

# PortSIP VoIP SDK Manual for iOS

Version 11.3  
Tue Nov 15 2016

# Table of Contents

Welcome to PortSIP VoIP SDK .....	4
Getting Started .....	4
Contents .....	4
SDK User Manual .....	4
Website .....	4
Support .....	4
Installation Prerequisites .....	4
Apple's iOS SDK .....	4
Note: .....	5
Device Requirements .....	5
Frequently Asked Questions .....	5
1. Where can I download the PortSIP VoIP SDK for test? .....	5
2. How can I compile the sample project? .....	5
3. How can I create a new project with PortSIP VoIP SDK? .....	5
4. How can I test the P2P call (without SIP server)? .....	6
5. Is the SDK thread safe? .....	6
6. Does the SDK support native 64-bit? .....	6
Module Index .....	7
Hierarchical Index .....	8
Class Index .....	9
Module Documentation .....	10
SDK Callback events .....	10
Register events .....	10
Call events .....	11
Refer events .....	14
Signaling events .....	16
MWI events .....	16
DTMF events .....	17
INFO/OPTIONS message events .....	18
Presence events .....	19
MESSAGE message events .....	20
Play audio and video file finished events .....	22
RTP callback events .....	22
Audio and video stream callback events .....	23
SDK functions .....	25
Initialize and register functions .....	25
NIC and local IP functions .....	29
Audio and video codecs functions .....	29
Additional settings functions .....	32
Access SIP message header functions .....	39
Audio and video functions .....	41
Call functions .....	46
Refer functions .....	50
Send audio and video stream functions .....	52
RTP packets, audio stream and video stream callback functions .....	54
Record functions .....	56
Play audio and video files to remote party .....	57
Conference functions .....	59
RTP and RTCP QOS functions .....	61
RTP statistics functions .....	63
Audio effect functions .....	65
Send OPTIONS/INFO/MESSAGE functions .....	66

Presence functions .....	68
Keep awake functions.....	70
Class Documentation.....	72
<PortSIPEventDelegate >.....	72
PortSIPSDK.....	74
PortSIPVideoRenderView.....	81
Index.....	83

# Welcome to PortSIP VoIP SDK

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

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

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

## Getting Started

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

## Contents

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

## SDK User Manual

To be started with, it is recommended to read the documentation of PortSIP VoIP SDK, [SDK User Manual page](#), which gives a brief description of each API 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 aren't 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

Development using the PortSIP VoIP/IMS SDK for iOS requires an Intel-based Macintosh running Snow Leopard (OS X 10.8 or higher)

## Apple's iOS SDK

If you are not yet a registered Apple developer, to be able to develop applications for the iOS, you do need to become a registered Apple developer. After registered, Apple grants you free access to a select set of technical resources, tools, and information for developing with iOS, Mac OS X, and Safari. You can open [registration page](#) and enroll.

Once registered, you can then go to the [iOS Dev Center](#), login and download the iOS SDK. The SDK contains documentation, frameworks, tools, and a simulator to help develop iOS applications. XCode (the developer toolset for iOS application development) is included in the download as well so you do not need to purchase any developer tools to build iOS applications - that is included in the enrollment fee. You will need to use a minimum of iOS SDK 10 for developing iPhone and iPod Touch applications. At the time of writing this document, iOS SDK 10 was the most recent version available and supported.

#### **Note:**

Beta and GM seed versions of the iOS SDK are generally not supported unless noted otherwise. Regardless of the iOS SDK you're using for development, you can still target your application for devices running on an older iOS version by configuring your Xcode project's iOS Deployment Target build settings. Be sure to add runtime checks where appropriate to ensure that you use only those iOS features available on the target platform/device. If your application attempts to use iOS features which are not available on the device, your application may crash.

#### **Device Requirements**

Applications built with PortSIP VoIP/IMS SDK for iOS can be run on iPhone 4S or higher, iPod touch 4 or higher, and iPad 2 or higher devices. These devices must be running iOS8 or higher. We strongly recommend that you test your applications on actual devices to ensure that they work as expected and perform well. Testing on the simulator alone does not provide a good measure of how the application will perform on the physical device.

#### **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:  
<http://www.PortSIP.com/downloads>  
<http://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 with your Xcode:
4. Compile the sample project directly. The trial version SDK allows you a 2-3 minutes conversation.

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

1. Download the Sample project and evaluation SDK and extract it to a directory.
2. Run the Xcode and create a new iOS Project.
3. Drag and drop PortSIPLib.framework from Finder to XCode->Frameworks.
4. Add depend Frameworks:  
Build Phases->Link Binary With Libraries, add libstdc++.6.tbd, libresolv.tbd, VideoToolbox.framework.
5. Add the code in .h file to import the SDK, example:  
`#import <PortSIPLib/PortSIPSDK.h>`

6. Inherit the interface `PortSIPEventDelegate` to process the callback events. For example:

```
@interface AppDelegate : UIResponder <UIApplicationDelegate,PortSIPEventDelegate>{
    PortSIPSDK* mPortSIPSDK;
}
@end
```

7. Initialize sdk. For example:

```
mPortSIPSDK = [[PortSIPSDK alloc] init];
mPortSIPSDK.delegate = self;
```

8. For more details, please read 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 in 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. 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.

4. The log box will appear "Initialized" if the SDK is successfully initialized.

5. To make call from A to B, enter "sip:222@192.168.1.11" and click "Dial" button; while to make call from B to A, enter "sip:111@192.168.1.10".

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

#### 5. Is the SDK thread safe?

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

#### 6. Does the SDK support native 64-bit?

Yes, both 32-bit and 64-bit are supported for SDK.

# Module Index

## Modules

Here is a list of all modules:

SDK Callback events .....	10
Register events .....	10
Call events .....	11
Refer events .....	14
Signaling events .....	16
MWI events .....	16
DTMF events .....	17
INFO/OPTIONS message events .....	18
Presence events .....	19
MESSAGE message events .....	20
Play audio and video file finished events .....	22
RTP callback events .....	22
Audio and video stream callback events .....	23
SDK functions .....	25
Initialize and register functions .....	25
NIC and local IP functions .....	29
Audio and video codecs functions .....	29
Additional settings functions .....	32
Access SIP message header functions .....	39
Audio and video functions .....	41
Call functions .....	46
Refer functions .....	50
Send audio and video stream functions .....	52
RTP packets, audio stream and video stream callback functions .....	54
Record functions .....	56
Play audio and video files to remote party .....	57
Conference functions .....	59
RTP and RTCP QOS functions .....	61
RTP statistics functions .....	63
Audio effect functions .....	65
Send OPTIONS/INFO/MESSAGE functions .....	66
Presence functions .....	68
Keep awake functions .....	70

# Hierarchical Index

## Class Hierarchy

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

NSObject	
PortSIPSDK .....	74
<NSObject>	
<PortSIPEventDelegate > .....	72
UIView	
PortSIPVideoRenderView .....	81



# Class Index

## Class List

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

<a href="#"><u>&lt;PortSIPEventDelegate&gt;</u></a> (PortSIP SDK Callback events Delegate ) .....	72
<a href="#"><u>PortSIPSDK</u></a> (PortSIP VoIP SDK functions class ) .....	74
<a href="#"><u>PortSIPVideoRenderView</u></a> (PortSIP VoIP SDK Video Render View class ) .....	81

# Module Documentation

## SDK Callback events

### Modules

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

### Detailed Description

SDK Callback events

## Register events

### Functions

- (void) - [<PortSIPEventDelegate >::onRegisterSuccess:statusCode:](#)
  - (void) - [<PortSIPEventDelegate >::onRegisterFailure:statusCode:](#)
- 

### Detailed Description

Register events

---

## Function Documentation

- (void PortSIPEventDelegate) onRegisterSuccess: (char \*) *statusText* statusCode: (int) *statusCode*

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

#### Parameters:

<i>statusText</i>	The status text.
<i>statusCode</i>	The status code.

- (void PortSIPEventDelegate) onRegisterFailure: (char \*) *statusText* statusCode: (int) *statusCode*

If registration to SIP server is fails, this event will be triggered.

**Parameters:**

<i>statusText</i>	The status text.
<i>statusCode</i>	The status code.

## Call events

### Functions

- (void) - [<PortSIPEventDelegate >::onInviteIncoming:callerDisplayName:caller:calleeDisplayName:callee:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [<PortSIPEventDelegate >::onInviteTrying:](#)
- (void) - [<PortSIPEventDelegate >::onInviteSessionProgress:audioCodecs:videoCodecs:existsEarlyMedia:existsAudio:existsVideo:](#)
- (void) - [<PortSIPEventDelegate >::onInviteRinging:statusText:statusCode:](#)
- (void) - [<PortSIPEventDelegate >::onInviteAnswered:callerDisplayName:caller:calleeDisplayName:callee:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [<PortSIPEventDelegate >::onInviteFailure:reason:code:](#)
- (void) - [<PortSIPEventDelegate >::onInviteUpdated:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [<PortSIPEventDelegate >::onInviteConnected:](#)
- (void) - [<PortSIPEventDelegate >::onInviteBeginingForward:](#)
- (void) - [<PortSIPEventDelegate >::onInviteClosed:](#)
- (void) - [<PortSIPEventDelegate >::onRemoteHold:](#)
- (void) - [<PortSIPEventDelegate >::onRemoteUnHold:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)

---

## Detailed Description

---

## Function Documentation

- (void PortSIPEventDelegate) onInviteIncoming: (long) *sessionId* callerDisplayName: (char \*) *callerDisplayName* caller: (char \*) *caller* calleeDisplayName: (char \*) *calleeDisplayName* callee: (char \*) *callee* audioCodecs: (char \*) *audioCodecs* videoCodecs: (char \*) *videoCodecs* existsAudio: (BOOL) *existsAudio* existsVideo: (BOOL) *existsVideo*

When the 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 indicates that this call includes the audio.
<i>existsVideo</i>	By setting to true, it indicates that this call includes the video.

- (void PortSIPEventDelegate) onInviteTrying: (long) *sessionId*[optional]

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

Parameters:

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

- (void PortSIPEventDelegate) onInviteSessionProgress: (long) *sessionId* audioCodecs: (char \*) *audioCodecs* videoCodecs: (char \*) *videoCodecs* existsEarlyMedia: (BOOL) *existsEarlyMedia* existsAudio: (BOOL) *existsAudio* existsVideo: (BOOL) *existsVideo*[optional]

Once the caller received the "183 session in progress" message, 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>existsEarlyMedia</i>	By setting to true, it indicates that the call has early media.
<i>existsAudio</i>	By setting to true, it indicates that this call includes the audio.
<i>existsVideo</i>	By setting to true, it indicates that this call includes the video.

- (void PortSIPEventDelegate) onInviteRinging: (long) *sessionId* statusText: (char \*) *statusText* statusCode: (int) *statusCode*[optional]

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.

- (void PortSIPEventDelegate) onInviteAnswered: (long) *sessionId* callerDisplayName: (char \*) *callerDisplayName* caller: (char \*) *caller* calleeDisplayName: (char \*) *calleeDisplayName* callee: (char \*) *callee* audioCodecs: (char \*) *audioCodecs* videoCodecs: (char \*) *videoCodecs* existsAudio: (BOOL) *existsAudio* existsVideo: (BOOL) *existsVideo*[optional]

If the remote party is answering the call, 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 indicates that this call includes the audio.
<i>existsVideo</i>	By setting to true, it indicates that this call includes the video.

- (void PortSIPEventDelegate) onInviteFailure: (long) *sessionId* reason: (char \*) *reason* code: (int) *code*[optional]

If the outgoing call fails, this event will be triggered.

**Parameters:**

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

- (void PortSIPEventDelegate) onInviteUpdated: (long) *sessionId* audioCodecs: (char \*) *audioCodecs* videoCodecs: (char \*) *videoCodecs* existsAudio: (BOOL) *existsAudio* existsVideo: (BOOL) *existsVideo*[optional]

This event will be triggered when remote party updates this 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 have more than one codec.
<i>existsAudio</i>	By setting to true, it indicates that this call includes the audio.
<i>existsVideo</i>	By setting to true, it indicates that this call includes the video.

- (void PortSIPEventDelegate) onInviteConnected: (long) *sessionId*[optional]

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

**Parameters:**

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

- (void PortSIPEventDelegate) onInviteBeginingForward: (char \*) *forwardTo*[optional]

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 for forwarding.
------------------	------------------------------------

- (void PortSIPEventDelegate) onInviteClosed: (long) *sessionId*[optional]

This event will be triggered once remote side closes the call.

**Parameters:**

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

- (void PortSIPEventDelegate) onRemoteHold: (long) *sessionId*[optional]

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

**Parameters:**

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

- (void PortSIPEventDelegate) onRemoteUnHold: (long) *sessionId* audioCodecs: (char \*) *audioCodecs* videoCodecs: (char \*) *videoCodecs* existsAudio: (BOOL) *existsAudio* existsVideo: (BOOL) *existsVideo*[optional]

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 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 indicates that this call includes the audio.
<i>existsVideo</i>	By setting to true, it indicates that this call includes the video.

## Refer events

### Functions

- (void) - [<PortSIPEventDelegate >::onReceivedRefer:referId:to:from:referSipMessage:](#)
- (void) - [<PortSIPEventDelegate >::onReferAccepted:](#)
- (void) - [<PortSIPEventDelegate >::onReferRejected:reason:code:](#)
- (void) - [<PortSIPEventDelegate >::onTransferTrying:](#)
- (void) - [<PortSIPEventDelegate >::onTransferRinging:](#)
- (void) - [<PortSIPEventDelegate >::onACTVTransferSuccess:](#)
- (void) - [<PortSIPEventDelegate >::onACTVTransferFailure:reason:code:](#)

---

## Detailed Description

---

## Function Documentation

- (void PortSIPEventDelegate) onReceivedRefer: (long) *sessionId* referId: (long) *referId* to: (char \*) *to* from: (char \*) *from* referSipMessage: (char \*) *referSipMessage*

This event will be triggered once receiving 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 PortSIPEventDelegate) 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 PortSIPEventDelegate) onReferRejected: (long) *sessionId* reason: (char \*)  
reason code: (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>	Reason for rejecting.
<i>code</i>	Rejecting code.

**- (void PortSIPEventDelegate) 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 PortSIPEventDelegate) 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 PortSIPEventDelegate) onACTVTransferSuccess: (long) *sessionId***

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

**Parameters:**

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

**- (void PortSIPEventDelegate) onACTVTransferFailure: (long) *sessionId* reason: (char \*)  
reason code: (int) *code***

When the refer call fails, this event will be triggered. ACTV means Active. For example: A starts the call with B, and A transfers B to C. When C rejects the referred call, 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) - [<PortSIPEventDelegate >::onReceivedSignaling:message:](#)
  - (void) - [<PortSIPEventDelegate >::onSendingSignaling:message:](#)
- 

### Detailed Description

---

### Function Documentation

- (void PortSIPEventDelegate) onReceivedSignaling: (long) *sessionId* message: (char \*) *message*

This event will be triggered when receiving a SIP message.

**Parameters:**

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

- (void PortSIPEventDelegate) onSendingSignaling: (long) *sessionId* message: (char \*) *message*

This event will be triggered when an SIP message sent.

**Parameters:**

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

## MWI events

### Functions

- (void) - [<PortSIPEventDelegate >::onWaitingVoiceMessage:urgentNewMessageCount:urgentOldMessageCount:newMessageCount:oldMessageCount:](#)
  - (void) - [<PortSIPEventDelegate >::onWaitingFaxMessage:urgentNewMessageCount:urgentOldMessageCount:newMessageCount:oldMessageCount:](#)
- 

### Detailed Description

---



## Function Documentation

- (void PortSIPEventDelegate) onWaitingVoiceMessage: (char \*) *messageAccount* urgentNewMessageCount: (int) *urgentNewMessageCount* urgentOldMessageCount: (int) *urgentOldMessageCount* newMessageCount: (int) *newMessageCount* oldMessageCount: (int) *oldMessageCount*

If there are any waiting voice messages (MWI), this event will be triggered.

### Parameters:

<i>messageAccount</i>	Account for voice message.
<i>urgentNewMessageCount</i>	Count of new urgent messages.
<i>urgentOldMessageCount</i>	Count of old urgent messages.
<i>newMessageCount</i>	Count of new messages.
<i>oldMessageCount</i>	Count of old messages.

- (void PortSIPEventDelegate) onWaitingFaxMessage: (char \*) *messageAccount* urgentNewMessageCount: (int) *urgentNewMessageCount* urgentOldMessageCount: (int) *urgentOldMessageCount* newMessageCount: (int) *newMessageCount* oldMessageCount: (int) *oldMessageCount*

If there are any the waiting fax messages (MWI), this event will be triggered.

### Parameters:

<i>messageAccount</i>	Account for fax message.
<i>urgentNewMessageCount</i>	Count of new urgent messages.
<i>urgentOldMessageCount</i>	Count of old urgent messages.
<i>newMessageCount</i>	Count of new messages.
<i>oldMessageCount</i>	Count of old messages.

## DTMF events

### Functions

- (void) - [<PortSIPEventDelegate >::onRecvDtmfTone:tone:](#)

---

## Detailed Description

---

## Function Documentation

- (void PortSIPEventDelegate) onRecvDtmfTone: (long) *sessionId* tone: (int) *tone*

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

**Parameters:**

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

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) - [<PortSIPEventDelegate >::onRecvOptions:](#)
- (void) - [<PortSIPEventDelegate >::onRecvInfo:](#)

---

### Detailed Description

---

### Function Documentation

**- (void PortSIPEventDelegate) onRecvOptions: (char \*) optionsMessage**

This event will be triggered when receiving the OPTIONS message.

**Parameters:**

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

**- (void PortSIPEventDelegate) onRecvInfo: (char \*) infoMessage**

This event will be triggered when receiving the INFO message.

**Parameters:**

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

## Presence events

### Functions

- (void) - [<PortSIPEventDelegate >::onPresenceRecvSubscribe:fromDisplayName:from:subject:](#)
- (void) - [<PortSIPEventDelegate >::onPresenceOnline:from:stateText:](#)
- (void) - [<PortSIPEventDelegate >::onPresenceOffline:from:](#)

---

### Detailed Description

---

### Function Documentation

- (void PortSIPEventDelegate) onPresenceRecvSubscribe: (long) *subscribeId* fromDisplayName: (char \*) *fromDisplayName* from: (char \*) *from* subject: (char \*) *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 PortSIPEventDelegate) onPresenceOnline: (char \*) *fromDisplayName* from: (char \*) *from* stateText: (char \*) *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 PortSIPEventDelegate) onPresenceOffline: (char \*) *fromDisplayName* from: (char \*) *from*

When the contact status changes to 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

## MESSAGE message events

### Functions

- (void) - [<PortSIPEventDelegate >::onRecvMessage:mimeType:subMimeType:messageData:messageDataLength:](#)
- (void) - [<PortSIPEventDelegate >::onRecvOutOfDialogMessage:from:toDisplayName:to:mimeType:subMimeType:messageData:messageDataLength:](#)
- (void) - [<PortSIPEventDelegate >::onSendMessageSuccess:messageId:](#)
- (void) - [<PortSIPEventDelegate >::onSendMessageFailure:messageId:reason:code:](#)
- (void) - [<PortSIPEventDelegate >::onSendOutOfDialogMessageSuccess:fromDisplayName:from:toDisplayName:to:](#)
- (void) - [<PortSIPEventDelegate >::onSendOutOfDialogMessageFailure:fromDisplayName:from:toDisplayName:to:reason:code:](#)

---

### Detailed Description

---

### Function Documentation

- (void PortSIPEventDelegate) onRecvMessage: (long) *sessionId* mimeType: (char \*) *mimeType* subMimeType: (char \*) *subMimeType* messageData: (unsigned char \*) *messageData* messageDataLength: (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's can be either text or binary data.
<i>messageDataLength</i>	The length of "messageData".

- (void PortSIPEventDelegate) onRecvOutOfDialogMessage: (char \*) *fromDisplayName* from: (char \*) *from* toDisplayName: (char \*) *toDisplayName* to: (char \*) *to* mimeType: (char \*) *mimeType* subMimeType: (char \*) *subMimeType* messageData: (unsigned char \*) *messageData* messageDataLength: (int) *messageDataLength*

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>contentType</i>	The message mime type.
<i>subMimeType</i>	The message sub mime type.
<i>messageData</i>	The received message body. It's can be text or binary data.
<i>messageDataLength</i>	The length of "messageData".

- (void PortSIPEventDelegate) onSendMessageSuccess: (long) *sessionId* *messageId*: (long) *messageId*

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

Parameters:

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

- (void PortSIPEventDelegate) onSendMessageFailure: (long) *sessionId* *messageId*: (long) *messageId* *reason*: (char \*) *reason code*: (int) *code*

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

Parameters:

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

- (void PortSIPEventDelegate) onSendOutOfDialogMessageSuccess: (long) *messageId* *fromDisplayName*: (char \*) *fromDisplayName* *from*: (char \*) *from* *toDisplayName*: (char \*) *toDisplayName* *to*: (char \*) *to*

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

Parameters:

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

- (void PortSIPEventDelegate) onSendOutOfDialogMessageFailure: (long) *messageId* *fromDisplayName*: (char \*) *fromDisplayName* *from*: (char \*) *from* *toDisplayName*: (char \*) *toDisplayName* *to*: (char \*) *to* *reason*: (char \*) *reason code*: (int) *code*

If the message fails to be sent 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>reason</i>	The failure reason.
<i>code</i>	The failure code.

## Play audio and video file finished events

### Functions

- (void) - [<PortSIPEventDelegate >::onPlayAudioFileFinished:fileName:](#)
  - (void) - [<PortSIPEventDelegate >::onPlayVideoFileFinished:](#)
- 

### Detailed Description

---

### Function Documentation

- (void PortSIPEventDelegate) onPlayAudioFileFinished: (long) *sessionId* fileName: (char \*) *fileName*

If playAudioFileToRemote function is called 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 PortSIPEventDelegate) onPlayVideoFileFinished: (long) *sessionId*

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

**Parameters:**

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

## RTP callback events

### Functions

- (void) - [<PortSIPEventDelegate >::onReceivedRTPPacket:isAudio:RTPPacket:packetSize:](#)
  - (void) - [<PortSIPEventDelegate >::onSendingRTPPacket:isAudio:RTPPacket:packetSize:](#)
- 

### Detailed Description

---

## Function Documentation

- (void PortSIPEventDelegate) onReceivedRTPPacket: (long) *sessionId* isAudio: (BOOL) *isAudio* RTPPacket: (unsigned char \*) *RTPPacket* packetSize: (int) *packetSize*

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

### Parameters:

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

### Note:

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

- (void PortSIPEventDelegate) onSendingRTPPacket: (long) *sessionId* isAudio: (BOOL) *isAudio* RTPPacket: (unsigned char \*) *RTPPacket* packetSize: (int) *packetSize*

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

### Parameters:

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

### Note:

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.

## Audio and video stream callback events

### Functions

- (void) - [<PortSIPEventDelegate >::onAudioRawCallback:audioCallbackMode:data:dataLength:samplingFreq Hz:](#)
- (void) - [<PortSIPEventDelegate >::onVideoRawCallback:videoCallbackMode:width:height:data:dataLength:](#)
- (void) - [<PortSIPEventDelegate >::onVideoDecoderCallback:width:height:framerate:bitrate:](#)

---

## Detailed Description

---

## Function Documentation

- (void PortSIPEventDelegate) onAudioRawCallback: (long) *sessionId* audioCallbackMode: (int) *audioCallbackMode* data: (unsigned char \*) *data* dataLength: (int) *dataLength* samplingFreqHz: (int) *samplingFreqHz*

This event will be triggered once receiving the audio packets when enableAudioStreamCallback function is called.

### Parameters:

<i>sessionId</i>	The session ID of the call.
<i>audioCallbackMode</i>	The type that is passed in enableAudioStreamCallback function.
<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, it could be 8000 or 16000.

### Note:

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

- (void PortSIPEventDelegate) onVideoRawCallback: (long) *sessionId* videoCallbackMode: (int) *videoCallbackMode* width: (int) *width* height: (int) *height* data: (unsigned char \*) *data* dataLength: (int) *dataLength*

This event will be triggered once received the video packets if called enableVideoStreamCallback function.

### Parameters:

<i>sessionId</i>	The session ID of the call.
<i>videoCallbackMode</i>	The type which is passed in enableVideoStreamCallback function.
<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, such as YV12.
<i>dataLength</i>	The data size.

### Note:

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

- (void PortSIPEventDelegate) onVideoDecoderCallback: (long) *sessionId* width: (int) *width* height: (int) *height* framerate: (int) *framerate* bitrate: (int) *bitrate*

This event will be triggered once per second containing the frame rate and bit rate for the incoming stream or new incoming Video size is detected if called enableVideoDecoderCallback function.

### Parameters:

<i>sessionId</i>	The session ID of the call.
<i>width</i>	The width of video image.
<i>height</i>	The height of video image.
<i>framerate</i>	The frame rate of video
<i>bitrate</i>	The bitrate of video codec



**Note:**

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.

## SDK functions

### Modules

- [Initialize and register functions](#)
- [NIC and local IP 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 functions](#)
- [Record functions](#)
- [Play audio and video files to remote party](#)
- [Conference functions](#)
- [RTP and RTCP QOS functions](#)
- [RTP statistics functions](#)
- [Audio effect functions](#)
- [Send OPTIONS/INFO/MESSAGE functions](#)
- [Presence functions](#)
- [Keep awake functions](#)

---

### Detailed Description

SDK functions

## Initialize and register functions

### Functions

- (int) - [PortSIPSDK::initialize:loglevel:logPath:maxLine:agent:audioDeviceLayer:videoDeviceLayer:](#)  
*Initialize the SDK.*
- (int) - [PortSIPSDK::setInstanceId:](#)  
*Set the instance Id, the outbound instanceId((RFC5626) ) used in contact headers.*
- (void) - [PortSIPSDK::unInitialize](#)  
*Un-initialize the SDK and release resources.*
- (int) - [PortSIPSDK::setUser:displayName:authName:password:localIP:localSIPPort:userDomain:SIPServer:SIPServerPort:STUNServer:STUNServerPort:outboundServer:outboundServerPort:](#)  
*Set user account info.*
- (int) - [PortSIPSDK::registerServer:retryTimes:](#)  
*Register to SIP proxy server (login to server)*

- (int) - [PortSIPSDK::refreshRegisterServer:](#)  
*Request a manual refresh of the registration.*
- (int) - [PortSIPSDK::unRegisterServer](#)  
*Un-register from the SIP proxy server.*
- (int) - [PortSIPSDK::setLicenseKey:](#)  
*Set the license key. It must be called before setUser function.*

## Detailed Description

Initialize and register functions

## Function Documentation

- (int) initialize: (TRANSPORT\_TYPE) *transport* loglevel: (PORTSIP\_LOG\_LEVEL) *logLevel* logPath: (NSString \*) *logFilePath* maxLine: (int) *maxCallLines* agent: (NSString \*) *sipAgent* audioDeviceLayer: (int) *audioDeviceLayer* videoDeviceLayer: (int) *videoDeviceLayer*

Initialize the SDK.

### Parameters:

<i>transport</i>	Transport for SIP signaling. TRANSPORT_PERS is the PortSIP private transport for anti the SIP blocking, it must be used with PERS.
<i>logLevel</i>	Set the application log level. SDK will generate "PortSIP_Log_datatime.log" file if the log is enabled.
<i>logFilePath</i>	The log file path. The path (folder) MUST be existent.
<i>maxCallLines</i>	Theoretically unlimited lines are supported depending on the device capability. For SIP client the recommend value ranges from 1 - 100;
<i>sipAgent</i>	The User-Agent header to be inserted in SIP messages.
<i>audioDeviceLayer</i>	Specifies the audio device layer that should be used: 0 = Use the OS default device. 1 = Virtual device, usually use this for the device which has no sound device installed.
<i>videoDeviceLayer</i>	Specifies the video device layer that should be used: 0 = Use the OS default device. 1 = Use Virtual device. Usually use this for the device which has no camera installed.

### Returns:

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

- (int) setInstanceId: (NSString \*) *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) setUser: (NSString \*) *userName* displayName: (NSString \*) *displayName* authName: (NSString \*) *authName* password: (NSString \*) *password* localIP: (NSString \*) *localIP* localSIPPort: (int) *localSIPPort* userDomain: (NSString \*) *userDomain* SIPServer: (NSString \*) *sipServer* SIPServerPort: (int) *sipServerPort* STUNServer: (NSString \*) *stunServer* STUNServerPort: (int) *stunServerPort* outboundServer: (NSString \*) *outboundServer* outboundServerPort: (int) *outboundServerPort*

Set user account info.

**Parameters:**

<i>userName</i>	Account (username) of the SIP. It's usually provided by an IP-Telephony service provider.
<i>displayName</i>	The display name of user. 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>	The password of user. It's optional.
<i>localIP</i>	The local computer IP address to be bound (for example: 192.168.1.108). It will be used for sending and receiving SIP message and RTP packet. If this IP is passed as the IPv6 format the SDK will use IPv6. If you want the SDK to choose correct network interface (IP) automatically, please pass the "0.0.0.0" (for IPv4) or "::" (for IPv6).
<i>localSIPPort</i>	The SIP message transport listener port (for example: 5060).
<i>userDomain</i>	User domain. This parameter is optional. It allows to pass an empty string if you are not using 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, used for NAT traversal. It's optional and can pass an 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 pass an empty string if not using outbound server.
<i>outboundServerPort</i>	Outbound proxy server port. It will be ignored if the outboundServer is empty.

**Returns:**

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

- (int) registerServer: (int) *expires* retryTimes: (int) *retryTimes*

Register to SIP proxy server (login to server)

**Parameters:**

<i>expires</i>	Registration refreshment interval in seconds. Maximum of 3600 supported. It will be inserted into SIP REGISTER message header.
<i>retryTimes</i>	The retry times if failed to refresh the registration. Once set to <= 0, the retry will be disabled and onRegisterFailure callback triggered for retry failure.

**Returns:**

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

**- (int) refreshRegisterServer: (int) expires**

Request a manual refresh of the registration.

**Parameters:**

<i>expires</i>	Registration refreshment interval in seconds. Maximum of 3600 supported. It will be inserted into SIP REGISTER message header. If it's set to 0, default value will be used.
----------------	--

**Returns:**

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

**- (int) unRegisterServer**

Un-register from the SIP proxy server.

**Returns:**

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

**- (int) setLicenseKey: (NSString \*) 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.

## NIC and local IP functions

### Functions

- (int) - [PortSIPSDK::getNICNums](#)  
*Get the Network Interface Card numbers.*
- (NSString \*) - [PortSIPSDK::getLocalIpAddress:](#)  
*Get the local IP address by Network Interface Card index.*

---

### Detailed Description

---

### Function Documentation

#### - (int) getNICNums

Get the Network Interface Card numbers.

#### Returns:

If the function succeeds, it will return value NIC numbers that is greater than or equal to 0. If the function fails, it will return a specific error code.

#### - (NSString\*) getLocalIpAddress: (int) *index*

Get the local IP address by Network Interface Card index.

#### Parameters:

<i>index</i>	The IP address index. For example, the PC has two NICs, and we want to obtain the second NIC IP, then set this parameter 1 and the first NIC IP index is 0.
--------------	---

#### Returns:

The buffer for receiving the IP.

## Audio and video codecs functions

### Functions

- (int) - [PortSIPSDK::addAudioCodec:](#)  
*Enable an audio codec. It will appear in SDP.*
- (int) - [PortSIPSDK::addVideoCodec:](#)  
*Enable a video codec, and it will appear in SDP.*

- (BOOL) - [PortSIPSDK::isAudioCodecEmpty](#)  
*Detect if the enabled audio codecs is empty.*
- (BOOL) - [PortSIPSDK::isVideoCodecEmpty](#)  
*Detect enabled video codecs is empty or not.*
- (int) - [PortSIPSDK::setAudioCodecPayloadType:payloadType:](#)  
*Set the RTP payload type for dynamic audio codec.*
- (int) - [PortSIPSDK::setVideoCodecPayloadType:payloadType:](#)  
*Set the RTP payload type for dynamic Video codec.*
- (void) - [PortSIPSDK::clearAudioCodec](#)  
*Remove all enabled audio codecs.*
- (void) - [PortSIPSDK::clearVideoCodec](#)  
*Remove all enabled video codecs.*
- (int) - [PortSIPSDK::setAudioCodecParameter:parameter:](#)  
*Set the codec parameter for audio codec.*
- (int) - [PortSIPSDK::setVideoCodecParameter:parameter:](#)  
*Set the codec parameter for video codec.*

## Detailed Description

## Function Documentation

### - (int) addAudioCodec: (AUDIOCODEC\_TYPE) *codecType*

Enable an audio codec. It will appear in SDP.

#### Parameters:

<i>codecType</i>	Audio codec type.
------------------	-------------------

#### Returns:

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

### - (int) addVideoCodec: (VIDEOCODEC\_TYPE) *codecType*

Enable a video codec, and it will appear in SDP.

#### Parameters:

<i>codecType</i>	Video codec type.
------------------	-------------------

#### Returns:

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

### - (BOOL) isAudioCodecEmpty

Detect if the enabled audio codecs is empty.

#### Returns:

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

### - (BOOL) isVideoCodecEmpty

Detect enabled video codecs is empty or not.

#### Returns:

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

### - (int) setAudioCodecPayloadType: (AUDIOCODEC\_TYPE) codecType payloadType: (int) payloadType

Set the RTP payload type for dynamic audio codec.

#### Parameters:

<i>codecType</i>	Audio codec type defined in the PortSIPTypes file.
<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) setVideoCodecPayloadType: (VIDEOCODEC\_TYPE) codecType payloadType: (int) payloadType

Set the RTP payload type for dynamic Video codec.

#### Parameters:

<i>codecType</i>	Video codec type defined in the PortSIPTypes file.
<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) setAudioCodecParameter: (AUDIOCODEC\_TYPE) codecType parameter: (NSString \*) parameter

Set the codec parameter for audio codec.

**Parameters:**

<i>codecType</i>	Audio codec type defined in the PortSIPTypes file.
<i>parameter</i>	The parameter 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:

```
[myVoIPsdk setAudioCodecParameter:AUDIOCODEC_AMR parameter:"mode-set=0;
octet-align=1; robust-sorting=0"];
```

- (int) **setVideoCodecParameter: (VIDEOCODEC\_TYPE) codecType parameter: (NSString \*) parameter**

Set the codec parameter for video codec.

**Parameters:**

<i>codecType</i>	Video codec type defined in the PortSIPTypes file.
<i>parameter</i>	The parameter in string format.

**Returns:**

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

**Remarks:**

Example:

```
[myVoIPsdk setVideoCodecParameter:VIDEOCODEC_H264
parameter:"profile-level-id=420033; packetization-mode=0"];
```

## Additional settings functions

### Functions

- (int) - [PortSIPSDK::setDisplayname:](#)  
*Set user display name.*
- (int) - [PortSIPSDK::getVersion:minorVersion:](#)  
*Get the current version number of the SDK.*
- (int) - [PortSIPSDK::enableReliableProvisional:](#)  
*Enable/disable PRACK.*
- (int) - [PortSIPSDK::enable3GppTags:](#)  
*Enable/disable the 3Gpp tags, including "ims.icsi.mmtel" and "g.3gpp.smsip".*
- (void) - [PortSIPSDK::enableCallbackSendingSignaling:](#)  
*Enable/disable to callback the sent SIP messages.*
- (int) - [PortSIPSDK::setSrtpPolicy:](#)  
*Set the SRTP policy.*



- (int) - [PortSIPSDK::setRtpPortRange:maximumRtpAudioPort:minimumRtpVideoPort:maximumRtpVideoPort:](#)  
Set the RTP ports range for audio and video streaming.
- (int) - [PortSIPSDK::setRtcpPortRange:maximumRtcpAudioPort:minimumRtcpVideoPort:maximumRtcpVideoPort:](#)  
Set the RTCP ports range for audio and video streaming.
- (int) - [PortSIPSDK::enableCallForward:forwardTo:](#)  
Enable call forward.
- (int) - [PortSIPSDK::disableCallForward](#)  
Disable the call forwarding. The SDK is not forwarding any incoming calls once this function is called.
- (int) - [PortSIPSDK::enableSessionTimer:refreshMode:](#)  
Allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending INVITE requests repeatedly.
- (int) - [PortSIPSDK::disableSessionTimer](#)  
Disable the session timer.
- (void) - [PortSIPSDK::setDoNotDisturb:](#)  
Enable the "Do not disturb" to enable/disable.
- (int) - [PortSIPSDK::detectMwi](#)  
Used to obtain the MWI status.
- (int) - [PortSIPSDK::enableCheckMwi:](#)  
Allows to enable/disable the check MWI (Message Waiting Indication).
- (int) - [PortSIPSDK::setRtpKeepAlive:keepAlivePayloadType:deltaTransmitTimeMS:](#)  
Enable or disable to send RTP keep-alive packet when the call is established.
- (int) - [PortSIPSDK::setKeepAliveTime:](#)  
Enable or disable to send SIP keep-alive packet.
- (int) - [PortSIPSDK::setAudioSamples:maxPtime:](#)  
Set the audio capturing sample.
- (int) - [PortSIPSDK::addSupportedMimeType:mimeType:subMimeType:](#)  
Set the SDK to receive the SIP message that includes special mime type.

## Detailed Description

## Function Documentation

- (int) **setDisplayname:** (NSString \*) *displayName*

Set user display name.

### Parameters:

<i>displayName</i>	The display name.
--------------------	-------------------

**Returns:**

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

**- (int) getVersion: (int \*) majorVersion minorVersion: (int \*) minorVersion**

Get the current version number of the SDK.

**Parameters:**

<i>majorVersion</i>	Return the major version number.
<i>minorVersion</i>	Return the minor version number.

**Returns:**

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

**- (int) enableReliableProvisional: (BOOL) enable**

Enable/disable PRACK.

**Parameters:**

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

**Returns:**

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

**- (int) enable3GppTags: (BOOL) enable**

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

**Parameters:**

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

**Returns:**

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

**- (void) enableCallbackSendingSignaling: (BOOL) enable**

Enable/disable to callback the sent SIP messages.

**Parameters:**

<i>enable</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.
---------------	--

**- (int) setSrtpPolicy: (SRTP\_POLICY) *srtpPolicy***

Set the SRTP policy.

**Parameters:**

<i>srtpPolicy</i>	The SRTP policy.
-------------------	------------------

**Returns:**

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

**- (int) setRtpPortRange: (int) *minimumRtpAudioPort* maximumRtpAudioPort: (int) *maximumRtpAudioPort* minimumRtpVideoPort: (int) *minimumRtpVideoPort* maximumRtpVideoPort: (int) *maximumRtpVideoPort***

Set the RTP ports range for audio and video streaming.

**Parameters:**

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

**Returns:**

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

**Remarks:**

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

**- (int) setRtcpPortRange: (int) *minimumRtcpAudioPort* maximumRtcpAudioPort: (int) *maximumRtcpAudioPort* minimumRtcpVideoPort: (int) *minimumRtcpVideoPort* maximumRtcpVideoPort: (int) *maximumRtcpVideoPort***

Set the RTCP ports range for audio and video streaming.

**Parameters:**

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

**Returns:**

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

**Remarks:**

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

- (int) enableCallForward: (BOOL) *forBusyOnly* forwardTo: (NSString \*) *forwardTo*

Enable call forward.

**Parameters:**

<i>forBusyOnly</i>	If this parameter is set as true, the SDK will forward all incoming calls when currently it's busy. If it's set as false, the SDK forward all incoming calls anyway.
<i>forwardTo</i>	The call forward target. It must be in the format of sip:xxxx@sip.portsip.com.

**Returns:**

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

- (int) disableCallForward

Disable the call forwarding. The SDK is not forwarding any incoming calls once this function is called.

**Returns:**

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

- (int) enableSessionTimer: (int) *timerSeconds* refreshMode: (SESSION\_REFRESH\_MODE) *refreshMode*

Allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending INVITE requests repeatedly.

**Parameters:**

<i>timerSeconds</i>	The value of the refreshment interval in seconds. Minimum value 90 seconds.
<i>refreshMode</i>	Allow to set the session refreshment by UAC or UAS: SESSION_REFERESH_UAC or SESSION_REFERESH_UAS;

**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 for remembering incoming and outgoing requests (stateful proxies) may continue to retain call state needlessly. If a UA fails to send a BYE message at the end of a session or if the BYE message is lost because of network problems, a stateful proxy does not know that

the session has ended. The re-INVITES ensure that active sessions stay active and completed sessions are terminated.

**- (int) disableSessionTimer**

Disable the session timer.

**Returns:**

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

**- (void) setDoNotDisturb: (BOOL) state**

Enable the "Do not disturb" to enable/disable.

**Parameters:**

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

**- (int) detectMwi**

Used to obtain the MWI status.

**Returns:**

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

**- (int) enableCheckMwi: (BOOL) state**

Allows to enable/disable the check MWI (Message Waiting Indication).

**Parameters:**

<i>state</i>	If it's set as true, it will check MWI automatically once successfully registered to a SIP proxy server.
--------------	--

**Returns:**

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

**- (int) setRtpKeepAlive: (BOOL) state keepAlivePayloadType: (int) keepAlivePayloadType deltaTransmitTimeMS: (int) deltaTransmitTimeMS**

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

**Parameters:**

<i>state</i>	Set as true to allow to send the keep-alive packet during the conversation.
<i>keepAlivePayload</i>	The payload type of the keep-alive RTP packet. Usually it's set as 126.

<i>Type</i>	
<i>deltaTransmitTime</i> <i>MS</i>	The keep-alive RTP packet sending interval, in milliseconds. 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) setKeepAliveTime: (int) *keepAliveTime***

Enable or disable to send SIP keep-alive packet.

**Parameters:**

<i>keepAliveTime</i>	This is the SIP keep-alive time interval in seconds. By setting 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) setAudioSamples: (int) *ptime* maxPtime: (int) *maxPtime***

Set the audio capturing sample.

**Parameters:**

<i>ptime</i>	It should be a multiple of 10 between 10 - 60 (with 10 and 60 inclusive).
<i>maxPtime</i>	For the "maxptime" attribute, it should be a multiple of 10 between 10 - 60 (with 10 and 60 inclusive). It cannot 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.

**Remarks:**

It will appear in the SDP of INVITE and 200 OK message as "ptime and "maxptime" attribute.

**- (int) addSupportedMimeType: (NSString \*) *methodName* mimeType: (NSString \*) *mimeType* subMimeType: (NSString \*) *subMimeType***

Set the SDK to receive the SIP message that includes 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 the RFC3261.
<i>mimeType</i>	The mime type of SIP message.
<i>subMimeType</i>	The sub mime type of SIP message.

**Returns:**

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

## Remarks:

By default, PortSIP VoIP SDK supports the media types (mime types) that are listed 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.  
"application/media_control+xml" in INFO message.
```

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

```
[myVoIPSdk addSupportedMimeType:@"INFO" mimeType:@"text" subMimeType:@"plain"];
```

To receive the NOTIFY message with "application/media\_control+xml":

```
[myVoIPSdk addSupportedMimeType:@"NOTIFY" mimeType:@"application"  
subMimeType:@"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

- (NSString \*) - [PortSIPSDK::getExtensionHeaderValue:headerName:](#)  
*Access the SIP header of SIP message.*
- (int) - [PortSIPSDK::addExtensionHeader:headerValue:](#)  
*Add the extension header (custom header) into every outgoing SIP message.*
- (int) - [PortSIPSDK::clearAddExtensionHeaders](#)  
*Clear the added extension headers (custom headers)*
- (int) - [PortSIPSDK::modifyHeaderValue:headerValue:](#)  
*Modify the special SIP header value for every outgoing SIP message.*
- (int) - [PortSIPSDK::clearModifyHeaders](#)  
*Clear the modified headers value, and do not modify every outgoing SIP message header values any longer.*

---

## Detailed Description

---

### Function Documentation

- (NSString\*) `getExtensionHeaderValue:` (NSString \*) *sipMessage* `headerName:` (NSString \*) *headerName*

Access the SIP header of SIP message.

**Parameters:**

<i>sipMessage</i>	The SIP message.
<i>headerName</i>	The header with which to access the SIP message.

**Returns:**

If the function succeeds, it will return value headerValue. If the function fails, it will return value null.

**Remarks:**

When receiving an SIP message in the onReceivedSignaling callback event, and wish to get SIP message header value, use getExtensionHeaderValue:

```
NSString* headerValue = [myVoIPSdk getExtensionHeaderValue:message headerName:name];
```

**- (int) addExtensionHeader: (NSString \*) headerName headerValue: (NSString \*) headerValue**

Add the extension header (custom header) into every outgoing SIP message.

**Parameters:**

<i>headerName</i>	The custom header name which will appear in every outgoing SIP message.
<i>headerValue</i>	The custom header value.

**Returns:**

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

**- (int) clearAddExtensionHeaders**

Clear the added extension headers (custom headers)

**Returns:**

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

**Remarks:**

```
For example, we have added two customized headers into every outgoing SIP message and wish to remove them.
[myVoIPSdk addExtensionHeader:@"Blling" headerValue:@"usd100.00"];
[myVoIPSdk addExtensionHeader:@"ServiceId" headerValue:@"8873456"];
[myVoIPSdk clearAddextensionHeaders];
```

**- (int) modifyHeaderValue: (NSString \*) headerName headerValue: (NSString \*) headerValue**

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

**Parameters:**

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

**- (int) clearModifyHeaders**

Clear the modified headers value, and do not modify every outgoing SIP message header values any longer.

**Returns:**

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

**Remarks:**

For example, modify two headers' value for every outgoing SIP message and then clear it:

```
[myVoIPSdk modifyHeaderValue:@"Expires" headerValue:@"1000"];
[myVoIPSdk modifyHeaderValue:@"User-Agent", headerValue:@"MyTest Softphone 1.0"];
[myVoIPSdk clearModifyHeaders];
```

## Audio and video functions

### Functions

- (int) - [PortSIPSDK::setVideoDeviceId:](#)  
*Set the video device that will be used for video call.*
- (int) - [PortSIPSDK::setVideoResolution:height:](#)  
*Set the video capturing resolution.*
- (int) - [PortSIPSDK::setAudioBitrate:codecType:bitrateKbps:](#)  
*Set the audio bit rate.*
- (int) - [PortSIPSDK::setVideoBitrate:](#)  
*Set the video bitrate.*
- (int) - [PortSIPSDK::setVideoFrameRate:](#)  
*Set the video frame rate.*
- (int) - [PortSIPSDK::sendVideo:sendState:](#)  
*Send the video to remote side.*
- (int) - [PortSIPSDK::setVideoOrientation:](#)  
*Change the orientation of the video.*
- (void) - [PortSIPSDK::setLocalVideoWindow:](#)  
*Set the window on which the local video image will be displayed.*
- (int) - [PortSIPSDK::setRemoteVideoWindow:remoteVideoWindow:](#)  
*Set the window for a session to display the received remote video image.*
- (int) - [PortSIPSDK::displayLocalVideo:](#)  
*Start/stop displaying the local video image.*
- (int) - [PortSIPSDK::setVideoNackStatus:](#)  
*Enable/disable the NACK feature (RFC4585) to help to improve the video quality.*
- (void) - [PortSIPSDK::muteMicrophone:](#)  
*Mute the device microphone. It's unavailable for Android and iOS.*

- (void) - [PortSIPSDK::muteSpeaker:](#)  
*Mute the device speaker. It's unavailable for Android and iOS.*
- (int) - [PortSIPSDK::setLoudspeakerStatus:](#)  
*Set the audio device that will be used for audio call.*
- (void) - [PortSIPSDK::getDynamicVolumeLevel:microphoneVolume:](#)  
*Obtain the dynamic microphone volume level from current call.*
- (int) - [PortSIPSDK::setChannelOutputVolumeScaling:scaling:](#)

## Detailed Description

## Function Documentation

- (int) **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) **setVideoResolution:** (int) *width* height: (int) *height*

Set the video capturing resolution.

### Parameters:

<i>width</i>	Video width.
<i>height</i>	Video height.

### Returns:

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

- (int) **setAudioBitrate:** (long) *sessionId* codecType: (AUDIOCODEC\_TYPE) *codecType*  
bitrateKbps: (int) *bitrateKbps*

Set the audio bit rate.

### Parameters:

<i>sessionId</i>	The session ID of the call.
<i>codecType</i>	Audio codec type.
<i>bitrateKbps</i>	The video bit rate in KBPS.

**Returns:**

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

**- (int) setVideoBitrate: (int) *bitrateKbps***

Set the video bitrate.

**Parameters:**

<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) setVideoFrameRate: (int) *frameRate***

Set the video frame rate.

**Parameters:**

<i>frameRate</i>	The frame rate value, with its minimum value 5 and maximum value 30. The greater value provides user better video quality while requires more bandwidth.
------------------	--

**Returns:**

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

**Remarks:**

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

**- (int) sendVideo: (long) *sessionId* sendState: (BOOL) *sendState***

Send the video to remote side.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>sendState</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) setVideoOrientation: (int) *rotation***

Change the orientation of the video.

**Parameters:**

<i>rotation</i>	The video rotation that you want to set (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.

- (void) **setLocalVideoWindow:** ([PortSIPVideoRenderView](#) \*) *localVideoWindow*

Set the window on which the local video image will be displayed.

**Parameters:**

<i>localVideoWindow</i>	The <a href="#">PortSIPVideoRenderView</a> for displaying local video image from camera.
-------------------------	--

- (int) **setRemoteVideoWindow:** (long) *sessionId* **remoteVideoWindow:** ([PortSIPVideoRenderView](#) \*) *remoteVideoWindow*

Set the window for a session to display the received remote video image.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>remoteVideoWindow</i>	The <a href="#">PortSIPVideoRenderView</a> for displaying 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) **displayLocalVideo:** (BOOL) *state*

Start/stop displaying the local video image.

**Parameters:**

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

**Returns:**

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

- (int) **setVideoNackStatus:** (BOOL) *state*

Enable/disable the NACK feature (RFC4585) to help 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.

- (void) **muteMicrophone: (BOOL) *mute***

Mute the device microphone. It's unavailable for Android and iOS.

**Parameters:**

<i>mute</i>	If the value is set to true, the microphone is muted, or set to false to un-mute it.
-------------	--

- (void) **muteSpeaker: (BOOL) *mute***

Mute the device speaker. It's unavailable for Android and iOS.

**Parameters:**

<i>mute</i>	If the value is set to true, the speaker is muted, or set to false to un-mute it.
-------------	---

- (int) **setLoudspeakerStatus: (BOOL) *enable***

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

**Parameters:**

<i>enable</i>	By setting to true the SDK uses loudspeaker for audio call. This is 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.

**Remarks:**

Allow to switch between earphone and loudspeaker only.

- (void) **getDynamicVolumeLevel: (int \*) *speakerVolume* microphoneVolume: (int \*) *microphoneVolume***

Obtain the dynamic microphone volume level from current call.

**Parameters:**

<i>speakerVolume</i>	Return the dynamic speaker volume by this parameter. It ranges from 0 - 9.
<i>microphoneVolume</i>	Return the dynamic microphone volume by this parameter. It ranges is 0 - 9.

**Remarks:**

Usually set a timer to call this function to refresh the volume level indicator.

- (int) **setChannelOutputVolumeScaling: (long) *sessionId* scaling: (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.

## Call functions

### Functions

- (long) - [PortSIPSDK::call:sendSdp:videoCall:](#)  
*Make a call.*
- (int) - [PortSIPSDK::rejectCall:code:](#)  
*rejectCall Reject the incoming call.*
- (int) - [PortSIPSDK::hangUp:](#)  
*hangUp Hang up the call.*
- (int) - [PortSIPSDK::answerCall:videoCall:](#)  
*answerCall Answer the incoming call.*
- (int) - [PortSIPSDK::updateCall:enableAudio:enableVideo:](#)  
*Use the re-INVITE to update the established call.*
- (int) - [PortSIPSDK::hold:](#)  
*Place a call on hold.*
- (int) - [PortSIPSDK::unHold:](#)  
*Take off hold.*
- (int) - [PortSIPSDK::muteSession:muteIncomingAudio:muteOutgoingAudio:muteIncomingVideo:muteOutgoingVideo:](#)  
*Mute the specified session audio or video.*
- (int) - [PortSIPSDK::forwardCall:forwardTo:](#)  
*Forward the call to another user once received an incoming call.*
- (int) - [PortSIPSDK::sendDtmf:dtmfMethod:code:dtmfDuration:playDtmfTone:](#)  
*Send DTMF tone.*

---

## Detailed Description

---

## Function Documentation

- (long) call: (NSString \*) *callee* sendSdp: (BOOL) *sendSdp* videoCall: (BOOL) *videoCall*

Make a call.

**Parameters:**

<i>callee</i>	The callee. It can be a name only or full SIP URI. For example, user001,
---------------	--

	sip: <a href="mailto:user001@sip.iptel.org">user001@sip.iptel.org</a> or sip: <a href="mailto:user002@sip.yourdomain.com">user002@sip.yourdomain.com</a> :5068.
<i>sendSdp</i>	If set to false, the outgoing call will not include the SDP in INVITE message.
<i>videoCall</i>	If set to true and at least one video codec was added, the outgoing call will include the video codec into 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 being processed, and you need to detect the final state of calling in onInviteTrying, onInviteRinging, onInviteFailure callback events.

- (int) **rejectCall:** (long) *sessionId* **code:** (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 and 480.

**Returns:**

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

- (int) **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) **answerCall:** (long) *sessionId* **videoCall:** (BOOL) *videoCall*

answerCall Answer the incoming call.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>videoCall</i>	If the incoming call is a video call and the video codec is matched, set it to true to answer the video call. If set to false, the answer call will 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) **updateCall:** (long) *sessionId* **enableAudio:** (BOOL) *enableAudio* **enableVideo:** (BOOL) *enableVideo*

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 updated 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 a specific error code.

**Remarks:**

Example scenario:

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

```
[myVoIPSdk clearVideoCodec];  
[myVoIPSdk addVideoCodec:VIDEOCODEC_H264];  
[myVoIPSdk updateCall:sessionId enableAudio:true enableVideo:true];
```

Example 2: Remove video stream from current conversation.

```
[myVoIPSdk updateCall:sessionId enableAudio:true enableVideo:false];
```

**- (int) hold: (long) *sessionId***

Place a call on hold.

**Parameters:**

<i>sessionId</i>	The 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) 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) muteSession: (long) *sessionId* mutelIncomingAudio: (BOOL) *mutelIncomingAudio* muteOutgoingAudio: (BOOL) *muteOutgoingAudio* mutelIncomingVideo: (BOOL) *mutelIncomingVideo* muteOutgoingVideo: (BOOL) *muteOutgoingVideo***



Mute the specified session audio or video.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>muteIncomingAudio</i>	Set it true to mute incoming audio stream, and user cannot hear from remote side audio.
<i>muteOutgoingAudio</i>	Set it true to mute outgoing audio stream, and the remote side cannot hear the audio.
<i>muteIncomingVideo</i>	Set it true to mute incoming video stream, and user cannot see remote side video.
<i>muteOutgoingVideo</i>	Set it true to mute outgoing video stream, and the remote side cannot see video.

**Returns:**

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

- (int) forwardCall: (long) *sessionId* forwardTo: (NSString \*) *forwardTo*

Forward the call to another user once received an incoming call.

**Parameters:**

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

**Returns:**

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

- (int) sendDtmf: (long) *sessionId* dtmfMethod: (DTMF\_METHOD) *dtmfMethod* code: (int) *code* dtmfDuration: (int) *dtmfDuration* playDtmfTone: (BOOL) *playDtmfTone*

Send DTMF tone.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>dtmfMethod</i>	Support sending DTMF tone with two methods: DTMF_RFC2833 and DTMF_INFO. The DTMF_RFC2833 is recommend.
<i>code</i>	The DTMF tone (0-16).

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>	By setting to true, the SDK plays local DTMF tone sound when sending 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) - [PortSIPSDK::refer:referTo:](#)  
*Refer the current call to another one.*
- (int) - [PortSIPSDK::attendedRefer:replaceSessionId:referTo:](#)  
*Make an attended refer.*
- (long) - [PortSIPSDK::acceptRefer:referSignaling:](#)  
*Once the REFER request accepted, a new call will be made if called this function. The function is usually called after onReceivedRefer callback event.*
- (int) - [PortSIPSDK::rejectRefer:](#)  
*Reject the REFER request.*

---

## Detailed Description

---

## Function Documentation

- (int) refer: (long) *sessionId* referTo: (NSString \*) *referTo*

Refer the current call to another one.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>referTo</i>	Target of the refer. It could 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:**

```
[myVoIPSdk refer:sessionId referTo:@"sip:testuser12@sip.portsip.com"];
```

You can watch the video on YouTube at <https://www.youtube.com/watch?v=2w9EGgr3FY>. It will demonstrate the transfer.

**- (int) attendedRefer: (long) sessionId replaceSessionId: (long) replaceSessionId referTo: (NSString \*) referTo**

Make an attended refer.

**Parameters:**

<i>sessionId</i>	The session ID of the call.
<i>replaceSessionId</i>	Session ID of the replaced call.
<i>referTo</i>	Target 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 value a specific error code.

**Remarks:**

Please read the sample project source code to get more details. Or you can watch the video on YouTube at <https://www.youtube.com/watch?v=2w9EGgr3FY>. It will demonstrate the transfer.

**- (long) acceptRefer: (long) referId referSignaling: (NSString \*) referSignaling**

Once the REFER request accepted, a new call will be made if called this function. 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 that is greater than 0 to the new call for REFER, otherwise returns a specific error code that is less than 0.

**- (int) 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) - [PortSIPSDK::enableSendPcmStreamToRemote:state:streamSamplesPerSec:](#)  
*Enable the SDK to send PCM stream data to remote side from another source instead of microphone.*
- (int) - [PortSIPSDK::sendPcmStreamToRemote:data:](#)  
*Send the audio stream in PCM format from another source instead of audio device capturing (microphone).*
- (int) - [PortSIPSDK::enableSendVideoStreamToRemote:state:](#)  
*Enable the SDK to send video stream data to remote side from another source instead of camera.*
- (int) - [PortSIPSDK::sendVideoStreamToRemote:data:width:height:](#)  
*Send the video stream to remote side.*

---

### Detailed Description

---

### Function Documentation

- (int) **enableSendPcmStreamToRemote:** (long) *sessionId* state: (BOOL) *state* streamSamplesPerSec: (int) *streamSamplesPerSec*

Enable the SDK to send PCM stream data to remote side from another source instead of microphone.

**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.

**Remarks:**

To send the PCM stream data to another side, this function MUST be called first.

- (int) **sendPcmStreamToRemote:** (long) *sessionId* data: (NSData \*) *data*

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.

**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:

```
[myVoIPSdk enableSendPcmStreamToRemote:sessionId state:YES  
streamSamplesPerSec:16000];  
[myVoIPSdk sendPcmStreamToRemote:sessionId data:data];
```

- (int) enableSendVideoStreamToRemote: (long) *sessionId* state: (BOOL) *state*

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

**Parameters:**

<i>sessionId</i>	The session ID of call.
<i>state</i>	Set to true to enable the sending 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) sendVideoStreamToRemote: (long) *sessionId* data: (NSData \*) *data* width: (int) *width* height: (int) *height*

Send the video stream to remote side.

**Parameters:**

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

**Returns:**

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

**Remarks:**

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.

Usually we should use it like below:

```
[myVoIPSdk enableSendVideoStreamToRemote:sessionId state:YES];  
[myVoIPSdk sendVideoStreamToRemote:sessionId data:data width:352 height:288];
```

# RTP packets, audio stream and video stream callback functions

## Functions

- (int) - [PortSIPSDK::setRtpCallback:](#)  
*Set the RTP callbacks to allow to access the sent and received RTP packets.*
- (int) - [PortSIPSDK::enableAudioStreamCallback:enable:callbackMode:](#)  
*Enable/disable the audio stream callback.*
- (int) - [PortSIPSDK::enableVideoStreamCallback:callbackMode:](#)  
*Enable/disable the video stream callback.*
- (int) - [PortSIPSDK::enableVideoDecoderCallback:](#)  
*Enable/disable the video Decoder callback.*

---

## Detailed Description

---

## Function Documentation

- (int) **setRtpCallback:** (BOOL) *enable*

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

### Parameters:

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

### Returns:

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

- (int) **enableAudioStreamCallback:** (long) *sessionId* **enable:** (BOOL) *enable*  
**callbackMode:** (AUDIOSTREAM\_CALLBACK\_MODE) *callbackMode*

Enable/disable the audio stream callback.

### 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>callbackMode</i>	The audio stream callback mode

Type	Description
AUDIOSTREAM_LOCAL_MIX	Callback the audio stream from microphone for all channels.
AUDIOSTREAM_LOCAL_PER_CHANNEL	Callback the audio stream from microphone

	for one channel based on the given <code>sessionId</code> .
AUDIOSTREAM_REMOTE_MIX	Callback the received audio stream that is mixed to include all channels.
AUDIOSTREAM_REMOTE_PER_CHANNEL	Callback the received audio stream for one channel based on the given <code>sessionId</code> .

**Returns:**

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

**Remarks:**

The `onAudioRawCallback` event will be triggered if the callback is enabled.

- (int) `enableVideoStreamCallback`: (long) `sessionId` `callbackMode`:  
(`VIDEOSTREAM_CALLBACK_MODE`) `callbackMode`

Enable/disable the video stream callback.

**Parameters:**

<code>sessionId</code>	The session ID of call.
<code>callbackMode</code>	The video stream callback mode.
Mode	Description
VIDEOSTREAM_NONE	Disable video stream callback.
VIDEOSTREAM_LOCAL	Local video stream callback.
VIDEOSTREAM_REMOTE	Remote video stream callback.
VIDEOSTREAM_BOTH	Both of local and remote video stream callback.

**Returns:**

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

**Remarks:**

The `onVideoRawCallback` event will be triggered if the callback is enabled.

- (int) `enableVideoDecoderCallback`: (BOOL) `enable`

Enable/disable the video Decoder callback.

**Parameters:**

<code>enable</code>	Set to true to enable video Decoder callback, or false to stop the callback.
---------------------	--

**Returns:**

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

**Remarks:**

The `onVideoDecoderCallback` event will be triggered if the callback is enabled.

## Record functions

### Functions

- (int) - [PortSIPSDK::startRecord:recordFilePath:recordFileName:appendTimeStamp:audioFileFormat:audioRecordMode:aviFileCodecType:videoRecordMode:](#)  
Start recording the call.
- (int) - [PortSIPSDK::stopRecord:](#)  
Stop recording.

---

### Detailed Description

---

### Function Documentation

- (int) **startRecord:** (long) *sessionId* **recordFilePath:** (NSString \*) *recordFilePath* **recordFileName:** (NSString \*) *recordFileName* **appendTimeStamp:** (BOOL) *appendTimeStamp* **audioFileFormat:** (AUDIO\_FILE\_FORMAT) *audioFileFormat* **audioRecordMode:** (RECORD\_MODE) *audioRecordMode* **aviFileCodecType:** (VIDEOCODEC\_TYPE) *aviFileCodecType* **videoRecordMode:** (RECORD\_MODE) *videoRecordMode*

Start recording the call.

#### Parameters:

<i>sessionId</i>	The session ID of call conversation.
<i>recordFilePath</i>	The filepath to which the recording will be saved. It must be existent.
<i>recordFileName</i>	The filename of the recording. For example audiorecord.wav or videorecord.avi.
<i>appendTimeStamp</i>	Set to true to append the timestamp to the filename of the recording.
<i>audioFileFormat</i>	The file format for the audio recording.
<i>audioRecordMode</i>	The audio recording mode.
<i>aviFileCodecType</i>	The codec that is used for compressing the video data to save into video recording.
<i>videoRecordMode</i>	Allow to set video recording 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) **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 files to remote party

### Functions

- (int) - [PortSIPSDK::playVideoFileToRemote:aviFile:loop:playAudio:](#)  
*Play an AVI file to remote party.*
- (int) - [PortSIPSDK::stopPlayVideoFileToRemote:](#)  
*Stop playing video file to remote party.*
- (int) - [PortSIPSDK::playAudioFileToRemote:filename:fileSamplesPerSec:loop:](#)  
*Play a wave file to remote party.*
- (int) - [PortSIPSDK::stopPlayAudioFileToRemote:](#)  
*Stop playing wave file to remote party.*
- (int) - [PortSIPSDK::playAudioFileToRemoteAsBackground:filename:fileSamplesPerSec:](#)  
*Play a wave file to remote party as conversation background sound.*
- (int) - [PortSIPSDK::stopPlayAudioFileToRemoteAsBackground:](#)  
*Stop playing a wave file to remote party as conversation background sound.*

### Detailed Description

### Function Documentation

- (int) **playVideoFileToRemote:** (long) *sessionId* **aviFile:** (NSString \*) *aviFile* **loop:** (BOOL) *loop* **playAudio:** (BOOL) *playAudio*

Play an AVI file to remote party.

**Parameters:**

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

**Returns:**

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

**- (int) stopPlayVideoFileToRemote: (long) *sessionId***

Stop playing video file to remote party.

**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) playAudioFileToRemote: (long) *sessionId* filename: (NSString \*) *filename* fileSamplesPerSec: (int) *fileSamplesPerSec* loop: (BOOL) *loop***

Play a wave file to remote party.

**Parameters:**

<i>sessionId</i>	Session ID of the call.
<i>filename</i>	The full filepath, such as "/test.wav".
<i>fileSamplesPerSec</i>	The sample wave file in seconds. It should be 8000, 16000 or 32000.
<i>loop</i>	Set to false to stop playing audio file when it is ended, or true to play it repeatedly.

**Returns:**

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

**- (int) stopPlayAudioFileToRemote: (long) *sessionId***

Stop playing wave file to remote party.

**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) playAudioFileToRemoteAsBackground: (long) *sessionId* filename: (NSString \*) *filename* fileSamplesPerSec: (int) *fileSamplesPerSec***

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

**Parameters:**

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

**Returns:**

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

**- (int) stopPlayAudioFileToRemoteAsBackground: (long) sessionId**

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

**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.

## Conference functions

### Functions

- (int) - [PortSIPSDK::createConference:videoWidth:videoHeight:displayLocalVideo:](#)  
*Create a conference. It will fail if the existent conference does not end yet.*
- (void) - [PortSIPSDK::destroyConference](#)  
*End the exist conference.*
- (int) - [PortSIPSDK::setConferenceVideoWindow:](#)  
*Set the window for a conference, which is used to display the remote video image received.*
- (int) - [PortSIPSDK::joinToConference:](#)  
*Join a session into existent conference. If the call is in hold, please un-hold it first.*
- (int) - [PortSIPSDK::removeFromConference:](#)  
*Remove a session from an existent conference.*

---

## Detailed Description

---

### Function Documentation

**- (int) createConference: ([PortSIPVideoRenderView](#) \*) conferenceVideoWindow  
videoWidth: (int) *videoWidth* videoHeight: (int) *videoHeight* displayLocalVideo: (BOOL)  
*displayLocalVideoInConference***

Create a conference. It will fail if the existent conference does not end yet.

**Parameters:**

<i>conferenceVideoWindow</i>	The <a href="#">PortSIPVideoRenderView</a> used for displaying the conference video.
<i>videoWidth</i>	The conference video width.
<i>videoHeight</i>	The conference video height.
<i>displayLocalVideoInConference</i>	Display the local video on video window.

**Returns:**

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

- (int) **setConferenceVideoWindow:** ([PortSIPVideoRenderView \\*](#)) *conferenceVideoWindow*

Set the window for a conference, which is used to display the remote video image received.

**Parameters:**

<i>conferenceVideoWindow</i>	The <a href="#">PortSIPVideoRenderView</a> that is used to display the conference video.
------------------------------	--

**Returns:**

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

- (int) **joinToConference:** (long) *sessionId*

Join a session into existent conference. If the call is in hold, please un-hold it first.

**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) **removeFromConference:** (long) *sessionId*

Remove a session from an existent 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) - [PortSIPSDK::setAudioRtcpBandwidth:BitsRR:BitsRS:KBitsAS:](#)  
*Set the audio RTCP bandwidth parameters as the RFC3556.*
- (int) - [PortSIPSDK::setVideoRtcpBandwidth:BitsRR:BitsRS:KBitsAS:](#)  
*Set the video RTCP bandwidth parameters as the RFC3556.*
- (int) - [PortSIPSDK::setAudioQos:DSCPValue:priority:](#)  
*Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for audio channel.*
- (int) - [PortSIPSDK::setVideoQos:DSCPValue:](#)  
*Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for video channel.*
- (int) - [PortSIPSDK::setVideoMTU:](#)  
*Set the MTU size for video RTP packet.*

---

### Detailed Description

---

### Function Documentation

- (int) **setAudioRtcpBandwidth:** (long) *sessionId* **BitsRR:** (int) *BitsRR* **BitsRS:** (int) *BitsRS* **KBitsAS:** (int) *KBitsAS*

Set the audio 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) **setVideoRtcpBandwidth:** (long) *sessionId* **BitsRR:** (int) *BitsRR* **BitsRS:** (int) *BitsRS* **KBitsAS:** (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) **setAudioQos: (BOOL) enable DSCPValue: (int) DSCPValue priority: (int) priority**

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

**Parameters:**

<i>enable</i>	Set to true to enable audio QoS.
<i>DSCPValue</i>	The six-bit DSCP value. Valid value ranges 0-63. As defined in RFC 2472, the DSCP value is the high-order 6 bits of the IP version 4 (IPv4) TOS field and the IP version 6 (IPv6) Traffic Class field.
<i>priority</i>	The 802.1p priority (PCP) field in a 802.1Q/VLAN tag. Values 0-7 indicate the priority, while value -1 leaves the priority setting unchanged.

**Returns:**

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

- (int) **setVideoQos: (BOOL) enable DSCPValue: (int) DSCPValue**

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

**Parameters:**

<i>enable</i>	Set as true to enable QoS, or false to disable.
<i>DSCPValue</i>	The six-bit DSCP value. Valid value ranges 0-63. As defined in RFC 2472, the DSCP value is the high-order 6 bits of the IP version 4 (IPv4) TOS field and the IP version 6 (IPv6) Traffic Class field.

**Returns:**

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

- (int) **setVideoMTU: (int) mtu**

Set the MTU size for video RTP packet.

**Parameters:**

<i>mtu</i>	Set MTU value. Allowed value ranges 512-65507. Other values will be automatically changed to the default 1400.
------------	--

**Returns:**

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

## RTP statistics functions

### Functions

- (int) - [PortSIPSDK::getNetworkStatistics:currentBufferSize:preferredBufferSize:currentPacketLossRate:currentDiscardRate:currentExpandRate:currentPreemptiveRate:currentAccelerateRate:](#)  
Get the "in-call" statistics. The statistics are reset after the query.
- (int) - [PortSIPSDK::getAudioRtpStatistics:averageJitterMs:maxJitterMs:discardedPackets:](#)  
Obtain the RTP statistics of audio channel.
- (int) - [PortSIPSDK::getAudioRtcpStatistics:bytesSent:packetsSent:bytesReceived:packetsReceived:sendFractionLost:sendCumulativeLost:recvFractionLost:recvCumulativeLost:](#)  
Obtain the RTCP statistics of audio channel.
- (int) - [PortSIPSDK::getVideoRtpStatistics:bytesSent:packetsSent:bytesReceived:packetsReceived:](#)  
Obtain the RTP statistics of video.

---

### Detailed Description

---

### Function Documentation

- (int) **getNetworkStatistics: (long) *sessionId* currentBufferSize: (int \*) *currentBufferSize* preferredBufferSize: (int \*) *preferredBufferSize* currentPacketLossRate: (int \*) *currentPacketLossRate* currentDiscardRate: (int \*) *currentDiscardRate* currentExpandRate: (int \*) *currentExpandRate* currentPreemptiveRate: (int \*) *currentPreemptiveRate* currentAccelerateRate: (int \*) *currentAccelerateRate***

Get the "in-call" statistics. The statistics are reset after the query.

#### Parameters:

<i>sessionId</i>	The session ID of call conversation.
<i>currentBufferSize</i>	Current jitter buffer size in ms.
<i>preferredBufferSize</i>	Preferred buffer size in ms.
<i>currentPacketLossRate</i>	Loss rate (network + late) in percentage.
<i>currentDiscardRate</i>	Late loss rate in percentage.
<i>currentExpandRate</i>	Fraction (of original stream) of synthesized speech inserted through expansion.
<i>currentPreemptiveRate</i>	Fraction of synthesized speech inserted through pre-emptive expansion.
<i>currentAccelerateRate</i>	Fraction of data removed through acceleration.

**Returns:**

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

**- (int) getAudioRtpStatistics: (long) sessionId averageJitterMs: (int \*) averageJitterMs maxJitterMs: (int \*) maxJitterMs discardedPackets: (int \*) discardedPackets**

Obtain the RTP statistics of audio channel.

**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>averageJitterMs</i>	Short-time average jitter, in milliseconds.
<i>maxJitterMs</i>	Maximum short-time jitter, in milliseconds.
<i>discardedPackets</i>	The number of discarded packets on a channel during the call.

**Returns:**

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

**- (int) getAudioRtcpStatistics: (long) sessionId bytesSent: (int \*) bytesSent packetsSent: (int \*) packetsSent bytesReceived: (int \*) bytesReceived packetsReceived: (int \*) packetsReceived sendFractionLost: (int \*) sendFractionLost sendCumulativeLost: (int \*) sendCumulativeLost rcvFractionLost: (int \*) rcvFractionLost rcvCumulativeLost: (int \*) rcvCumulativeLost**

Obtain the RTCP statistics of audio channel.

**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>bytesSent</i>	The number of sent bytes.
<i>packetsSent</i>	The number of sent packets.
<i>bytesReceived</i>	The number of received bytes.
<i>packetsReceived</i>	The number of received packets.
<i>sendFractionLost</i>	Fraction of sent lost in percentage.
<i>sendCumulativeLost</i>	The number of sent cumulative lost packet.
<i>st</i>	
<i>rcvFractionLost</i>	Fraction of received lost in percent.
<i>rcvCumulativeLost</i>	The number of received cumulative lost packets.
<i>st</i>	

**Returns:**

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

**- (int) getVideoRtpStatistics: (long) sessionId bytesSent: (int \*) bytesSent packetsSent: (int \*) packetsSent bytesReceived: (int \*) bytesReceived packetsReceived: (int \*) packetsReceived**

Obtain the RTP statistics of video.



**Parameters:**

<i>sessionId</i>	The session ID of call conversation.
<i>bytesSent</i>	The number of sent bytes.
<i>packetsSent</i>	The number of sent packets.
<i>bytesReceived</i>	The number of received bytes.
<i>packetsReceived</i>	The number of received packets.

**Returns:**

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

## Audio effect functions

### Functions

- (void) - [PortSIPSDK::enableVAD:](#)  
*Enable/disable Voice Activity Detection (VAD).*
- (void) - [PortSIPSDK::enableAEC:](#)  
*Enable/disable AEC (Acoustic Echo Cancellation).*
- (void) - [PortSIPSDK::enableCNG:](#)  
*Enable/disable Comfort Noise Generator (CNG).*
- (void) - [PortSIPSDK::enableAGC:](#)  
*Enable/disable Automatic Gain Control (AGC).*
- (void) - [PortSIPSDK::enableANS:](#)  
*Enable/disable Audio Noise Suppression (ANS).*

---

## Detailed Description

---

### Function Documentation

- (void) **enableVAD: (BOOL) state**

Enable/disable Voice Activity Detection (VAD).

**Parameters:**

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

- (void) **enableAEC: (EC\_MODES) state**

Enable/disable AEC (Acoustic Echo Cancellation).

**Parameters:**

<i>state</i>	AEC type. Default is EC_NONE.
--------------	-------------------------------

- (void) enableCNG: (BOOL) *state*

Enable/disable Comfort Noise Generator (CNG).

**Parameters:**

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

- (void) enableAGC: (AGC\_MODES) *state*

Enable/disable Automatic Gain Control (AGC).

**Parameters:**

<i>state</i>	AGC type. Default is AGC_NONE.
--------------	--------------------------------

- (void) enableANS: (NS\_MODES) *state*

Enable/disable Audio Noise Suppression (ANS).

**Parameters:**

<i>state</i>	NS type. Default is NS_NONE.
--------------	------------------------------

## Send OPTIONS/INFO/MESSAGE functions

### Functions

- (int) - [PortSIPSDK::sendOptions:sdp:](#)  
*Send OPTIONS message.*
- (int) - [PortSIPSDK::sendInfo:mimeType:subMimeType:infoContents:](#)  
*Send an INFO message to remote side in dialog.*
- (long) - [PortSIPSDK::sendMessage:mimeType:subMimeType:message:messageLength:](#)  
*Send a MESSAGE message to remote side in dialog.*
- (long) - [PortSIPSDK::sendOutOfDialogMessage:mimeType:subMimeType:message:messageLength:](#)  
*Send an out of dialog MESSAGE message to remote side.*

---

### Detailed Description

---

## Function Documentation

- (int) **sendOptions:** (NSString \*) *to* sdp: (NSString \*) *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 wish to send the SDP with OPTIONS message.

### Returns:

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

- (int) **sendInfo:** (long) *sessionId* mimeType: (NSString \*) *mimeType* subMimeType: (NSString \*) *subMimeType* infoContents: (NSString \*) *infoContents*

Send an 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 to 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) **sendMessage:** (long) *sessionId* mimeType: (NSString \*) *mimeType* subMimeType: (NSString \*) *subMimeType* message: (NSData \*) *message* messageLength: (int) *messageLength*

Send a MESSAGE message to remote side in dialog.

### Parameters:

<i>sessionId</i>	The session ID of the 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 to 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 less than 0.

### Remarks:

Example 1: Send a plain text message. Note: to send other languages text, please use the UTF-8 to encode the message before sending.

```
[myVoIPsdk sendMessage:sessionId mimeType:@"text" subMimeType:@"plain" message:data  
messageLength:dataLen];
```

Example 2: Send a binary message.

```
[myVoIPsdk sendMessage:sessionId mimeType:@"application" subMimeType:@"vnd.3gpp.sms"  
message:data messageLength:dataLen];
```

**- (long) sendOutOfDialogMessage: (NSString \*) to mimeType: (NSString \*) mimeType  
subMimeType: (NSString \*) subMimeType message: (NSData \*) message  
messageLength: (int) messageLength**

Send an out of dialog MESSAGE message to remote side.

### Parameters:

<i>to</i>	The message recipient, such as 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>message</i>	The contents 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 less than 0.

### Remarks:

Example 1: Send a plain text message. Note: to send text in other languages, please use UTF-8 to encode the message before sending.

```
[myVoIPsdk sendOutOfDialogMessage:@"sip:user1@sip.portsip.com" mimeType:@"text"  
subMimeType:@"plain" message:data messageLength:dataLen];
```

Example 2: Send a binary message.

```
[myVoIPsdk sendOutOfDialogMessage:@"sip:user1@sip.portsip.com"  
mimeType:@"application" subMimeType:@"vnd.3gpp.sms" message:data  
messageLength:dataLen];
```

## Presence functions

### Functions

- (int) - [PortSIPSDK::presenceSubscribeContact:subject:](#)  
*Send a SUBSCRIBE message for presence to a contact.*
- (int) - [PortSIPSDK::presenceAcceptSubscribe:](#)  
*Accept the presence SUBSCRIBE request that is received from contact.*
- (int) - [PortSIPSDK::presenceRejectSubscribe:](#)  
*Reject a presence SUBSCRIBE request that is received from contact.*
- (int) - [PortSIPSDK::presenceOnline:statusText:](#)  
*Send a NOTIFY message to contact to notify that presence status is online/changed.*
- (int) - [PortSIPSDK::presenceOffline:](#)

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

---

## Detailed Description

---

## Function Documentation

- (int) **presenceSubscribeContact: (NSString \*) contact subject: (NSString \*) subject**

Send a SUBSCRIBE message for presence to a contact.

### Parameters:

<i>contact</i>	The target contact. It must be like 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 may be in UTF-8 format. You should use UTF-8 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) **presenceAcceptSubscribe: (long) subscribelId**

Accept the presence SUBSCRIBE request that is received from contact.

### Parameters:

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

### Returns:

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

- (int) **presenceRejectSubscribe: (long) subscribelId**

Reject a presence SUBSCRIBE request that is received from contact.

### Parameters:

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

### Returns:

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

- (int) presenceOnline: (long) *subscriptionId* statusText: (NSString \*) *statusText*

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

**Parameters:**

<i>subscriptionId</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 online. For example: "I'm here"

**Returns:**

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

- (int) presenceOffline: (long) *subscriptionId*

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

**Parameters:**

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

**Returns:**

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

## Keep awake functions

### Functions

- (BOOL) - [PortSIPSDK::startKeepAwake](#)  
*Keep VoIP awake in the background. If you want your application to be able to receive the incoming call while it's running in background, you should call this function in applicationDidEnterBackground.*
- (BOOL) - [PortSIPSDK::stopKeepAwake](#)  
*Keep VoIP awake in the background. Call this function in applicationWillEnterForeground once your application comes back to foreground.*

---

## Detailed Description

---

## Function Documentation

- (BOOL) startKeepAwake

Keep VoIP awake in the background. If you want your application to be able to receive the incoming call while it's running in background, you should call this function in `applicationDidEnterBackground`.

**Returns:**

If the function succeeds, it will return value `true`. If the function fails, it will return value `false`.

**- (BOOL) stopKeepAwake**

Keep VoIP awake in the background. Call this function in `applicationWillEnterForeground` once your application comes back to foreground.

**Returns:**

If the function succeeds, it will return value `true`. If the function fails, it will return value `false`.

# Class Documentation

## <PortSIPEventDelegate > Protocol Reference

PortSIP SDK Callback events Delegate.

```
#import <PortSIPEventDelegate.h>
```

Inherits <NSObject>.

### Instance Methods

- (void) - [onRegisterSuccess:statusCode:](#)
- (void) - [onRegisterFailure:statusCode:](#)
- (void) - [onInviteIncoming:callerDisplayName:caller:calleeDisplayName:callee:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [onInviteTrying:](#)
- (void) - [onInviteSessionProgress:audioCodecs:videoCodecs:existsEarlyMedia:existsAudio:existsVideo:](#)
- (void) - [onInviteRinging:statusText:statusCode:](#)
- (void) - [onInviteAnswered:callerDisplayName:caller:calleeDisplayName:callee:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [onInviteFailure:reason:code:](#)
- (void) - [onInviteUpdated:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [onInviteConnected:](#)
- (void) - [onInviteBeginingForward:](#)
- (void) - [onInviteClosed:](#)
- (void) - [onRemoteHold:](#)
- (void) - [onRemoteUnHold:audioCodecs:videoCodecs:existsAudio:existsVideo:](#)
- (void) - [onReceivedRefer:referId:to:from:referSipMessage:](#)
- (void) - [onReferAccepted:](#)
- (void) - [onReferRejected:reason:code:](#)
- (void) - [onTransferTrying:](#)
- (void) - [onTransferRinging:](#)
- (void) - [onACTVTransferSuccess:](#)
- (void) - [onACTVTransferFailure:reason:code:](#)
- (void) - [onReceivedSignaling:message:](#)
- (void) - [onSendingSignaling:message:](#)
- (void) - [onWaitingVoiceMessage:urgentNewMessageCount:urgentOldMessageCount:newMessageCount:oldMessageCount:](#)
- (void) - [onWaitingFaxMessage:urgentNewMessageCount:urgentOldMessageCount:newMessageCount:oldMessageCount:](#)
- (void) - [onRecvDtmfTone:tone:](#)
- (void) - [onRecvOptions:](#)
- (void) - [onRecvInfo:](#)
- (void) - [onPresenceRecvSubscribe:fromDisplayName:from:subject:](#)
- (void) - [onPresenceOnline:from:stateText:](#)
- (void) - [onPresenceOffline:from:](#)
- (void) - [onRecvMessage:mimeType:subMimeType:messageData:messageDataLength:](#)



- (void) - [onRecvOutOfDialogMessage:from:toDisplayName:to:mimeType:subMimeType:messageData:messageDataLength:](#)
- (void) - [onSendMessageSuccess:messageId:](#)
- (void) - [onSendMessageFailure:messageId:reason:code:](#)
- (void) - [onSendOutOfDialogMessageSuccess:fromDisplayName:from:toDisplayName:to:](#)
- (void) - [onSendOutOfDialogMessageFailure:fromDisplayName:from:toDisplayName:to:reason:code:](#)
- (void) - [onPlayAudioFileFinished:fileName:](#)
- (void) - [onPlayVideoFileFinished:](#)
- (void) - [onReceivedRTPPacket:isAudio:RTPPacket:packetSize:](#)
- (void) - [onSendingRTPPacket:isAudio:RTPPacket:packetSize:](#)
- (void) - [onAudioRawCallback:audioCallbackMode:data:dataLength:samplingFreqHz:](#)
- (void) - [onVideoRawCallback:videoCallbackMode:width:height:data:dataLength:](#)
- (void) - [onVideoDecoderCallback:width:height:framerate:bitrate:](#)

## Detailed Description

PortSIP SDK Callback events Delegate.

### Author:

Copyright (c) 2006-2014 PortSIP Solutions, Inc. All rights reserved.

### Version:

11.2

### See also:

<http://www.PortSIP.com> PortSIP SDK Callback events Delegate description.

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

- PortSIPEventDelegate.h

## PortSIP SDK Class Reference

PortSIP VoIP SDK functions class.

```
#import <PortSIPSDK.h>
```

Inherits NSObject.

### Instance Methods

- (int) - [initialize:loglevel:logPath:maxLine:agent:audioDeviceLayer:videoDeviceLayer:](#)  
*Initialize the SDK.*
- (int) - [setInstanceId:](#)  
*Set the instance Id, the outbound instanceId((RFC5626) ) used in contact headers.*
- (void) - [unInitialize](#)  
*Un-initialize the SDK and release resources.*
- (int) - [setUser:displayName:authName:password:localIP:localSIPPort:userDomain:SIPServer:SIPServerPort:STUNServer:STUNServerPort:outboundServer:outboundServerPort:](#)  
*Set user account info.*
- (int) - [registerServer:retryTimes:](#)  
*Register to SIP proxy server (login to server)*
- (int) - [refreshRegisterServer:](#)  
*Request a manual refresh of the registration.*
- (int) - [unRegisterServer](#)  
*Un-register from the SIP proxy server.*
- (int) - [setLicenseKey:](#)  
*Set the license key. It must be called before setUser function.*
- (int) - [getNICNums](#)  
*Get the Network Interface Card numbers.*
- (NSString \*) - [getLocalIpAddress:](#)  
*Get the local IP address by Network Interface Card index.*
- (int) - [addAudioCodec:](#)  
*Enable an audio codec. It will appear in SDP.*
- (int) - [addVideoCodec:](#)  
*Enable a video codec, and it will appear in SDP.*
- (BOOL) - [isAudioCodecEmpty](#)  
*Detect if the enabled audio codecs is empty.*
- (BOOL) - [isVideoCodecEmpty](#)  
*Detect enabled video codecs is empty or not.*
- (int) - [setAudioCodecPayloadType:payloadType:](#)  
*Set the RTP payload type for dynamic audio codec.*
- (int) - [setVideoCodecPayloadType:payloadType:](#)  
*Set the RTP payload type for dynamic Video codec.*
- (void) - [clearAudioCodec](#)  
*Remove all enabled audio codecs.*
- (void) - [clearVideoCodec](#)  
*Remove all enabled video codecs.*
- (int) - [setAudioCodecParameter:parameter:](#)

- Set the codec parameter for audio codec.*

  - (int) - [setVideoCodecParameter:parameter:](#)  
*Set the codec parameter for video codec.*
  - (int) - [setDisplayname:](#)  
*Set user display name.*
  - (int) - [getVersion:minorVersion:](#)  
*Get the current version number of the SDK.*
  - (int) - [enableReliableProvisional:](#)  
*Enable/disable PRACK.*
  - (int) - [enable3GppTags:](#)  
*Enable/disable the 3Gpp tags, including "ims.icsi.mmtel" and "g.3gpp.smsip".*
  - (void) - [enableCallbackSendingSignaling:](#)  
*Enable/disable to callback the sent SIP messages.*
  - (int) - [setSrtpPolicy:](#)  
*Set the SRTP policy.*
  - (int) - [setRtpPortRange:maximumRtpAudioPort:minimumRtpVideoPort:maximumRtpVideoPort:](#)  
*Set the RTP ports range for audio and video streaming.*
  - (int) - [setRtcpPortRange:maximumRtcpAudioPort:minimumRtcpVideoPort:maximumRtcpVideoPort:](#)  
*Set the RTCP ports range for audio and video streaming.*
  - (int) - [enableCallForward:forwardTo:](#)  
*Enable call forward.*
  - (int) - [disableCallForward:](#)  
*Disable the call forwarding. The SDK is not forwarding any incoming calls once this function is called.*
  - (int) - [enableSessionTimer:refreshMode:](#)  
*Allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending INVITE requests repeatedly.*
  - (int) - [disableSessionTimer:](#)  
*Disable the session timer.*
  - (void) - [setDoNotDisturb:](#)  
*Enable the "Do not disturb" to enable/disable.*
  - (int) - [detectMwi:](#)  
*Used to obtain the MWI status.*
  - (int) - [enableCheckMwi:](#)  
*Allows to enable/disable the check MWI (Message Waiting Indication).*
  - (int) - [setRtpKeepAlive:keepAlivePayloadType:deltaTransmitTimeMS:](#)  
*Enable or disable to send RTP keep-alive packet when the call is established.*
  - (int) - [setKeepAliveTime:](#)  
*Enable or disable to send SIP keep-alive packet.*
  - (int) - [setAudioSamples:maxPtime:](#)  
*Set the audio capturing sample.*
  - (int) - [addSupportedMimeType:mimeType:subMimeType:](#)  
*Set the SDK to receive the SIP message that includes special mime type.*
  - (NSString \*) - [getExtensionHeaderValue:headerName:](#)  
*Access the SIP header of SIP message.*
  - (int) - [addExtensionHeader:headerValue:](#)  
*Add the extension header (custom header) into every outgoing SIP message.*

- (int) - [clearAddExtensionHeaders](#)  
*Clear the added extension headers (custom headers)*
- (int) - [modifyHeaderValue:headerValue:](#)  
*Modify the special SIP header value for every outgoing SIP message.*
- (int) - [clearModifyHeaders](#)  
*Clear the modified headers value, and do not modify every outgoing SIP message header values any longer.*
- (int) - [setVideoDeviceId:](#)  
*Set the video device that will be used for video call.*
- (int) - [setVideoResolution:height:](#)  
*Set the video capturing resolution.*
- (int) - [setAudioBitrate:codecType:bitrateKbps:](#)  
*Set the audio bit rate.*
- (int) - [setVideoBitrate:](#)  
*Set the video bitrate.*
- (int) - [setVideoFrameRate:](#)  
*Set the video frame rate.*
- (int) - [sendVideo:sendState:](#)  
*Send the video to remote side.*
- (int) - [setVideoOrientation:](#)  
*Change the orientation of the video.*
- (void) - [setLocalVideoWindow:](#)  
*Set the window on which the local video image will be displayed.*
- (int) - [setRemoteVideoWindow:remoteVideoWindow:](#)  
*Set the window for a session to display the received remote video image.*
- (int) - [displayLocalVideo:](#)  
*Start/stop displaying the local video image.*
- (int) - [setVideoNackStatus:](#)  
*Enable/disable the NACK feature (RFC4585) to help to improve the video quality.*
- (void) - [muteMicrophone:](#)  
*Mute the device microphone. It's unavailable for Android and iOS.*
- (void) - [muteSpeaker:](#)  
*Mute the device speaker. It's unavailable for Android and iOS.*
- (int) - [setLoudspeakerStatus:](#)  
*Set the audio device that will be used for audio call.*
- (void) - [getDynamicVolumeLevel:microphoneVolume:](#)  
*Obtain the dynamic microphone volume level from current call.*
- (int) - [setChannelOutputVolumeScaling:scaling:](#)
- (long) - [call:sendSdp:videoCall:](#)  
*Make a call.*
- (int) - [rejectCall:code:](#)  
*rejectCall Reject the incoming call.*
- (int) - [hangUp:](#)  
*hangUp Hang up the call.*
- (int) - [answerCall:videoCall:](#)  
*answerCall Answer the incoming call.*
- (int) - [updateCall:enableAudio:enableVideo:](#)

Use the re-INVITE to update the established call.

- (int) - [hold:](#)  
Place a call on hold.
- (int) - [unHold:](#)  
Take off hold.
- (int) - [muteSession:muteIncomingAudio:muteOutgoingAudio:muteIncomingVideo:muteOutgoingVideo:](#)  
Mute the specified session audio or video.
- (int) - [forwardCall:forwardTo:](#)  
Forward the call to another user once received an incoming call.
- (int) - [sendDtmf:dtmfMethod:code:dtmfDuration:playDtmfTone:](#)  
Send DTMF tone.
- (int) - [refer:referTo:](#)  
Refer the current call to another one.
  
- (int) - [attendedRefer:replaceSessionId:referTo:](#)  
Make an attended refer.
- (long) - [acceptRefer:referSignaling:](#)  
Once the REFER request accepted, a new call will be made if called this function. The function is usually called after onReceivedRefer callback event.
- (int) - [rejectRefer:](#)  
Reject the REFER request.
- (int) - [enableSendPcmStreamToRemote:state:streamSamplesPerSec:](#)  
Enable the SDK to send PCM stream data to remote side from another source instead of microphone.
- (int) - [sendPcmStreamToRemote:data:](#)  
Send the audio stream in PCM format from another source instead of audio device capturing (microphone).
- (int) - [enableSendVideoStreamToRemote:state:](#)  
Enable the SDK to send video stream data to remote side from another source instead of camera.
- (int) - [sendVideoStreamToRemote:data:width:height:](#)  
Send the video stream to remote side.
- (int) - [setRtpCallback:](#)  
Set the RTP callbacks to allow to access the sent and received RTP packets.
- (int) - [enableAudioStreamCallback:enable:callbackMode:](#)  
Enable/disable the audio stream callback.
- (int) - [enableVideoStreamCallback:callbackMode:](#)  
Enable/disable the video stream callback.
- (int) - [enableVideoDecoderCallback:](#)  
Enable/disable the video Decoder callback.
- (int) - [startRecord:recordFilePath:recordFileName:appendTimeStamp:audioFileFormat:audioRecordMode:aviFileCodecType:videoRecordMode:](#)  
Start recording the call.
- (int) - [stopRecord:](#)  
Stop recording.
- (int) - [playVideoFileToRemote:aviFile:loop:playAudio:](#)  
Play an AVI file to remote party.

- (int) - [stopPlayVideoFileToRemote:](#)  
*Stop playing video file to remote party.*
- (int) - [playAudioFileToRemote:filename:fileSamplesPerSec:loop:](#)  
*Play a wave file to remote party.*
- (int) - [stopPlayAudioFileToRemote:](#)  
*Stop playing wave file to remote party.*
- (int) - [playAudioFileToRemoteAsBackground:filename:fileSamplesPerSec:](#)  
*Play a wave file to remote party as conversation background sound.*
- (int) - [stopPlayAudioFileToRemoteAsBackground:](#)  
*Stop playing a wave file to remote party as conversation background sound.*
- (int) - [createConference:videoWidth:videoHeight:displayLocalVideo:](#)  
*Create a conference. It will fail if the existent conference does not end yet.*
- (void) - [destroyConference](#)  
*End the exist conference.*
- (int) - [setConferenceVideoWindow:](#)  
*Set the window for a conference, which is used to display the remote video image received.*
- (int) - [joinToConference:](#)  
*Join a session into existent conference. If the call is in hold, please un-hold it first.*
- (int) - [removeFromConference:](#)  
*Remove a session from an existent conference.*
- (int) - [setAudioRtcpBandwidth:BitsRR:BitsRS:KBitsAS:](#)  
*Set the audio RTCP bandwidth parameters as the RFC3556.*
- (int) - [setVideoRtcpBandwidth:BitsRR:BitsRS:KBitsAS:](#)  
*Set the video RTCP bandwidth parameters as the RFC3556.*
- (int) - [setAudioQos:DSCPValue:priority:](#)  
*Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for audio channel.*
- (int) - [setVideoQos:DSCPValue:](#)  
*Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for video channel.*
- (int) - [setVideoMTU:](#)  
*Set the MTU size for video RTP packet.*
- (int) -  
[getNetworkStatistics:currentBufferSize:preferredBufferSize:currentPacketLossRate:currentDiscardRate:currentExpandRate:currentPreemptiveRate:currentAccelerateRate:](#)  
*Get the "in-call" statistics. The statistics are reset after the query.*
- (int) - [getAudioRtpStatistics:averageJitterMs:maxJitterMs:discardedPackets:](#)  
*Obtain the RTP statistics of audio channel.*
- (int) -  
[getAudioRtcpStatistics:bytesSent:packetsSent:bytesReceived:packetsReceived:sendFractionLost:sendCumulativeLost:recvFractionLost:recvCumulativeLost:](#)  
*Obtain the RTCP statistics of audio channel.*
- (int) - [getVideoRtpStatistics:bytesSent:packetsSent:bytesReceived:packetsReceived:](#)  
*Obtain the RTP statistics of video.*
- (void) - [enableVAD:](#)  
*Enable/disable Voice Activity Detection (VAD).*
- (void) - [enableAEC:](#)  
*Enable/disable AEC (Acoustic Echo Cancellation).*
- (void) - [enableCNG:](#)  
*Enable/disable Comfort Noise Generator (CNG).*

- (void) - [enableAGC:](#)  
*Enable/disable Automatic Gain Control (AGC).*
- (void) - [enableANS:](#)  
*Enable/disable Audio Noise Suppression (ANS).*
- (int) - [sendOptions:sdp:](#)  
*Send OPTIONS message.*
- (int) - [sendInfo:mimeType:subMimeType:infoContents:](#)  
*Send an INFO message to remote side in dialog.*
- (long) - [sendMessage:mimeType:subMimeType:message:messageLength:](#)  
*Send a MESSAGE message to remote side in dialog.*
- (long) - [sendOutOfDialogMessage:mimeType:subMimeType:message:messageLength:](#)  
*Send an out of dialog MESSAGE message to remote side.*
- (int) - [presenceSubscribeContact:subject:](#)  
*Send a SUBSCRIBE message for presence to a contact.*
- (int) - [presenceAcceptSubscribe:](#)  
*Accept the presence SUBSCRIBE request that is received from contact.*
- (int) - [presenceRejectSubscribe:](#)  
*Reject a presence SUBSCRIBE request that is received from contact.*
- (int) - [presenceOnline:statusText:](#)  
*Send a NOTIFY message to contact to notify that presence status is online/changed.*
- (int) - [presenceOffline:](#)  
*Send a NOTIFY message to contact to notify that presence status is offline.*
- (BOOL) - [startKeepAwake](#)  
*Keep VoIP awake in the background. If you want your application to be able to receive the incoming call while it's running in background, you should call this function in applicationDidEnterBackground.*
- (BOOL) - [stopKeepAwake](#)  
*Keep VoIP awake in the background. Call this function in applicationWillEnterForeground once your application comes back to foreground.*

## Properties

- id< PortSIPEventDelegate > **delegate**

## Detailed Description

PortSIP VoIP SDK functions class.

### Author:

Copyright (c) 2006-2016 PortSIP Solutions,Inc. All rights reserved.

### Version:

11.2

### See also:

<http://www.PortSIP.com>

PortSIP SDK functions class description.

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

- PortSIPSDK.h





## PortSIPVideoRenderView Class Reference

PortSIP VoIP SDK Video Render View class.

```
#import <PortSIPVideoRenderView.h>
```

Inherits UIView.

### Instance Methods

- (void) - [initWithVideoRender](#)  
*Initialize the Video Render view. Render should be initialized before being used.*
- (void) - [releaseVideoRender](#)  
*Release the Video Render.*
- (void \*) - [getVideoRenderView](#)  
*Don't use this. Just call by SDK.*
- (void) - [updateVideoRenderFrame:](#)  
*Change the Video Render size.*

---

### Detailed Description

PortSIP VoIP SDK Video Render View class.

#### Author:

Copyright (c) 2006-2015 PortSIP Solutions,Inc. All rights reserved.

#### Version:

11.2.2

#### See also:

<http://www.PortSIP.com>

PortSIP VoIP SDK Video Render View class description.

---

### Method Documentation

#### - (void) updateVideoRenderFrame: (CGRect) frameRect

Change the Video Render size.

#### Remarks:

Example:

```
CGRect rect = videoRenderView.frame;
rect.size.width += 20;
rect.size.height += 20;

videoRenderView.frame = rect;
[videoRenderView setNeedsDisplay:YES];

CGRect renderRect = [videoRenderView bounds];
[videoRenderView updateVideoRenderFrame:renderRect];
```

---

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

- PortSIPVideoRenderView.h

# Index

INDEX