)
- void [com.portsip.OnPortSIPEvent.onAudioRawCallback](#) (long sessionId, int enum\_direction, byte[] data, int dataLength, int samplingFreqHz)
- void [com.portsip.OnPortSIPEvent.onVideoRawCallback](#) (long sessionId, int enum\_direction, int width, int height, byte[] data, int dataLength)

---

### Detailed Description

---

### Function Documentation

**void com.portsip.OnPortSIPEvent.onRTPPacketCallback (long sessionId, int mediaType, int enum\_direction, byte[] RTPPacket, int packetSize)**

If enableRtpCallback function is called to enable the RTP callback, this event will be triggered once a RTP packet is received or sent.

#### Parameters

<i>sessionId</i>	The session ID of the call.
<i>mediaType</i>	RTP packet media type, 0 for audio, 1 for video , 2 for screen.
<i>enum_direction</i>	RTP packet direction <a href="#">ENUM_DIRECTION_SEND</a> , <a href="#">ENUM_DIRECTION_RECV</a> .
<i>RTPPacket</i>	The received or sent RTP packet.
<i>packetSize</i>	The size of the RTP packet in bytes.

#### Remarks

Donot 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\_direction, 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_direction</i>	The type passed in enableAudioStreamCallback function. Below types allowed: <a href="#">ENUM_DIRECTION_SEND</a> , <a href="#">ENUM_DIRECTION_RECV</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\_direction*, 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_direction</i>	The type which is passed in enableVideoStreamCallback function. Below types allowed: <a href="#">ENUM_DIRECTION_SEND</a> , <a href="#">ENUM_DIRECTION_RECV</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 and RTMP/RTSP stream 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

- 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)
  - void [com.portsip.PortSipSdk.unInitialize](#) ()
  - 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 waitMS)
  - int [com.portsip.PortSipSdk.setDisplayName](#) (String displayName)
  - int [com.portsip.PortSipSdk.setLicenseKey](#) (String key)

---

### Detailed Description

---

### Function Documentation

**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> ,
<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_datetime.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

### **void com.portsip.PortSipSdk.unInitialize ()**

Un-initialize the SDK and release resources.

### **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 outboundServer 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 outboundServer 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 <= 0, the attempt will be disabled and onRegisterFailure 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 (int *waitMS*)**

Un-register from the SIP proxy server.

#### Parameters

<i>waitMS</i>	Wait for the server to reply that the un-registration is successful, waitMS is the longest waiting milliseconds, 0 means not waiting.
---------------	---

#### 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.setDisplayName (String *displayName*)**

Set the display name of user.

#### Parameters

<i>displayName</i>	That will appear in the From/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.setLicenseKey (String *key*)**

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

**Parameters**

<i>key</i>	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

<i>enum_audiocodec</i>	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.enablePriorityIPv6Domain](#) (boolean enable)
- int [com.portsip.PortSipSdk.setUriUserEncoding](#) (String character, boolean enable)
- int [com.portsip.PortSipSdk.setReliableProvisional](#) (int mode)
- 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 minimumRtpPort, int maximumRtpPort)
- 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.enablePriorityIPv6Domain (boolean *enable*)**

Enable/disable which allows specifying the preferred protocol when a domain supports both IPV4 and IPV6 simultaneously.

#### Parameters

<i>enable</i>	Set to true to enable priority IPv6 Domain. with the default priority being IPV4.
---------------	---

#### 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.setUriUserEncoding (String *character*, boolean *enable*)**

Modifies the default URI user character needs to be escaped.

#### Parameters

<i>character</i>	The character to be modified, set one character at a time.
<i>enable</i>	Whether escaping is required, true for allowing escaping, false for disabling escaping.

#### 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.setReliableProvisional (int *mode*)**

Enable/Disable PRACK.

#### Parameters

<i>mode</i>	Modes work as follows: 0 - Never, Disable PRACK,By default the PRACK is disabled. 1 - SupportedEssential, Only send reliable provisionals if sending a body and far end supports. 2 - Supported, Always send reliable provisionals if far end supports. 3 - Required Always send reliable provisionals.
-------------	---

#### 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 *minimumRtpPort*, int *maximumRtpPort*)**

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

### Parameters

<i>minimumRtpPort</i>	The minimum RTP port.
<i>maximumRtpPort</i>	The maximum RTP port.

### 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="#">xxxx@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 *mime*Type, 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>mime</i> Type	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,"Blling", "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.setVideoOrientation](#) (int rotation)
- int [com.portsip.PortSipSdk.enableVideoHardwareCodec](#) (boolean enableHWEncoder, boolean enableHWDecoder)
- int [com.portsip.PortSipSdk.setVideoResolution](#) (int width, int height)
- 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)
- int [com.portsip.PortSipSdk.setRemoteVideoWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- int [com.portsip.PortSipSdk.setRemoteScreenWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- void [com.portsip.PortSipSdk.displayLocalVideo](#) (boolean state, boolean mirror, [PortSIPVideoRenderer](#) renderer)
- 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)
- void [com.portsip.PortSipSdk.enableAudioManager](#) (boolean state)
- 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.setVideoOrientation (int *rotation*)**

Setting the video Device Orientation.

##### **Parameters**

<i>rotation</i>	Device Orientation for video device (camera), e.g 0,90,180,270.
-----------------	---

##### **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 *enableHWDecoder*)**

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.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.

**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.

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

Set the window for a session that is used for displaying the received remote screen 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 screen 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, boolean mirror, [PortSIPVideoRenderer](#) renderer)**

Start/stop displaying the local video image.

**Parameters**

<i>state</i>	Set to true to display local video image.
<i>mirror</i>	Set to true to display the mirror image of local video.
<i>renderer</i>	<a href="#">SurfaceView</a> a SurfaceView for displaying local video image from camera.

**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.setChannelInputVolumeScaling (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.

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

enable/disable sdk audio manager,when enable sdk will auto manager audio device input/output. if the state is enabled, the onAudioDeviceChanged event will be triggered when available audio devices changed or audio device currently in use changed .

### Parameters

<i>state</i>	@true enable sdk audio manager @false disable audio manager
--------------	---

**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="#">user001@sip.iptel.org</a> or sip: <a href="#">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(VIDEODEC_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 cannot be seen.
<i>muteOutgoingVideo</i>	Set it to true to mute outgoing video stream, the remote side cannot see the 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.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: <code>DTMF_RFC2833</code> or <code>DTMF_INFO</code> . The <code>DTMF_RFC2833</code> 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](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](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

- long [com.portsip.PortSipSdk.enableRtpCallback](#) (long sessionId, int mediaType, int directionMode)
- void [com.portsip.PortSipSdk.enableAudioStreamCallback](#) (long sessionId, boolean enable, int enum\_direction)
- void [com.portsip.PortSipSdk.enableVideoStreamCallback](#) (long sessionId, int enum\_direction)

---

### Detailed Description

functions

---

### Function Documentation

**long com.portsip.PortSipSdk.enableRtpCallback (long sessionId, int mediaType, int directionMode)**

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

#### Parameters

<i>sessionId</i>	The session ID of the call.
<i>mediaType</i>	RTP packet media type, 0 for audio, 1 for video , 2 for screen.
<i>directionMode</i>	RTP packet direction, 0 for sending, 1 for receiving.

#### Returns

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

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

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_direction</i>	The audio stream callback mode. Supported modes include <a href="#">ENUM_DIRECTION_NONE</a> , <a href="#">ENUM_DIRECTION_SEND</a> , <a href="#">ENUM_DIRECTION_RECV</a> <a href="#">ENUM_DIRECTION_SEND_RECV</a> .

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

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_direction</i>	The video stream callback mode. Supported modes include <a href="#">ENUM_DIRECTION_NONE</a> , <a href="#">ENUM_DIRECTION_SEND</a> , <a href="#">ENUM_DIRECTION_RECV</a> , <a href="#">ENUM_DIRECTION_SEND_RECV</a> .

## Record functions

### Functions

- int [com.portsip.PortSipSdk.startRecord](#) (long sessionId, String recordFilePath, String recordFileName, boolean appendTimeStamp, int audioChannels, int enum\_fileFormat, int enum\_audioRecordMode, 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 audioChannels, int enum\_fileFormat, int enum\_audioRecordMode, 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>audioChannels</i>	Set to record file audio channels, 1 - mono 2 - stereo.
<i>enum_fileFormat</i>	The record file format, allow below values: ENUM_FILE_FORMAT_WAVE = 1, /// The record audio file is WAVE format. ENUM_FILE_FORMAT_AMR = 2, /// The record audio file is in AMR format with all voice data compressed by AMR codec. ENUM_FILE_FORMAT_MP3 = 3, /// The record audio file is in MP3 format. ENUM_FILE_FORMAT_MP4 = 4, /// The record video file is in MP4(AAC and H264) format.
<i>enum_audioRecordMode</i>	The audio record mode, allow below values: ENUM_RECORD_MODE_NONE = 0, /// Not Record. ENUM_RECORD_MODE_RECV = 1, /// Only record the received data. ENUM_RECORD_MODE_SEND, /// Only record send data. ENUM_RECORD_MODE_BOTH /// Record both received and sent data.
<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 and RTMP/RTSP stream functions

### Functions

- int [com.portsip.PortSipSdk.startPlayingFileToRemote](#) (long sessionId, String fileUrl, boolean loop, int playAudio)
- int [com.portsip.PortSipSdk.stopPlayingFileToRemote](#) (long sessionId)
- int [com.portsip.PortSipSdk.startPlayingFileLocally](#) (String fileUrl, boolean loop, [PortSIPVideoRenderer](#) renderer)
- int [com.portsip.PortSipSdk.stopPlayingFileLocally](#) ()
- void [com.portsip.PortSipSdk.audioPlayLoopbackTest](#) (boolean enable)

---

### Detailed Description

---

### Function Documentation

**int com.portsip.PortSipSdk.startPlayingFileToRemote (long sessionId, String fileUrl, boolean loop, int playAudio)**

Play an local file or RTSP/RTMP stream to remote party.

#### Parameters

<i>sessionId</i>	Session ID of the call.
<i>fileUrl</i>	The url or 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>	0 - Not play file audio. 1 - Play file audio. 2 - Play file audio mix with Mic.

#### 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.stopPlayingFileToRemote (long sessionId)**

Stop play 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.startPlayingFileLocally (String fileUrl, boolean loop, [PortSIPVideoRenderer](#) renderer)**

Play an local file or RTSP/RTMP stream.

#### Parameters

<i>fileUrl</i>	The url or 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>renderer</i>	<a href="#">SurfaceView</a> a SurfaceView for displaying the play image.

### 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.stopPlayingFileLocally ()**

Stop play file locally.

### 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, int layout)
- 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, int layout)**

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>layout</i>	Conference Video layout, default is 0 - Adaptive. 0 - Adaptive(1,3,5,6) 1 - Only Local Video 2 - 2 video, PIP 3 - 2 video, Left and right 4 - 2 video, Up and Down 5 - 3 video 6 - 4 split video 7 - 5 video

##### **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
------------------------------	--



<i>indow</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.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.getStatistics](#) (long sessionId)
- 

### Detailed Description

---

### Function Documentation

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

Obtain the statistics of channel. the event onStatistics will be triggered.

#### Parameters

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

#### Returns

If the function succeeds, it will return value is 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 mimeType, 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>mimeType</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 mimeType, 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>mimeType</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](#) list 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 callerDisplayName, String caller, String calleeDisplayName, String callee, String reason, int code, String sipMessage)
- void [onInviteUpdated](#) (long sessionId, String audioCodecs, String videoCodecs, String screenCodecs, boolean existsAudio, boolean existsVideo, boolean existsScreen, String sipMessage)
- void [onInviteConnected](#) (long sessionId)
- void [onInviteBeginingForward](#) (String forwardTo)
- void [onInviteClosed](#) (long sessionId, String sipMessage)
- 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, String sipMessage)

- void [onSendMessageFailure](#) (long sessionId, long messageId, String reason, int code, String sipMessage)
- void [onSendOutOfDialogMessageSuccess](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String sipMessage)
- void [onSendOutOfDialogMessageFailure](#) (long messageId, String fromDisplayName, String from, String toDisplayName, String to, String reason, int code, String sipMessage)
- void [onSubscriptionFailure](#) (long subscribeId, int statusCode)
- void [onSubscriptionTerminated](#) (long subscribeId)
- void [onPlayFileFinished](#) (long sessionId, String fileName)
- void [onStatistics](#) (long sessionId, String statistics)
- void [onAudioDeviceChanged](#) (PortSipEnumDefine.AudioDevice audioDevice, Set<PortSipEnumDefine.AudioDevice > devices)
- void [onAudioFocusChange](#) (int focusChange)
- void [onRTPPacketCallback](#) (long sessionId, int mediaType, int enum\_direction, byte[] RTPPacket, int packetSize)
- void [onAudioRawCallback](#) (long sessionId, int enum\_direction, byte[] data, int dataLength, int samplingFreqHz)
- void [onVideoRawCallback](#) (long sessionId, int enum\_direction, 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 [onFrameCaptured](#) (VideoFrame frame)

### Public Attributes

- CameraVideoCapturer **capturer**
- 

### Member Function Documentation

**void com.portsip.PortSIPCameraCapturer.onCapturerStarted (boolean *success*)**

Notify if the capturer have been started successfully or not.

**void com.portsip.PortSIPCameraCapturer.onCapturerStopped ()**

Notify that the capturer has been stopped.

**void com.portsip.PortSIPCameraCapturer.onFrameCaptured (VideoFrame *frame*)**

Delivers a captured frame.

---

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\\_SRTTPOLICY\\_NONE](#) = 0
- static final int [ENUM\\_SRTTPOLICY\\_FORCE](#) = 1
- static final int [ENUM\\_SRTTPOLICY\\_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\\_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\\_DIRECTION\\_NONE](#) = 0
- static final int [ENUM\\_DIRECTION\\_SEND\\_RECV](#) = 1
- static final int [ENUM\\_DIRECTION\\_SEND](#) = 2
- static final int [ENUM\\_DIRECTION\\_RECV](#) = 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\\_FILE\\_FORMAT\\_WAVE](#) = 1  
*The record audio file is in WAVE format.*
- static final int [ENUM\\_FILE\\_FORMAT\\_AMR](#) = 2  
*The record audio file is in AMR format - all voice data are compressed by AMR codec. Mono.*
- static final int [ENUM\\_FILE\\_FORMAT\\_MP3](#) = 3  
*The record audio file is in MP3 format.*



- static final int [ENUM\\_FILE\\_FORMAT\\_MP4](#) = 4  
*The record video file is in MP4(AAC and H264) format.*
- 

## 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\_DIRECTION\_SEND\_RECV = 1 [static]**

both received and sent

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

Only the sent

**final int com.portsip.PortSipEnumDefine.ENUM\_DIRECTION\_RECV = 3 [static]**

Only the received

**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 **ECoreInitializeWinsockFailure** = -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

Inherits AppRTCAudioManager.AudioManagerEvents.

### Classes

- class **MainHandler**

### Public Member Functions

- void **onAudioDeviceChanged** (AppRTCAudioManager.AudioDevice innerAdioDevice, Set<AppRTCAudioManager.AudioDevice > innerSet)
- void **onAudioFocusChange** (int focusChange)
- **PortSipSdk** (Context context)
- 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)
- void [unInitialize](#) ()
- 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 waitMS)
- int [setDisplayNames](#) (String displayName)
- 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 [enablePriorityIPv6Domain](#) (boolean enable)
- int [setUriUserEncoding](#) (String character, boolean enable)
- int [setReliableProvisional](#) (int mode)
- int [enable3GppTags](#) (boolean enable)
- void [enableCallbackSignaling](#) (boolean enableSending, boolean enableReceived)
- void [setSrtpPolicy](#) (int enum\_srtpolicy)
- int [setRtpPortRange](#) (int minimumRtpPort, int maximumRtpPort)
- 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 [setVideoOrientation](#) (int rotation)
- int [enableVideoHardwareCodec](#) (boolean enableHWEncoder, boolean enableHWDecoder)
- int [setVideoResolution](#) (int width, int height)
- 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)
- int [setRemoteVideoWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- int [setRemoteScreenWindow](#) (long sessionId, [PortSIPVideoRenderer](#) renderer)
- void [displayLocalVideo](#) (boolean state, boolean mirror, [PortSIPVideoRenderer](#) renderer)
- int [setVideoNackStatus](#) (boolean state)
- int [setChannelOutputVolumeScaling](#) (long sessionId, int scaling)
- int [setChannelInputVolumeScaling](#) (long sessionId, int scaling)
- void [enableAudioManager](#) (boolean state)
- 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)
- long [enableRtpCallback](#) (long sessionId, int mediaType, int directionMode)
- void [enableAudioStreamCallback](#) (long sessionId, boolean enable, int enum\_direction)
- void [enableVideoStreamCallback](#) (long sessionId, int enum\_direction)
- int [startRecord](#) (long sessionId, String recordFilePath, String recordFileName, boolean appendTimeStamp, int audioChannels, int enum\_fileFormat, int enum\_audioRecordMode, int enum\_videoRecordMode)

- int [stopRecord](#) (long sessionId)
- int [startPlayingFileToRemote](#) (long sessionId, String fileUrl, boolean loop, int playAudio)
- int [stopPlayingFileToRemote](#) (long sessionId)
- int [startPlayingFileLocally](#) (String fileUrl, boolean loop, [PortSIPVideoRenderer](#) renderer)
- int [stopPlayingFileLocally](#) ()
- void [audioPlayLoopbackTest](#) (boolean enable)
- int [createAudioConference](#) ()
- int [createVideoConference](#) ([PortSIPVideoRenderer](#) conferenceVideoWindow, int videoWidth, int videoHeight, int layout)
- 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 [getStatistics](#) (long sessionId)
- 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 [rtpPacketCallback](#) (long sessionId, int mediaType, int directionMode, 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 [finalize](#) ()

---

## Detailed Description

**Author**

PortSIP Solutions, Inc. All rights reserved.

**Version**

19

---

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 [setVideoRotation](#) (int rotation)
- void [release](#) ()
- void **surfaceDestroyed** (SurfaceHolder holder)

---

### 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.setVideoRotation (int *rotation*)**

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

#### **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, 53
- Access SIP message header functions, 41
  - addSipMessageHeader, 41
  - clearAddedSipMessageHeaders, 42
  - clearModifiedSipMessageHeaders, 43
  - getSipMessageHeaderValue, 41
  - modifySipMessageHeader, 42
  - removeAddedSipMessageHeader, 41
  - removeModifiedSipMessageHeader, 42
- addAudioCodec
  - Audio and video codecs functions, 32
- Additional settings functions, 35
  - addSupportedMimeType, 39
  - disableCallForward, 37
  - disableSessionTimer, 38
  - enable3GppTags, 36
  - enableAutoCheckMwi, 38
  - enableCallbackSignaling, 37
  - enableCallForward, 37
  - enableEarlyMedia, 35
  - enablePriorityIPv6Domain, 36
  - enableRport, 35
  - enableSessionTimer, 38
  - getVersion, 35
  - setAudioSamples, 39
  - setDoNotDisturb, 38
  - setKeepAliveTime, 39
  - setReliableProvisional, 36
  - setRtpKeepAlive, 38
  - setRtpPortRange, 37
  - setSrtpPolicy, 37
  - setUriUserEncoding, 36
- addSipMessageHeader
  - Access SIP message header functions, 41
- addSupportedMimeType
  - Additional settings functions, 39
- addVideoCodec
  - Audio and video codecs functions, 32
- answerCall
  - Call functions, 49
- attendedRefer
  - Refer functions, 52
- attendedRefer2
  - Refer functions, 53
- Audio and video codecs functions, 32
  - addAudioCodec, 32
  - addVideoCodec, 32
  - clearAudioCodec, 33
  - clearVideoCodec, 33
  - isAudioCodecEmpty, 32
  - isVideoCodecEmpty, 33
  - setAudioCodecParameter, 33
  - setAudioCodecPayloadType, 33
  - setVideoCodecParameter, 34
  - setVideoCodecPayloadType, 33
- Audio and video functions, 44
  - displayLocalVideo, 46
  - enableAudioManager, 47
  - enableVideoHardwareCodec, 44
  - getAudioDevices, 47
  - sendVideo, 46
  - setAudioBitrate, 45
  - setAudioDevice, 47
  - setChannelInputVolumeScaling, 47
  - setChannelOutputVolumeScaling, 46
  - setRemoteScreenWindow, 46
  - setRemoteVideoWindow, 46
  - setVideoBitrate, 45
  - setVideoDeviceId, 44
  - setVideoFrameRate, 45
  - setVideoNackStatus, 46
  - setVideoOrientation, 44
  - setVideoResolution, 45
- audio device changed, Play audio and video file finished events, 23
  - onAudioDeviceChanged, 23
  - onAudioFocusChange, 23
  - onPlayFileFinished, 23
  - onStatistics, 23
- Audio effect functions, 67
  - enableAEC, 67
  - enableAGC, 67
  - enableANS, 67
  - enableCNG, 67
  - enableVAD, 67
- audioPlayLoopbackTest
  - Play audio and video file and RTMP/RTSP stream functions, 61
- call
  - Call functions, 48
- Call events, 10
  - onDialogStateUpdated, 13
  - onInviteAnswered, 11
  - onInviteBeginingForward, 12
  - onInviteClosed, 12
  - onInviteConnected, 12
  - onInviteFailure, 12
  - onInviteIncoming, 10
  - onInviteRinging, 11
  - onInviteSessionProgress, 11
  - onInviteTrying, 11
  - onInviteUpdated, 12
  - onRemoteHold, 13
  - onRemoteUnHold, 13
- Call functions, 48
  - answerCall, 49
  - call, 48
  - forwardCall, 50
  - hangUp, 49
  - hold, 49
  - muteSession, 50

- pickupBLFCall, 50
- rejectCall, 48
- sendDtmf, 51
- unHold, 50
- updateCall, 49
- clearAddedSipMessageHeaders
  - Access SIP message header functions, 42
- clearAudioCodec
  - Audio and video codecs functions, 33
- clearModifiedSipMessageHeaders
  - Access SIP message header functions, 43
- clearVideoCodec
  - Audio and video codecs functions, 33
- com.portsip.OnPortSIPEvent, 75
- com.portsip.PortSIPCameraCapturer, 77
  - onCapturerStarted, 77
  - onCapturerStopped, 77
  - onFrameCaptured, 77
- com.portsip.PortSipEnumDefine, 78
  - ENUM\_DIRECTION\_RECV, 79
  - ENUM\_DIRECTION\_SEND, 79
  - ENUM\_DIRECTION\_SEND\_RECV, 79
  - ENUM\_RECORD\_MODE\_BOTH, 79
  - ENUM\_RECORD\_MODE\_NONE, 79
  - ENUM\_RECORD\_MODE\_RECV, 79
  - ENUM\_RECORD\_MODE\_SEND, 79
  - ENUM\_VIDEOCODEC\_I420, 79
  - ENUM\_VIDEOCODEC\_NONE, 79
- com.portsip.PortSipEnumDefine.AUDIOCOD  
EC, 73
- com.portsip.PortSipEnumDefine.AudioDevice,  
74
- com.portsip.PortSipErrorcode, 80
- com.portsip.PortSipSdk, 83
- com.portsip.PortSIPVideoRenderer, 87
  - PortSIPVideoRenderer, 87
  - release, 87
  - setVideoRotation, 87
- Conference functions, 62
  - createAudioConference, 62
  - createVideoConference, 62
  - destroyConference, 62
  - joinToConference, 63
  - removeFromConference, 63
  - setConferenceVideoWindow, 62
- createAudioConference
  - Conference functions, 62
- createVideoConference
  - Conference functions, 62
- destroyConference
  - Conference functions, 62
- disableCallForward
  - Additional settings functions, 37
- disableSessionTimer
  - Additional settings functions, 38
- displayLocalVideo
  - Audio and video functions, 46
- DTMF events, 18
  - onRecvDtmfTone, 18
- enable3GppTags
  - Additional settings functions, 36
- enableAEC
  - Audio effect functions, 67
- enableAGC
  - Audio effect functions, 67
- enableANS
  - Audio effect functions, 67
- enableAudioManager
  - Audio and video functions, 47
- enableAudioQos
  - RTP and RTCP QOS functions, 64
- enableAudioStreamCallback
  - RTP packets, Audio stream and video  
stream callback, 57
- enableAutoCheckMwi
  - Additional settings functions, 38
- enableCallbackSignaling
  - Additional settings functions, 37
- enableCallForward
  - Additional settings functions, 37
- enableCNG
  - Audio effect functions, 67
- enableEarlyMedia
  - Additional settings functions, 35
- enablePriorityIPv6Domain
  - Additional settings functions, 36
- enableRport
  - Additional settings functions, 35
- enableRtpCallback
  - RTP packets, Audio stream and video  
stream callback, 57
- enableSendPcmStreamToRemote
  - Send audio and video stream functions,  
55
- enableSendVideoStreamToRemote
  - Send audio and video stream functions,  
56
- enableSessionTimer
  - Additional settings functions, 38
- enableVAD
  - Audio effect functions, 67
- enableVideoHardwareCodec
  - Audio and video functions, 44
- enableVideoQos
  - RTP and RTCP QOS functions, 65
- enableVideoStreamCallback
  - RTP packets, Audio stream and video  
stream callback, 57
- ENUM\_DIRECTION\_RECV
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_DIRECTION\_SEND
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_DIRECTION\_SEND\_RECV
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_RECORD\_MODE\_BOTH
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_RECORD\_MODE\_NONE
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_RECORD\_MODE\_RECV
  - com.portsip.PortSipEnumDefine, 79

- ENUM\_RECORD\_MODE\_SEND
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_VIDEOCODEC\_I420
  - com.portsip.PortSipEnumDefine, 79
- ENUM\_VIDEOCODEC\_NONE
  - com.portsip.PortSipEnumDefine, 79
- forwardCall
  - Call functions, 50
- getAudioDevices
  - Audio and video functions, 47
- getSipMessageHeaderValue
  - Access SIP message header functions, 41
- getStatistics
  - RTP statistics functions, 66
- getVersion
  - Additional settings functions, 35
- hangUp
  - Call functions, 49
- hold
  - Call functions, 49
- INFO/OPTIONS message events, 19
  - onRecvInfo, 19
  - onRecvNotifyOfSubscription, 19
  - onRecvOptions, 19
- initialize
  - Initialize and register functions, 28
- Initialize and register functions, 28
  - initialize, 28
  - refreshRegistration, 30
  - registerServer, 30
  - setDisplayname, 30
  - setInstanceId, 29
  - setLicenseKey, 31
  - setUser, 29
  - unInitialize, 29
  - unRegisterServer, 30
- isAudioCodecEmpty
  - Audio and video codecs functions, 32
- isVideoCodecEmpty
  - Audio and video codecs functions, 33
- joinToConference
  - Conference functions, 63
- modifySipMessageHeader
  - Access SIP message header functions, 42
- muteSession
  - Call functions, 50
- MWI events, 17
  - onWaitingFaxMessage, 17
  - onWaitingVoiceMessage, 17
- onACTVTransferFailure
  - Refer events, 15
- onACTVTransferSuccess
  - Refer events, 15
- onAudioDeviceChanged
  - audio device changed, Play audio and video file finished events, 23
- onAudioFocusChange
  - audio device changed, Play audio and video file finished events, 23
- onAudioRawCallback
  - RTP callback events, 25
- onCapturerStarted
  - com.portsip.PortSIPCameraCapturer, 77
- onCapturerStopped
  - com.portsip.PortSIPCameraCapturer, 77
- onDialogStateUpdated
  - Call events, 13
- onFrameCaptured
  - com.portsip.PortSIPCameraCapturer, 77
- onInviteAnswered
  - Call events, 11
- onInviteBeginingForward
  - Call events, 12
- onInviteClosed
  - Call events, 12
- onInviteConnected
  - Call events, 12
- onInviteFailure
  - Call events, 12
- onInviteIncoming
  - Call events, 10
- onInviteRinging
  - Call events, 11
- onInviteSessionProgress
  - Call events, 11
- onInviteTrying
  - Call events, 11
- onInviteUpdated
  - Call events, 12
- onPlayFileFinished
  - audio device changed, Play audio and video file finished events, 23
- onPresenceOffline
  - Presence events, 20
- onPresenceOnline
  - Presence events, 20
- onPresenceRecvSubscribe
  - Presence events, 20
- onReceivedRefer
  - Refer events, 14
- onReceivedSignaling
  - Signaling events, 16
- onRecvDtmfTone
  - DTMF events, 18
- onRecvInfo
  - INFO/OPTIONS message events, 19
- onRecvMessage
  - Presence events, 21
- onRecvNotifyOfSubscription
  - INFO/OPTIONS message events, 19
- onRecvOptions
  - INFO/OPTIONS message events, 19
- onRecvOutOfDialogMessage
  - Presence events, 21
- onReferAccepted
  - Refer events, 14
- onReferRejected
  - Refer events, 14
- onRegisterFailure
  - Register events, 9

- onRegisterSuccess
  - Register events, 9
- onRemoteHold
  - Call events, 13
- onRemoteUnHold
  - Call events, 13
- onRTPPacketCallback
  - RTP callback events, 25
- onSendingSignaling
  - Signaling events, 16
- onSendMessageFailure
  - Presence events, 21
- onSendMessageSuccess
  - Presence events, 21
- onSendOutOfDialogMessageFailure
  - Presence events, 22
- onSendOutOfDialogMessageSuccess
  - Presence events, 22
- onStatistics
  - audio device changed, Play audio and video file finished events, 23
- onSubscriptionFailure
  - Presence events, 22
- onSubscriptionTerminated
  - Presence events, 22
- onTransferRinging
  - Refer events, 15
- onTransferTrying
  - Refer events, 14
- onVideoRawCallback
  - RTP callback events, 26
- onWaitingFaxMessage
  - MWI events, 17
- onWaitingVoiceMessage
  - MWI events, 17
- outOfDialogRefer
  - Refer functions, 53
- pickupBLFCall
  - Call functions, 50
- Play audio and video file and RTMP/RTSP stream functions, 60
  - audioPlayLoopbackTest, 61
  - startPlayingFileLocally, 60
  - startPlayingFileToRemote, 60
  - stopPlayingFileLocally, 61
  - stopPlayingFileToRemote, 60
- PortSIPVideoRenderer
  - com.portsip.PortSIPVideoRenderer, 87
- Presence events, 20
  - onPresenceOffline, 20
  - onPresenceOnline, 20
  - onPresenceRecvSubscribe, 20
  - onRecvMessage, 21
  - onRecvOutOfDialogMessage, 21
  - onSendMessageFailure, 21
  - onSendMessageSuccess, 21
  - onSendOutOfDialogMessageFailure, 22
  - onSendOutOfDialogMessageSuccess, 22
  - onSubscriptionFailure, 22
  - onSubscriptionTerminated, 22
- presenceAcceptSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 70
- presenceRejectSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 71
- presenceSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 70
- presenceTerminateSubscribe
  - Send OPTIONS/INFO/MESSAGE functions, 70
- Record functions, 58
  - startRecord, 58
  - stopRecord, 58
- refer
  - Refer functions, 52
- Refer events, 14
  - onACTVTransferFailure, 15
  - onACTVTransferSuccess, 15
  - onReceivedRefer, 14
  - onReferAccepted, 14
  - onReferRejected, 14
  - onTransferRinging, 15
  - onTransferTrying, 14
- Refer functions, 52
  - acceptRefer, 53
  - attendedRefer, 52
  - attendedRefer2, 53
  - outOfDialogRefer, 53
  - refer, 52
  - rejectRefer, 53
- refreshRegistration
  - Initialize and register functions, 30
- Register events, 9
  - onRegisterFailure, 9
  - onRegisterSuccess, 9
- registerServer
  - Initialize and register functions, 30
- rejectCall
  - Call functions, 48
- rejectRefer
  - Refer functions, 53
- release
  - com.portsip.PortSIPVideoRenderer, 87
- removeAddedSipMessageHeader
  - Access SIP message header functions, 41
- removeFromConference
  - Conference functions, 63
- removeModifiedSipMessageHeader
  - Access SIP message header functions, 42
- RTP and RTCP QOS functions, 64
  - enableAudioQos, 64
  - enableVideoQos, 65
  - setAudioRtcpBandwidth, 64
  - setVideoMTU, 65
  - setVideoRtcpBandwidth, 64
- RTP callback events, 25
  - onAudioRawCallback, 25
  - onRTPPacketCallback, 25

- onVideoRawCallback, 26
- RTP packets, Audio stream and video stream callback, 57
  - enableAudioStreamCallback, 57
  - enableRtpCallback, 57
  - enableVideoStreamCallback, 57
- RTP statistics functions, 66
  - getStatistics, 66
- SDK Callback events, 8
- SDK functions, 27
- Send audio and video stream functions, 55
  - enableSendPcmStreamToRemote, 55
  - enableSendVideoStreamToRemote, 56
  - sendPcmStreamToRemote, 55
  - sendVideoStreamToRemote, 56
- Send OPTIONS/INFO/MESSAGE functions, 68
  - presenceAcceptSubscribe, 70
  - presenceRejectSubscribe, 71
  - presenceSubscribe, 70
  - presenceTerminateSubscribe, 70
  - sendInfo, 68
  - sendMessage, 69
  - sendOptions, 68
  - sendOutOfDialogMessage, 69
  - sendSubscription, 71
  - setDefaultPublicationTime, 70
  - setDefaultSubscriptionTime, 70
  - setPresenceMode, 69
  - setPresenceStatus, 71
  - terminateSubscription, 71
- sendDtmf
  - Call functions, 51
- sendInfo
  - Send OPTIONS/INFO/MESSAGE functions, 68
- sendMessage
  - Send OPTIONS/INFO/MESSAGE functions, 69
- sendOptions
  - Send OPTIONS/INFO/MESSAGE functions, 68
- sendOutOfDialogMessage
  - Send OPTIONS/INFO/MESSAGE functions, 69
- sendPcmStreamToRemote
  - Send audio and video stream functions, 55
- sendSubscription
  - Send OPTIONS/INFO/MESSAGE functions, 71
- sendVideo
  - Audio and video functions, 46
- sendVideoStreamToRemote
  - Send audio and video stream functions, 56
- setAudioBitrate
  - Audio and video functions, 45
- setAudioCodecParameter
  - Audio and video codecs functions, 33
- setAudioCodecPayloadType
  - Audio and video codecs functions, 33
- setAudioDevice
  - Audio and video functions, 47
- setAudioRtcpBandwidth
  - RTP and RTCP QOS functions, 64
- setAudioSamples
  - Additional settings functions, 39
- setChannelInputVolumeScaling
  - Audio and video functions, 47
- setChannelOutputVolumeScaling
  - Audio and video functions, 46
- setConferenceVideoWindow
  - Conference functions, 62
- setDefaultPublicationTime
  - Send OPTIONS/INFO/MESSAGE functions, 70
- setDefaultSubscriptionTime
  - Send OPTIONS/INFO/MESSAGE functions, 70
- setDisplayName
  - Initialize and register functions, 30
- setDoNotDisturb
  - Additional settings functions, 38
- setInstanceId
  - Initialize and register functions, 29
- setKeepAliveTime
  - Additional settings functions, 39
- setLicenseKey
  - Initialize and register functions, 31
- setPresenceMode
  - Send OPTIONS/INFO/MESSAGE functions, 69
- setPresenceStatus
  - Send OPTIONS/INFO/MESSAGE functions, 71
- setReliableProvisional
  - Additional settings functions, 36
- setRemoteScreenWindow
  - Audio and video functions, 46
- setRemoteVideoWindow
  - Audio and video functions, 46
- setRtpKeepAlive
  - Additional settings functions, 38
- setRtpPortRange
  - Additional settings functions, 37
- setSrtpPolicy
  - Additional settings functions, 37
- setUriUserEncoding
  - Additional settings functions, 36
- setUser
  - Initialize and register functions, 29
- setVideoBitrate
  - Audio and video functions, 45
- setVideoCodecParameter
  - Audio and video codecs functions, 34
- setVideoCodecPayloadType
  - Audio and video codecs functions, 33
- setVideoDeviceId
  - Audio and video functions, 44

- setVideoFrameRate
  - Audio and video functions, 45
- setVideoMTU
  - RTP and RTCP QOS functions, 65
- setVideoNackStatus
  - Audio and video functions, 46
- setVideoOrientation
  - Audio and video functions, 44
- setVideoResolution
  - Audio and video functions, 45
- setVideoRotation
  - com.portsip.PortSIPVideoRenderer, 87
- setVideoRtcpBandwidth
  - RTP and RTCP QOS functions, 64
- Signaling events, 16
  - onReceivedSignaling, 16
  - onSendingSignaling, 16
- startPlayingFileLocally
  - Play audio and video file and RTMP/RTSP stream functions, 60
- startPlayingFileToRemote
  - Play audio and video file and RTMP/RTSP stream functions, 60
- startRecord
  - Record functions, 58
- stopPlayingFileLocally
  - Play audio and video file and RTMP/RTSP stream functions, 61
- stopPlayingFileToRemote
  - Play audio and video file and RTMP/RTSP stream functions, 60
- stopRecord
  - Record functions, 58
- terminateSubscription
  - Send OPTIONS/INFO/MESSAGE functions, 71
- unHold
  - Call functions, 50
- unInitialize
  - Initialize and register functions, 29
- unRegisterServer
  - Initialize and register functions, 30
- updateCall
  - Call functions, 49