PortSIP VoIP SDK User Manual for Visual C++

Compatibles with Windows 2000/XP/Vista/7.

http://www.portsip.com

v7.1, Jan 16, 2012

NOTICE

For further information contact PortSIP Solutions, Inc.

Table of Contents

Overview, Q & A

This document describes the functions and callback events supported by PortSIP VoIP SDK libraries. It includes the SIP and XMPP SDKs.

All samples of the PortSIP VoIP SDK are available at
http://www.portsip.com/downloads.htm

The Sample source code for Visual C++ at
http://www.portsip.com/downloads/voip/VCSample.zip

Q:How to test the P2P call(without SIP server)?

A:

1) After you downloaded and extracted the SDK sample project ZIP file, then you can found a project which named P2PSample, just compile and run it.
2) Run the P2Psample.exe on two PCs, for example, run it on PC A and PC B, PC A IP address is 192.168.1.10, PC B IP address is 192.168.1.11.
3) Enter a user name and password on PC A, for example, user name is 111, password is aaa(you can enter anything for the password, the SDK will ignore it.). Enter a user name and password on PC B, for example: user name is 222, password is aaa.
4) Click the "Initialize" button on PC A and PC B. If the default port 5060 in using, then the P2PSample will said "Initialize failed". In this case please click the "Uninitialize" button and change the local port, then click the "Initialize" button again.
5) The log box will appears "Initialized." If the SDK initialize succeeded.
6) For make call from PC A to B, enter: sip:222@192.168.1.11 and click "Dial" button; For make call from PC B to A, enter: sip:111@192.168.1.10

Note: If changed the local sip port to other port, for example, the PC A using local port 5080, and the PC B using local port 6021, then:
For make call from PC A to B, enter: sip:222@192.168.1.11:6021 and click "Dial" button; For make call from PC B to A, enter: sip:111@192.168.1.10:5080 and click "Dial" button.

IMPORTANT:

P1 / P2 mismatch build issues with MSVC 2005

Question

I'm currently trying to build PortSIP Sample with MSVC 2005. It builds fine until the linking process starts and reports: "fatal error C1900: II mismatch between 'P1' version '20060201' and 'P2' version '20050411'". How can I resolve this issue?

[Answer]

The PortSIP SDK files and sample project has built by Visual Studio 2005 which has been installed the Microsoft® Visual Studio® 2005 Service Pack 1 and Visual Studio 2005 Service Pack 1 Update for Windows Vista

So if you using the Visual Studio 2005 without these two SP1, then you will encountering this issue, please download and install these two SP1 and rebuilt the sample project.

Section 1: The SIP Core SDK

Callback events:

```
typedef enum
{
    SIP_UNKNOWN = 0,
    SIP_REGISTER_SUCCESS = 1, // Register to SIP server succeeded
    SIP_REGISTER_FAILURE,     // Register to SIP server failed

    SIP_INVITE_INCOMING,      //  The call is incoming
    SIP_INVITE_TRYING,            //  The call is trying
    SIP_INVITE_RINGING,           //  The call is ringing
    SIP_INVITE_ANSWERED,      //  The callee has answered this call
    SIP_INVITE_FAILURE,           //  The call is failed
    SIP_INVITE_CLOSED,            //  The call is closed
    SIP_INVITE_UPDATED,           //  The remote party updated this call
    SIP_INVITE_UAS_CONNECTED, //  When the UAS receivd the ACK of 200 OK
    SIP_INVITE_UAC_CONNECTED, //  When the UAC sent the ACK for 200 OK
    SIP_INVITE_BEGINING_FORWARD,//  Whe the SDK begin forwarding the call

    SIP_REMOTE_HOLD,          //  The remote party has hold this call
    SIP_REMOTE_UNHOLD,            //  The remote party has take off the hold

    SIP_TRANSFER_TRYING,      //  The call transfer is trying
    SIP_TRANSFER_RINGING,     //  The transfer call is ringing
    SIP_PASV_TRANSFER_SUCCESS,    //  The passive transfer all is succeeded
    SIP_ACTV_TRANSFER_SUCCESS,    //  The active transfer call is succeeded
    SIP_PASV_TRANSFER_FAILURE,    //  The passive transfer call is failed
    SIP_ACTV_TRANSFER_FAILURE,    //  The active transfer call is failed

    SIP_RECV_PAGERMESSAGE,            //  Received a pager message
    SIP_RECV_BINARY_PAGERMESSAGE, //  Received a binary pager message
    SIP_SEND_PAGERMESSAGE_FAILURE,    //  Send the pager message failed
    SIP_SEND_PAGERMESSAGE_SUCCESS,    //  Send the pager message succeeded

    SIP_ARRIVED_SIGNALING,            //  This event will be fired when the SDK received every SIP message
```

```
    SIP_WAITING_VOICEMESSAGE,     //  If have waiting voice message, this event will be fired(MWI)
    SIP_WAITING_FAXMESSAGE,       //  If have waiting fax message, this event will be fired(MWI)

    SIP_RECV_DTMFTONE,            //  This event will be fired when received a DTMF tone

    SIP_PRESENCE_RECV_SUBSCRIBE,  //  The remote side request subscribe presence state
    SIP_PRESENCE_ONLINE,          //  The contact is go online
    SIP_PRESENCE_OFFLINE,         //  The contact is go offline

    SIP_RECV_OPTIONS,             //  Received an OPTIONS message out of dialog
    SIP_RECV_INFO,                //  Received a INFO message in dialog
    SIP_RECV_MESSAGE,             //  Received a MESSAGE message in dialog
    SIP_RECV_BINARY_MESSAGE,      //  Received a binary MESSAGE in dialog

}SIP_EVENT;
```

Example

A called B and the call was established(original call, between A and B), the A use the **PortSIP_refer** function transfer the B to C:

1: B received this transfer, B call to C. (refer call, between B and C)

2: A and B will got the **SIP_TRANSFER_TRYING** event.

3: When refer call the is ringing, A and B will got the **SIP_TRANSFER_RINGING** event

4: If refer call has failed(for example, the C rejected the call), A will got **SIP_ACTV_TRANSFER_FAILURE** event, B will got **SIP_PASV_TRANSFER_FAILURE** event.

5: If C answered the refer call, A will got **SIP_ACTV_TRANSFER_SUCCESS** event and hung up the conversation(original call, between A and B) automatically;

6: B will got the **SIP_PASV_TRANSFER_SUCCESS** event.

7: Transfer call successful, and there have the call between B and C only.

The PASV means passive.

The ACTV means active.

Audio and Video codecs definition

```
// Audio codec
typedef enum
{
  AUDIOCODEC_G723       = 4,     //8KHZ
  AUDIOCODEC_G729       = 18,    //8KHZ
  AUDIOCODEC_PCMA       = 8,     //8KHZ
  AUDIOCODEC_PCMU       = 0,     //8KHZ
  AUDIOCODEC_GSM        = 3,     //8KHZ
  AUDIOCODEC_G722       = 9,     //16KHZ
  AUDIOCODEC_ILBC       = 97,    //8KHZ
  AUDIOCODEC_AMR        = 98,    //8KHZ
  AUDIOCODEC_AMRWB      = 99,    //16KHZ
  AUDIOCODEC_SPEEX      = 100,   //8KHZ
  AUDIOCODEC_SPEEXWB    = 102,   //16KHZ
  AUDIOCODEC_G7221      = 121,   //16KHZ
  AUDIOCODEC_DTMF       = 101
}AUDIOCODEC_TYPE;
```

```
// Video codec
typedef enum
{
  VIDEO_CODEC_I420          = 113,
  VIDEO_CODEC_H263          = 34,
  VIDEO_CODEC_H263_1998     = 115,
  VIDEO_CODEC_H264          = 125
}VIDEOCODEC_TYPE;
```

Video resolution

```
typedef enum
{
  VIDEO_QCIF    =   1,      //  176X144     - for H263, H263-1998, H264
  VIDEO_CIF     =   2,      //  352X288     - for H263, H263-1998, H264
  VIDEO_VGA     =   3,      //  640X480     - for H264 only
  VIDEO_SVGA    =   4,      //  800X600     - for H264 only
  VIDEO_XVGA    =   5,      //  1024X768    - for H264 only
  VIDEO_720P    =   6,      //  1280X720    - for H264 only
  VIDEO_QVGA    =   7       //  320X240     - for H264 only
}VIDEO_RESOLUTION;
```

Audio recording format

```
typedef enum
{
  FILEFORMAT_WAVE = 1,
  FILEFORMAT_OGG,
  FILEFORMAT_MP3
}AUDIO_RECORDING_FILEFORMAT;
```

Audio and video stream callback mode

```
typedef enum
{
  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 base on the session ID
  AUDIOSTREAM_REMOTE_MIX,               //  Callback the received audio stream that mixed including all channels.
  AUDIOSTREAM_REMOTE_PER_CHANNEL,       //  Callback the received audio stream for one channel base on the session ID.
}AUDIOSTREAM_CALLBACK_MODE;
```

Audio and video stream callback function prototype

```
typedef long  (CALLBACK *fnAudioRawCallback)(void * obj, int sessionId, AUDIOSTREAM_CALLBACK_MODE type,
                        unsigned char * data, int dataLength,int samplingFreqHz);

typedef long (CALLBACK *fnVideoRawCallback)(void * obj, int sessionId, int width, int height,
                        unsigned char * data, int dataLength);
```

Play wave and AVI file finished callback function prototype

```
// Callback functioins for play wave file and AVI file to remote side
  typedef long (CALLBACK * fnPlayWaveFileToRemoteFinished)(void * obj, long sessionId, char * filePathName);
  typedef long (CALLBACK * fnPlayAviFileFinished)(void * obj, long sessionId);
```

Callback event Mechanism

SIP Core SDK using an abstract struct(SIPCallbackEvent) to implement callback event report. The abstract struct SIPCallbackEvent define as (In SIPEvent.hxx file):

```
struct SIPCallbackEvent
{
  virtual void onCommand(SIPCallbackCommand * command) = 0;
};
```

All callback events are report by **onCommand()** member function of the abstract struct **SIPCallbackEvent**, and the callback event details transferred by parameter **command** of **onCommand()**.

The **SIPCallbackCommand** class was define in the SIPEvent.hxx file .

User should derive and define onCommand() function to process callback events. such as:

```
struct MySIPEvent : public SIPCallbackEvent
{
   virtual void onCommand(SIPCallbackCommand * command)
   {
         if (!command)
         {
               return;
         }
   switch(command->getEventType())
   {
   case SIP_REGISTER_SUCCESS:
         m_Registered = true;
         break;

   case SIP_REGISTER_FAILURE:
      m_Registered = false;
         break;

   ……

   case SIP_SEND_PAGERMESSAGE_SUCCESS:
         onSIPSendPagerMessageSuccess(command);
         break;
   }
   PortSIP_delSIPCallbackCommand(command);
   command = NULL;
};
```

Initialize the SDK:

```
MySIPEvent * event = new MySIPEvent;
Void * SIPCore = PortSIP_initialize(event, ……);
```

SIP SDK Initialize and setting functions

```
void * PortSIP_initialize(SIPCallbackEvent * callbackEvent,
                          TRANSPORT_TYPE transportType,
                          PORTSIP_LOG_LEVEL appLogLevel,
                          const char * logFilePath,
                          int maximumLines,
                          const char * agent,
                          const char * STUNServer,
                          int STUNServerPort,
                          int * errorCode);
```

To initialize the SIP core SDK.

Parameters

callbackEvent	-	The callback event pointer.
transportType	-	Transport of SIP, tt can be set as TRANSPORT_UDP, TRANSPORT_TLS, TRANSPORT_TCP.
appLogLevel	-	Use to set the application log level, the levels defined in the PortSIPTypes.hxx file, if enable the log, then will generate `PORTSIP_APP_LOGFILE_xxxx` log file.
LogFilePath	-	The log file path, the path(folder) MUST is exists.
maximumLines	-	This parameter allows maximum 100 lines.
Agent	-	The User-Agent header to insert in messages.
STUNServer	-	Stun server, it's optional and can be pass NULL if you are not use STUN server.
STUNServerPort	-	Stun server port, it will be ignored if the STUNServer is empty.
errorCode	-	If the function succeeds, this parameter value is 0. If the function fails, this value is a specific error code can be found in PortSIP_Errors.hxx file.

Return Values

If the function succeeds, the return value is a handle to the Core SDK object. If the function fails, value is NULL.

Remarks

If want to use the TLS transport, you have to install the root certificate into the "Trusted Root Certification Authorities area." Usually the certificate was issued by your ITSP.

```
int PortSIP_setUserInfo(void * SIPCoreLib,
                  const char * userName,
                  const char * displayName,
                  const char * authName,
                  const char * password,
                  const char * localIP,
                  int localSIPPort,
                  const char * userDomain,
                  const char * SIPServer,
                  int SIPServerPort,
                  const char * outboundServer,
                  int outboundServerPort);
```

Set the user information into SDK.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

username - Account(User name) of the SIP, usually provided by an IP-Telephony service provider.

displayName - *The display name of user name, you can set it as your like, likes: "James Kend". It's optional.*

authName - Authorization user name (usually equals the username).

Password - Password.

localIP - The local computer IP address to bind (for example: 192.168.1.108), it will be using for send and receive SIP message and RTP packet. If pass this IP as the IPv6 format then the SDK will using IPv6.

localSIPPort - The SIP message transport listener port(for example: 5060).

userDomain - User domain; this parameter is optional and can be pass NULL if you are not use domain.

SIPServer - SIP proxy server IP or domain(for example: xx.xxx.xx.x or sip.xxx.com).

SIPServer Port - Port of the SIP proxy server, (for example: 5060).

outboundServer - Outbound proxy server (for example: sip.domain.com), it's optional and can be pass NULL if you are not use outbound server.

outboundServerPort - Outbound proxy server port, it will be ignored if the outboundServer is empty.

Return Values

If the function succeeds, the return value is 0. If the function fails, the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_unInitialize(Void * SIPCoreLib);
```

To un-initialize the SIP Core SDK and release resources.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

```
int PortSIP_getLocalIP(Void * SIPCoreLib, int index, char * IP, int ipLength);
```

To get local computer IP address

Parameters

index - The IP address index, for example, the PC has two NICs, we want to obtain the second NIC IP, then set this parameter 1. The first NIC IP index is 0.

IP - A pointer to the buffer that receives the IP

IPLength - The IP parameter buffer size, don't let it less than 16.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
bool Device_getLocalIP6(Void * SIPCoreLib, int index, char * IP, int ipLength);
```

To get local computer IP address in IPv6 format.

Parameters

index - The IP address index, for example, the PC has two NICss, we want to obtain the second NIC IP, then set this parameter 1. The first NIC IP index is 0.

IP - A pointer to the buffer that receives the IP

IPLength - The IP parameter buffer size, don't let it less than 64.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_setLicenseKey(Void * SIPCoreLib, const char * key);
```

Set the license key into SDK.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

key - The license key, provide by PortSIP Solutions, Inc. (http://www.portsip.com)

```
int PortSIP_getVersion(Void * SIPCoreLib, int * majorVersion, int * minorVersion);
```

The getVersion function returns the current version number of the SDK.

Parameters

marjorVersion - Pointer to a "int" data that the function fills with SDK major version number.

minorVersion - Pointer to a "int" data that the function fills with SDK minor version number.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_enableStackLog(Void * SIPCoreLib, const char * logFilePath,);
```

To enable the SIP stack log, it will generate a log file which named `PORTSIP_STACK_LOG_xxxx.log` in application directory.

Parameters

logFilePath - The log file path.

```
int  PortSIP_enableSessionTimer(void * SIPCoreLib, int timerSeconds);
```

This function allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending repeated INVITE requests. The repeated INVITE requests, or re-INVITEs, are sent during an active call leg to allow user agents (UA) or proxies to determine the status of a SIP session. Without this keepalive mechanism, proxies that remember 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.

Parameters

timerSeconds - The value of the refresh interval in seconds. Minimum requires 90 seconds.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_disableSessionTimer(void * SIPCoreLib);
```

Disable the session timer.

```
void PortSIP_setKeepAliveTime(void * SIPCoreLib, int keepAliveTime);
```

Set the SIP keep alive.

Parameters

keepAliveTime - This is the SIP keep alive time interval in seconds, if set to 0 then SDK will disable the SIP keep alive.

```
void PortSIP_enableCheckMwi(void * SIPCoreLib, bool state);
```

This method allows you enable/disable the SDK check MWI(Message Waiting Indication)

Parameters

state - If set it as true, then the SDK will check MWI once registered to a SIP proxy server; Set to false will not check it.

```
void PortSIP_detectMwi(void * SIPCoreLib);
```

This method will obtain the MWI status.

Remark

The MWI status will return by `SIP_WAITING_VOICEMESSAGE` and `SIP_WAITING_FAXMESSAGE` events.

```
void PortSIP_setSrtpPolicy(void * SIPCoreLib, SRTP_POLICY srtpPolicy);
```

Set the SRTP policy..

Parameters

srtpPolicy - The SRTP policy is defined as:

If set the policy as **SRTP_POLICY_NONE**, the SDK can receive the encrypted call(SRTP) and unencrypted call both, but can't place outgoing encrypted call.

If set the policy as **SRTP_POLICY_FORCE**, the SDK just allows receive encrypted Call and place outgoing encrypted call only.

If set the policy as **SRTP_POLICY_PREFER**, the SDK allows receive encrypted and uncrypted call, and allows place outgoing encrypted call and unencrypted call

```
int PortSIP_setRtpPortRange(Void * SIPCoreLib,
                            int minimumRtpAudioPort,
                            int maximumRtpAudioPort,
                            int minimumRtpVideoPort,
                            int maximumRtpVideoPort
                           );
```

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

Parameters

minimumRtpAudioPort - The minimum RTP port for audio stream.

maximumRtpAudioPort - The maximum RTP port for audio stream.

minimumRtpVideoPort - The minimum RTP port for video stream.

maxmumRtpVideoPort - The maximum RTP port for video stream.

Return Values

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

The ports range must is an even number, don't let the audio RTP port range lapped over with video RTP port range, otherwise the function will failed.

```
void PortSIP_setRtpKeepAlive(void * SIPCoreLib, bool state, int keepAlivePayloadType, int
deltaTransmitTimeMS);
```

To enable or disable the SDK send RTP keep-alive packet for RTP channel during the call is established.

Parameters

state - If set it as true, the SDK will send the keep-alive packet for RTP channel; Set to false will not send the keep-alive packets.

keepAlivePayloadType - The payload type of the keep-alive RTP packet, usually set to 126.

deltaTransmitTimeMS - The keep-alive RTP packet send interval, in millisecond, usually recommend 15000 – 300000.

```
int PortSIP_registerServer(void * SIPCoreLib, int expires);
```

Register to SIP proxy server.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

expires - Registration refresh Interval.

Return Values

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_unRegisterServer(Void * SIPCoreLib);
```

To un-register from the SIP proxy server.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

SIP SDK audio/video codec functions

```
void PortSIP_addAudioCodec(Void * SIPCoreLib, AUDIOCODEC_TYPE codecType);
```

To adding an audio codec.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

codecType - Audio codec type, it allows these values:

```
// Audio codec
typedef enum
{
   AUDIOCODEC_G723       = 4,//8KHZ
   AUDIOCODEC_G729       = 18,    //8KHZ
   AUDIOCODEC_PCMA       = 8,//8KHZ
   AUDIOCODEC_PCMU       = 0,//8KHZ
   AUDIOCODEC_GSM        = 3,//8KHZ
   AUDIOCODEC_G722       = 9,//16KHZ
   AUDIOCODEC_ILBC       = 97,    //8KHZ
   AUDIOCODEC_AMR        = 98,    //8KHZ
   AUDIOCODEC_AMRWB = 99,    //16KHZ
   AUDIOCODEC_SPEEX = 100,   //8KHZ
   AUDIOCODEC_SPEEXWB    = 102,   //16KHZ
   AUDIOCODEC_G7221 = 121,   //16KHZ
}
```

```
void PortSIP_addVideoCodec(void * SIPCoreLib, VIDEOCODEC_TYPE codecType);
```

To adding a video codec.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

codecType - Video codec type, it allows these values:

```
typedef enum
{
   VIDEOCODEC_H263             = 34,
   VIDEOCODEC_H263_1998        = 115,
   VIDEOCODEC_H264             = 125
}VIDEOCODEC_TYPE;
```

```
void PortSIP_clearAudioCodec(void * SIPCoreLib);
```

To remove all added audio codecs.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

```
void PortSIP_clearVideoCodec(void * SIPCoreLib);
```

To remove all added video codecs.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

```
bool PortSIP_isAudioCodecEmpty(void * SIPCoreLib);
```

Detect added audio codecs.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

Return Values

If no audio codec set, the return value is true, otherwise is false.

```
bool PortSIP_isVideoCodecEmpty(void * SIPCoreLib);
```

Detect added video codecs.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

Return Values

If no video codec set, the return value is true, otherwise is false.

SIP SDK audio/video device functions

```
void PortSIP_setAudioDeviceId(void * SIPCoreLib, DWORD recordingDeviceId, DWORD playoutDeviceId);
```

To set audio device to use for the audio call.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

recordingDeviceId - Device ID(index) for audio recording(Microphone)

playoutDeviceId - Device ID(index) for audio playback(Speaker)

Remark

Support switching audio device during the call.

```
void PortSIP_setVideoDeviceId(HANDLE SIPCoreLib, DWORD deviceId);
```

To set video device to use for the video call.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

deviceId - Device ID(index) for video capture, if without video device, then please set this parameter as ***PORTSIP_VIRTUAL_CAMERA_ID***. If you without video device(webcam), and want to use ***PortSIP_setPlayAVIFileToRemote***, then you have to called this function to set the ***PORTSIP_VIRTUAL_CAMERA_ID*** before you make the video call.

SIP SDK video setting functions

```
void PortSIP_setVideoBitrate(void * SIPCoreLib, int bitrateKbps);
```

Set video bit rate.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object.

bitrateKbps - The video bit rate in KBPS.

```
void PortSIP_setVideoFrameRate(void * SIPCoreLib, int frameRate);
```

Set the video frame rate. Usually you do not need to call this function set the frame rate, the SDK will use default frame rate.

Parameters

frameRate - The frame rate value, minimum is 5, maximum is 30. The bigger value will give you better video quality but require more bandwidth;

```
void PortSIP_setVideoResolution(void * SIPCoreLib, VIDEO_RESOLUTION resolution);
```

Set the video capture resolution.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

resolution - Video resolution, it's allows these values:

```
typedef enum
{
    VIDEO_QCIF    =    1,        //    176X144        - for H263, H263-1998, H264
    VIDEO_CIF     =    2,        //    352X288        - for H263, H263-1998, H264
    VIDEO_VGA     =    3,        //    640X480        - for H264 only
    VIDEO_SVGA    =    4,        //    800X600        - for H264 only
    VIDEO_XVGA    =    5         //    1024X768       - for H264 only
    VIDEO_720P    =    6,        //    1280X720       - for H264 only
    VIDEO_QVGA    =    7         //    320X240        - for H264 only
}VIDEO_RESOLUTION;
```

Some cameras haven't support SVGA and XVGA, 720P, please read your camera manual.

```
void PortSIP_setLocalVideoWindow(void * SIPCoreLib, void * localVideoWindow);
```

Set the the window that using to display the local video image.

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

localVideoWindow - The window to display local video image from camera.

```
int PortSIP_setRemoteVideoWindow(void * SIPCoreLib, long sessionId, void * remoteVideoWindow);
```

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

Parameters

SIPCoreLib - Handle to the Core SDK object. The PortSIP_initialize function returns this handle.

sessionId - The session ID

localVideoWindow - The window to display received remote video image.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_viewLocalVideo(void * SIPCoreLib, bool state);
```

To start/stop the camera to display the local video image.

Parameters

state - If the value is set to true, then show local video iamge. If set to false, stop show local video image.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_startVideoSending(void * SIPCoreLib, long sessionId, bool state);
```

To start/stop send the video stream to remote party.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

sessionId - Session ID of the call

state - If the value is set to true, then start send local video to remote party. If set to false, stop send video streaming.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK call functions

```
long PortSIP_call(void * SIPCoreLib, const char *  callee, bool sendSdp, int * errorCode);
```

Make the call(invite).

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

callee - The callee username(number), it can be "sip:usernumber@sipserver.com" or "usernumber"

sendSdp - To enable/disable send SDP with invite. If set to true, then invite will include SDP, if set to false, make invite without SDP.

errorCode - If the function succeeds, this parameter value is 0. If the function fails, this value is a specific error code can be found in PortSIP_Errors.hxx file.

Return Value

If the function succeeds, the return value is a session ID to the call, otherwise is `INVALID_SESSION_ID(-1)`.

```
int PortSIP_rejectCall(void * SIPCoreLib, long sessionId, int code, const char * reason);
```

To reject incoming call.

Parameters

sessionId - Session ID of the call.

code - Rejected call status code.

reason - Rejected call status text(rejected reason).

Return Values

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_answerCall(void * SIPCoreLib, long sessionId);
```

Answer the incoming call.

Parameters

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_terminateCall(void * SIPCoreLib, long sessionId);
```

Terminate the conversation(Hang up the call).

Parameters

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_forwardCall(void * SIPCoreLib, long sessionId, const char * forwardTo);
```

Forward the call to another one when received the incoming call.

Parameters

sessionId - Session ID of the call

forwardTo - The call forwarding target.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_enableCallForwarding(void * SIPCoreLib, bool forBusyOnly, const char * forwardingTo);
```

Set the call forwarding.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

forBusyOnly - If set this parameter as true, the SDK will forward all incoming calls when currently it's busy. *If set this as false, the SDK will forward all inconing calls anyway*.

forwardingTo - The call forwarding target, it's must likes sip:xxxx@sip.portsip.com.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_disableCallForwarding(HANDLE SIPCoreLib);
```

Disable the call forwarding, the SDK will not forward any incoming call after this function was called.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

```
int PortSIP_updateInvite(void * SIPCoreLib, long sessionId);
```

To update a conversation.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

sessionId - Session ID of the call

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

For example, A called B with the audio only, and B answered A, there has an audio conversation between with A, B. Now A want to saw B video image, A use these functions to do it:

The audio conversation is established, A want to update the INVITE for video to saw B video image.

```
PortSIP_clearVideoCodec(m_SIPLib);
PortSIP_addVideoCodec(m_SIPLib, VIDEOCODEC_H264);
PortSIP_updateInvite(m_SIPLib, sessionId);
```

```
int PortSIP_hold(void * SIPCoreLib, long sessionId);
```

To place a call on call.

Parameters

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_unHold(void * SIPCoreLib, long sessionId);
```

Take off hold.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_refer(void * SIPCoreLib, long sessionId, const char * referTo);
```

Refer the call to another one.

Parameters

sessionId - Session ID of the call.

referTo - Target of the refer, it can be "sip:number@sipserver.com" or "number" only

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Example: transfer the call to sip:testuser12@sip.portsip.com:8000

PortSIP_refer(mSIPLib, sessionId, "sip:testuser12@sip.portsip.com:8000");

Or transfer the call to sip:test1@176.11.90.202:

PortSIP_refer(mSIPLib, sessionId, "sip:test1@176.11.90.202");

You can download the demo AVI at: http://www.portsip.com/downloads/voip/blindtransfer.rar, please use the Windows Media Player to play the AVI file after extracted, it will shows how to do the transfer.

```
int PortSIP_attendedRefer(void * SIPCoreLib, long sessionId, long replaceSessionId, const char *
referTo);
```

To make an attended refer.

Parameters

sessionId - Session ID of the call.

replaceSessionId - Session ID of the replace call.

referTo - Target of the refer, it can be "sip:number@sipserver.com" or "number" only

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Please view the sample project to got more details. Or download the demo AVI at: http://www.portsip.com/downloads/voip/attendedtransfer.rar, please use the Windows Media Player to play the AVI file after extracted, it will shows how to do the attended transfer.

SIP SDK RTP statistics functions

```
int PortSIP_getAudioRtpStatistics(void * SIPCoreLib,
                                  long sessionId,
                                  long * averageJitterMs,
                                  long * maxJitterMs,
                                  long * discardedPackets);
```

Obtain the RTP statisics of audio.

Parameters

sessionId - Session ID of the call

`averageJitterMs` - Short-time average jitter (in milliseconds).

`maxJitterMs` - Maximum short-time jitter (in milliseconds).

`discardedPackets` - The number of discarded packets on a channel during the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getAudioRtcpStatistics(void * SIPCoreLib,
                                   long sessionId,
                                   long * bytesSent,
                                   long * packetsSent,
                                   long * bytesReceived,
                                   long * packetsReceived);
```

Obtain the RTCP statisics of audio channel.

Parameters

sessionId - Session ID of the call

bytesSent - The number of sent bytes.

packetsSent - The number of sent packets.

bytesReceived - The number of received bytes.

packetsReceived - The number of received packets.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getVideoRtpStatistics(void * SIPCoreLib,
                                  long sessionId,
                                  long * bytesSent,
                                  long * packetsSent,
                                  long * bytesReceived,
                                  long * packetsReceived);
```

Obtain the RTP statisics of video channel.

Parameters

sessionId - Session ID of the call

bytesSent - The number of sent bytes.

packetsSent - The number of sent packets.

bytesReceived - The number of received bytes.

packetsReceived - The number of received packets.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK audio effect functions

```
void PortSIP_enableAEC(HANDLE SIPCoreLib, bool state);
```

To enable/disable AEC (Acoustic Echo Cancellation).

Parameters

state - If the value is set to true, then AEC is enabled. If set to false, AEC is disabled

```
void PortSIP_enableVAD(HANDLE SIPCoreLib, bool state);
```

To enable/disable Voice Activity Detection(VAD).

Parameters

state - If the value is set to true, then VAD is enabled. If set to false, VAD is disabled.

```
void PortSIP_enableCNG(HANDLE SIPCoreLib, bool state);
```

To enable/disable Comfort Noise Generator(CNG).

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

state - If the value is set to true, then CNG is enabled. If set to false, CNG is disabled.

```
void PortSIP_enableAGC(HANDLE SIPCoreLib, bool state);
```

To enable/disable Automatic Gain Control(AGC).

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

state - If the value is set to true, then AGC is enabled. If set to false, AGC is disabled.

```
int PortSIP_setAudioSamples(void * SIPCoreLib, int samples);
```

This function set the audio capture sample which will be appears in the SDP of INVITE and 200 OK message.

Parameters

sample - It's should be a multiple of 10, and between 10 and 100(included 10 and 100).

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK DTMF functions

```
void PortSIP_setDTMFSamples(void * SIPCoreLib, int samples);
```

To set the samples of DTMF tone.

Parameters

samples - Samples of the DTMF tone, default is 160.

```
void PortSIP_enableDTMFOfRFC2833(void * SIPCoreLib, int RTPPayloadOf2833);
```

To enable the SDK send DTMF tone as RTP event(RFC2833).

Parameters

RTPPayloadOf2833 - Usually this value is 101(the default value is 101)

```
void PortSIP_enableDTMFOfInfo(void * SIPCoreLib);
```

To enable the SDK send DTMF tone as SIP INFO method.

```
int PortSIP_sendDTMF(void * SIPCoreLib, long sessionId, char code);
```

Send DTMF tone.

Parameters

sessionId - Session ID of the call.

code - DTMF Digit (1, 2, 3, 4, 5, ... 0, *, #)

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

For example send the # DTMF tone: ***PortSIP_sendDtmf(mSIPCoreLib, 1, '#');***

SIP SDK audio/video recording functions

```
int PortSIP_startAudioRecording(void * SIPCoreLib,
                                long sessionId,
                                const char * filePathName,
                                AUDIO_RECORDING_FILEFORMAT fileFormat);
```

Start record the voice of a call.

Parameters

sessionId - Session ID of the call.

filePathName - Recording file name, for example: audiofile.wav

fileFormat - Record file format, if pass it as **FILEFORMAT_WAVE**, then will be record the audio into a file as WAV format; If pass it as **FILEFORMAT_OGG**, then will be as OGG format; Set to **FILEFORMAT_MP3**, then record as MP3 file.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_stopAudioRecording(void * SIPCoreLib, long sessionId);
```

Stop the audio call recording.

Parameters

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_startVideoRecording(void * SIPCoreLib,
                                long sessionId,
                                const char * filePathName,
                                VIDEOCODEC_TYPE codecType);
```

Start record the video of a call.

Parameters

sessionId - Session ID of the call.

filePathName - Recording file name, for example: audiofile.avi

codecType - The SDK will compress the video data before write into the record file, this parameter allows you special *the compress codec, allows below values*:

```
VIDEO_CODEC_I420          = 113,   //  No compress
VIDEO_CODEC_H263          = 34,    //  Compress by H.263 codec
VIDEO_CODEC_H263_1998     = 115,   //  Compress by H.263-1998 codec
VIDEO_CODEC_H264          = 125    //  Compress by H.264 codec
```

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_stopVideoRecording(void * SIPCoreLib, long sessionId);
```

Stop the video call recording.

Parameters

sessionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK speaker/microphone mute functions

```
void PortSIP_muteMicrophone(void * SIPCoreLib, bool mute);
```

To mute the microphone.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

mute - If the value is set to true, then mute the microphone, the remote party can't heaing. If set to false, un-mute microphone.

```
void PortSIP_muteSpeaker(void * SIPCoreLib, bool mute);
```

To mute the speaker.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

mute - If the value is set to true, then mute the microphone. If set to false, un-mute speaker.

SIP SDK access SIP message header functions

```
int PortSIP_getExtensionHeaderValue(void * SIPCoreLib,
                                    char * sipMessage,
                                    const char * headerName,
                                    char * headerValue,
                                    int headerValueLength);
```

This function allows access the SIP header of SIP message.

Parameters

sipMessage - The SIP message

headerName - Which header want to access of the SIP message.

HeaderValue - A pointer to the buffer that receives the value of the SIP message header.

headerValueLength - The headerValue buffer size. Usually we recommended set it more than 512 bytes.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

If you received SIP message in ***SIP_ARRIVED_SIGNALING*** event, and want to got the header value of SIP message, then you can use this function to do it:

```
char value[ET_BUFFER_SIZE] = { 0 };
char message[SIP_MESSAGE_MAXSIZE] = { 0 };
command->getSignaling(message, SIP_MESSAGE_MAXSIZE);
PortSIP_getExtensionHeaderValue(m_SIPLib, message, "CSeq", value, ET_BUFFER_SIZE);
```

```
int PortSIP_addExtensionHeader(void * SIPCoreLib, const char * headerName, const char *
headerValue);
```

This function allows to add the extension header(custom header) into every outgoing SIP message.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

headerName - The custom header name which will be appears in every outgoing SIP message.

headerValue - The custom header value.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in

PortSIP_Errors.hxx file.

Example

For example, now we want to add a custom header which name is "Billing" into every outging SIP message:

PortSIP_addExtensionHeader(mPortSIPCore, "Billing", "usd100.00");

```
void PortSIP_clearAddExtensionHeaders(HANDLE SIPCoreLib);
```

To clear the added extension headers(custom headers), if called this function, then the added extension headers will not appear in every outgoing SIP message.

Example

For example, we have added two custom headers into every outging SIP message:

PortSIP_addExtensionHeader(mPortSIPCore, "Billing", "usd100.00");

PortSIP_addExtensionHeader(mPortSIPCore, "ServiceId", "8873456");

Now we want to do not adding these two headers, then:

PortSIP_clearAddextensionHeaders(mPortSIPCore);

```
int PortSIP_modifyHeaderValue(void * SIPCoreLib, const char * headerName, const char *
headerValue);
```

This function will modify the special SIP header value for every outgoing SIP message.

Parameters

headerName - The SIP header name which will be modify its value.

headerValue - The header value.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Example

Now we want to modify "Expires" header and "User-Agent" headver value for every outging SIP message:

PortSIP_ modifyHeaderValue(mPortSIPCore, "Expires", "1000");

PortSIP_ modifyHeaderValue(mPortSIPCore, "User-Agent", "MyTest Softphone 1.0");

```
void PortSIP_clearModifyHeaders(HANDLE SIPCoreLib);
```

To clear the modify headers value, if called this function, then the SDK will not modify every outgoing SIP message header values.

Example

We have modified two headers value for every outging SIP message:

PortSIP_modifyHeaderValue(mPortSIPCore, "Expires", "1000");

PortSIP_ modifyHeaderValue(mPortSIPCore, "User-Agent", "MyTest Softphone 1.0");

Now we want to do not modify them, then:

PortSIP_cleaModifyHeaders(mPortSIPCore);

SIP SDK QoS functions

```
int PortSIP_setAudioQos(void * SIPCoreLib, bool enable, int DSCPValue, int priority);
```

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

Parameters

DSCPValue - The six-bit DSCP value. Valid range is 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.

enable - Set as true to enable QoS, false to disable.

Priority - The 802.1p priority (PCP) field in a 802.1Q/VLAN tag. Values 0 – 7 set the priority, value -1 leaves the priority setting unchanged.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Setting the DSCP value on Windows requires that the executable runs with Administrator privileges. The DSCP value will not be modified unless this condition is fulfilled.

```
int PortSIP_setVideoQos(void * SIPCoreLib, bool enable, int DSCPValue);
```

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

Parameters

DSCPValue - The six-bit DSCP value. Valid range is 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.

enable - Set as true to enable QoS, false to disable.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Setting the DSCP value on Windows requires that the executable runs with Administrator privileges. The DSCP value will not be modified unless this condition is fulfilled.

SIP SDK audio dynamic volume functions

```
void PortSIP_getDynamicVolumeLevel(void * SIPCoreLib, int * speakerVolume, int *
microphoneVolume);
```

This function allows obtain the dynamic speaker and microphone volume level in the call. Usually set a timer to call this function to refresh the volume level indicator.

Parameters

speakerVolume - Return the dynamic speaker volume by this parameter, the range is 0 – 9.

MicrophoneVolume - Return the dynamic microphone volume by this parameter, the range is 0 – 9.

SIP SDK codec parameter functions

```
int  PortSIP_setAudioCodecParameter(void * SIPCoreLib, AUDIOCODEC_TYPE codecType, const char *
parameter);
```

This function use to set the audio codec parameter.

Parameters

codecType - The codec type which you want to set the parameter to it.

parameter - The parameter in string format.

Remark

PortSIP_ setAudioCodecParameter(AUDIOCODEC_AMR, “mode-set=0; octet-align=1; robust-sorting=0”);

```
int  PortSIP_setVideoCodecParameter(void * SIPCoreLib, VIDEOCODEC_TYPE codecType, const char *
parameter);
```

This function use to set the video codec parameter.

Parameters

codecType - The codec type which you want to set the parameter to it.

parameter - The parameter in string format.

Remark

PortSIP_ setVideoCodecParameter(VIDEOCODEC_H264, “profile-level-id=420033; packetization-mode=0”);

SIP SDK send audio PCM stream functions

```
bool PortSIP_enableSendPcmStreamToRemote(void * SIPCoreLib, long sessionId, bool state, int
streamSamplesPerSec);
```

Enable the SDK send PCM stream data to remote side, if want to send the PCM stream data to another side, MUST called this function first.

Parameters

sessionId - The session ID of call conversation.

State - Send state, true means allows send the audio steam, the false means disabled.

streamSamplesPerSec – The PCM stream data sample in seconds, for example: 8000 or 16000.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_sendPcmStreamToRemote(void * SIPCoreLib, long sessionId, const unsigned char * data,
int dataLength);
```

Send the audio stream directly in PCM format to instead of audio device capture(microphone). Before called this funtion, you MUST call the enableSendPcmStreamToRemote function.

Parameters

sessionId - Session ID of the call conversation.

data - The PCM audio stream data, must is 16bit, mono.

dataLength - The size of data.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Usually we should use it likes:

PortSIP_enableSendPcmStreamToRemote(sessionId, true, 16000);

PortSIP_sendPcmStreamToRemote(sessionId, data, dataSize);

SIP SDK play wave/avi file functions

```
void PortSIP_setPlayWaveFileToRemote(void * SIPCoreLib,
                                     long sessionId,
                                     const char * waveFile,
                                     bool enableState,
                                     int playMode,
                                     int fileSamplesPerSec,
                                     void * callbackObj,
                                     fnPlayWaveFileToRemoteFinished fnFinished);
```

Play a wave file to remote party, the remote side will be hearing this wave file once the call is established.

Parameters

sessionId - Session ID of the call.

waveFile - Wave file name, likes: c:\\sample.wav

enableState - If set as true, then allows SDK play the wave file. Set false to disable play wave file.

playMode - If set to 0, will send silence data to remote after wave file is end. If set to 1, play the wave file as repeat.

obj - This parameter allows pass as a object , it can be received in the callback function.

fileSamplePerSec – The wave file sample in seconds, should be 8000 or 16000.

fnPlayWaveFileToRemoteFinished – Callback function, it's defined as:

```
typedef long (CALLBACK * fnPlayWaveFileToRemoteFinished)(void * obj,
                                                         long sessionId,
                                                         char * filePathName);
```

Note: This first parameter ***obj*** is same as passed to ***PortSIP_setPlayWaveFileToRemote*** function.

The ***fileName*** parameter is the wave file name.

If set this callback function to NULL, then SDK does not fire the callback function.

Remark

The wave file format must is 8000Hz or16000Hz, 16bit, mono.

If you want the SDK continue send voice from microphone after the file play finished, you should call this function again and pass the "enableState" as false to stop the play.

```
int PortSIP_setPlayAviFileToRemote(void * SIPCoreLib,
                                   long sessionId,
                                   const char * aviFile,
                                   bool enableState,
                                   bool loop,
                                   bool playAudio,
                                   void * callbackObj,
                                   fnPlayAviFileFinished fnFinished);
```

Set an AVI file to play to remote party, the remote side will be see this AVI file iamge once the call is established.

Parameters

sessionId - Session ID of the call.

aviFile - AVI file name, likes: c:\\sample.avi

enableState - If set as true, then allows SDK play the AVI file. Set false to disable play AVI file.

loop - If set to false will stop play AVI file when it is end. Set to true will repeat play AVI file.

playAudio - If set to true then will play audio and video together, set to false will play video only.

callbackObj - This parameter allows pass as a object , it can be received in the callback function.

fnPlayAviFileToRemoteFinished – Callback function, it's defined as:

```
typedef long (CALLBACK * fnPlayAviFileFinished)(void * obj, long sessionId);
```

Note: This first parameter ***obj*** is same as passed to ***PortSIP_setPlayAVIFileToRemote*** function.

The ***sessionId*** parameter is the session which is play the AVI file.

If set this callback function to NULL, then SDK does not fire the callback function.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

The AVI file format must is i420 or H.263, H.263+, H.264..

If you without video device(webcam), and want to use ***PortSIP_setPlayAVIFileToRemote***, then you have to called ***PortSIP_setVideoDeviceId*** function to set the ***PORTSIP_VIRTUAL_CAMERA_ID*** before you make the video call.

SIP SDK audio and video callback functions

```
int PortSIP_enableAudioStreamCallback(void * SIPCoreLib,
                                      int sessionId,
                                      bool enable,
                                      AUDIOSTREAM_CALLBACK_MODE type,
                                      void * obj,
                                      fnAudioRawCallback callback);
```

To enable/disable the audio stream callback.

Parameters

sessionId - Session ID of the call.

enable - Enable/disable the callback.

type - Callback type, the values are defined in ***portsiptypes.hxx*** file.

```
typedef enum
{
//Callback the audio stream from microphone for all channels.
  AUDIOSTREAM_LOCAL_MIX,
//  Callback the audio stream from microphone for one channel base on the session ID
  AUDIOSTREAM_LOCAL_PER_CHANNEL,
//Callback the received audio stream that mixed including all channels.
  AUDIOSTREAM_REMOTE_MIX,
//  Callback the received audio stream for one channel base on the session ID.
  AUDIOSTREAM_REMOTE_PER_CHANNEL,
}AUDIOSTREAM_CALLBACK_MODE;
```

Obj - This parameter allows pass as a object , it can be received in the callback function.

fnAudioRawCallback – Callback function, it's defined as:

```
typedef long  (CALLBACK *fnAudioRawCallback)(void * obj,
                                             int sessionId,
                                             AUDIOSTREAM_CALLBACK_MODE type,
                                             unsigned char * data,
                                             int dataLength,
                                             int samplingFreqHz);
```

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_enableVideoStreamCallback(
                  HANDLE SIPCoreLib,
                  VIDEOSTREAM_CALLBACK_MODE type,
                  void * obj,
                  fnVideoRawCallback callback
                );
```

To enable/disable the audio stream callback.

Parameters

type - Callback type, the values are:

```
typedef enum
{
  VIDEOSTREAM_NONE = 0, // 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
}VIDEOSTREAM_CALLBACK_MODE;
```

obj - This parameter allows pass as a object , it can be received in the callback function.

fnVideoRawCallback – Callback function, it's defined as:

```
typedef long (*fnVideoRawCallback)(void * obj,
                                   int sessionId,
                                   int width,
                                   int height,
                                   unsigned char * data,
                                   int dataLength);
```

Note: This first parameter obj is same as passed to ***PortSIP_enableVideoStreamCallback*** function. The data parameter is video stream data, the dataLength is data length for bytes. The video data format it YUV420.

Remarks

If enable the video stream callback with VIDEOSTREAM_LOCAL parameter, in the callback funtion the sessionId parameter value always is 0; With VIDEOSTREAM_REMOTE, the sessionId is id of a call conversation; With VIDEOSTREAM_BOTH, when the local video is callback, the sesionId is 0; When the remote video is call back, the sessionId is id of a call conversation.

SIP SDK conference functions

```
int PortSIP_createConference(void * SIPCoreLib,
                             void * conferenceVideoWindow,
                             VIDEO_RESOLUTION videoResolution,
                             bool viewLocalVideo);
```

Create a conference. If the exists conference does not destroy yet, this function will failed.

Parameters

ConferenceWindow - The window which using to display the conference video.

videoResolution - The conference video resolution.

viewLocalVideo - set to true then the video image will include local video; Set to false will not display local video.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_createConferenceEx(void * SIPCoreLib,
                               void * conferenceVideoWindow,
                               VIDEO_RESOLUTION videoResolution,
                               bool viewLocalVideo,
                               long sessionIds[],
                               int sessionIdNums);
```

Create a conference with special sessions.

Parameters

ConferenceWindow - The window which using to display the conference video.

videoResolution - The conference video resolution.

viewLocalVideo - set to true then the video image will include local video; Set to false will not display local video.

sessionIds - The array of session IDs that want to add into conference.

sessionIdNums - The session IDs numbers of the sessionIds array parameter.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_destroyConference(HANDLE SIPCoreLib);
```

Destroy the exists conference.

Remark

After the conference is destroy, must place all calls in HOLD.

```
int PortSIP_joinToConference(HANDLE SIPCoreLib, long sessionId);
```

Join a session into an exist conference, the SDK will un-hold it automatically.

Parameters

seesionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_removeFromConference(HANDLE SIPCoreLib, long sessionId);
```

Remove a session from an exist conference.

Parameters

seesionId - Session ID of the call.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK OPTIONS/INFO/MESSAGE functions

```
int PortSIP_addSupportedMimeType(void * SIPCoreLib,
                                 const char * methodName,
                                 const char * mimeType,
                                 const char * subMimeType);
```

Set the SDK receive the SIP message that included special mime type.

Parameters

methodName - Method name of the SIP message, likes INVITE, OPTION, INFO, MESSAGE, UPDATE, ACK etc. *More details please read the RFC3261*.

mimeType - The mime type of SIP message.

subMimeType – The sub mime type of SIP message.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

Default, the PortSIP VoIP SDK support these media types(mime types) that 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 received SIP message that included above mime types. Now if remote side send a INFO SIP message, this messge “Content-Type” header value is “text/plain”, the SDK will rejected this INFO message. Because “text/plain” of INFO message does not included in the default support list.

Then how to let the SDK receive the SIP INFO message that included “text/plain” mime type ? We should use addSupportedMimyType to do it:

PortSIP_addSupportedMimeType(“INFO”, “text”, “plain”);

If want to receive the NOTIFY message with "application/media_control+xml", then:

PortSIP_addSupportedMimeType(“NOTIFY”, “application”, “media_control+xml”);

```
int PortSIP_sendInfo(void * SIPCoreLib,
                     long sessionId,
                     const char * mimeType,
                     const char * subMimeType,
                     const char * infoContents);
```

Send a INFO message to remote side in dialog.

Parameters

sessionId - Session ID of the call conversation.

mimeType - The mime type of INFO message.

subMimeType – The sub mime type of INFO message.

infoContents - What contents that you want to send with INFO message.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

With SIP message, usually it includes a header: "Content-Type", it consists of a mime type and a sub mime type. For example: application/sdp. The "application" is mime type, the "sdp" is sub mime type.

Example:

PortSIP_sendInfo(1, "text", "plain", "Hi, how are you?");

About the mime type details, please visit this website: http://www.iana.org/assignments/media-types/

```
int PortSIP_sendOptions(void * SIPCoreLib, const char * to, const char * sdp);
```

Send OPTIONS message.

Parameters

to - The receiver of OPTIONS message.

Sdp - If you want to send the OPTIONS message with SDP, then pass the SDP to this parameter, otherwise set it as emtpy.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_sendMessage(HANDLE SIPCoreLib,
                        long sessionId,
                        const char * mimeType,
                        const char * subMimeType,
                        const char * message);
```

Send a MESSAGE message to remote side in dialog.

Parameters

sessionId - Session ID of the call conversation.

mimeType - The mime type of MESSAGE message.

subMimeType – The sub mime type of MESSAGE message.

Message - What contents that you want to send with MESSAGE message.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

With SIP message, usually it includes a header: "Content-Type", it consists of a mime type and a sub mime type.

For example: application/sdp. The "application" is mime type, the "sdp" is sub mime type.

Example:

PortSIP_sendMessage(1, "text", "plain", "Hi, how are you?");

About the mime type details, please visit this website: http://www.iana.org/assignments/media-types/

```
int PortSIP_sendMessageEx(HANDLE SIPCoreLib,
                          long sessionId,
                          const char * mimeType,
                          const char * subMimeType,
                          const unsigned char * message,
                          int messageLength);
```

Send the binary message by a MESSAGE message to remote side in dialog.

Parameters

sessionId - Session ID of the call conversation.

mimeType - The mime type of MESSAGE message, currently just support mimeType as "application" only.

subMimeType – The sub mime type of MESSAGE message, currently just support "vnd.3gpp.sms" and "vnd.3gpp2.sms"..

Message - The contents that you want to send with MESSAGE message, it should be binary data.

MessageLength - The binary data length.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_sendOutOfDialogMessage(HANDLE SIPCoreLib,
                                   const char * to,
                                   const char * mimeType,
                                   const char * subMimeType,
                                   const char * message);
```

Send out of dialog message with special mime type.

Parameters

to - The pager message receiver. Likes sip:receiver@portsip.com

mimeType - The mime type of SIP message.

subMimeType – The sub mime type of SIP message.

message - The pager messge that need to send, usually we recommended it less than 2048 bytes.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_sendOutOfDialogMessageEx(HANDLE SIPCoreLib,
                                     const char * to,
                                     const char * mimeType,
                                     const char * subMimeType,
                                     const unsigned char * message,
                                     int messageLength);
```

Send binary data via message with special mime type.

Parameters

to - The pager message receiver. Likes sip:receiver@portsip.com

mimeType - The mime type of MESSAGE message, currently just support mimeType as “application” only.

subMimeType – The sub mime type of MESSAGE message, currently just support “vnd.3gpp.sms” and “vnd.3gpp2.sms”..

Message - The contents that you want to send with MESSAGE message, it should be binary data.

MessageLength - The binary data length.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_sendPagerMessage(void * SIPCoreLib, const char * to, const char * message);
```

Send pager message.

Parameters

to - The pager message receiver. Likes sip:receiver@portsip.com

message - The pager messge that need to send, usually we recommended it less than 2048 bytes.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK presence functions

```
int PortSIP_presenceSubscribeContact(HANDLE SIPCoreLib, const char * contact, const char * subject);
```

Send a SUBSCRIBE message for presence to a contact.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

contact - The target contact, it must likes sip:contact001@sip.portsip.com:

subject- - This subject text will be insert into the SUBSCRIBE message. For example:"Hello, I'm Jason".

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_presenceRejectSubscribe(HANDLE SIPCoreLib, long subscribeId);
```

Reject the presence SUBSCRIBE request which received from contact.

Parameters

subscribeId - Subscribe id, when received a SUBSCRIBE request from contactt, the callback event ***SIP_PRESENCE_RECV_SUBSCRIBE*** will be fired, this callback event has inclued the subscribe id.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_presenceAcceptSubscribe(HANDLE SIPCoreLib, long subscribeId);
```

Accept the presence SUBSCRIBE request which received from contact.

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

subscribeId - Subscribe id, when received a SUBSCRIBE request from contactt, the callback event ***SIP_PRESENCE_RECV_SUBSCRIBE*** will be fired, this callback event has inclued the subscribe id.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

Remark

After accepted contact's subscribe request, then should to use PortSIP_presenceOnline or PortSIP_presenceOffline to notify contact if your presence status was changed.

```
int PortSIP_presenceOffline(HANDLE SIPCoreLib, long subscribeId);
```

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

Parameters

SIPCoreLib - Handle to the SIP Core SDK object. The PortSIP_initialize function returns this handle.

subscribeId - Subscribe id, when received a SUBSCRIBE request from contactt, the callback event ***SIP_PRESENCE_RECV_SUBSCRIBE*** will be fired, this callback event has inclued the subscribe id.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_presenceOnline(HANDLE SIPCoreLib, long subscribeId, const char * stateText);
```

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

Parameters

subscribeId - Subscribe id, when received a SUBSCRIBE request from contactt, the callback event ***SIP_PRESENCE_RECV_SUBSCRIBE*** will be fired, this callback event has inclued the subscribe id.

stateText - Allows set your online status details, likes “on phone”, “do not disturb”, “away” etc.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

SIP SDK audio/video device manage functions

```
int PortSIP_getNumOfRecordingDevices(void * SIPCoreLib);
```

Gets the number of audio devices available for audio recording.

Return Value

The return value is number of recording devices, if failed the return value is less than 0 which is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getNumOfPlayoutDevices(void * SIPCoreLib);
```

Gets the number of audio devices available for audio playout.

Return Value

The return value is number of playout devices, if failed the return value is less than 0 which is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getRecordingDeviceName(void * SIPCoreLib,
                                   int index,
                                   char * nameUTF8,
                                   int nameUTF8Length);
```

Gets the name of a specific recording device given by an index.

Parameters

index - Device index (0, 1, 2, …, N-1), where N is given by `PortSIP_getNumOfRecordingDevices` (). Also -1 is a valid value and will return the name of the default recording device.

NameUTF8 - A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format.

NameUTF8Length – The size of nameUTF8 buffer, don't let it less than 128.

Return Value

If the function succeeds, the return value is 0, otherwise is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getPlayoutDeviceName(void * SIPCoreLib,
                                 int index,
                                 char * nameUTF8,
                                 int nameUTF8Length);
```

Gets the name of a specific playout device given by an index.

Parameters

index - Device index (0, 1, 2, …, N-1), where N is given by `PortSIP_getNumOfPlayoutDevices` (). Also -1 is a valid value and will return the name of the default playout device.

NameUTF8 - A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format.

NameUTF8Length – The size of nameUTF8 buffer, don't let it less than 128.

Return Value

If the function succeeds, the return value is 0, otherwise is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_setSpeakerVolume(void * SIPCoreLib, int volume);
```

Sets the speaker volume level.

Parameters

Volume – Volume level of speaker, valid range is 0 – 255.

Return Value

If the function succeeds, the return value is 0, otherwise is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getSpeakerVolume(void * SIPCoreLib);
```

Gets the speaker volume level.

Return Value

If the function succeeds, the return value is the volume, otherwise the return value is less than 0 which is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_setSystemOutputMute(void * SIPCoreLib, bool enable);
```

Mutes the speaker device completely in the OS.

Parameters

enable – If set to true, the speaker device is muted. If set to false, the speaker device is unmuted.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
bool PortSIP_getSystemOutputMute(void * SIPCoreLib);
```

This function retrieves the output device mute state in the operating system.

Return Value

If return the true, the output device is muted. If false, the output device is not muted.

```
int PortSIP_setMicVolume(void * SIPCoreLib, int volume);
```

Sets the microphone volume level.

Parameters

volume –The microphone volume level, the valid value is 0 – 255.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getMicVolume(void * SIPCoreLib);
```

This function retrieves the current microphone volume.

Return Value

If the function succeeds, the return value is the volume, otherwise the return value is less than 0 which is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_setSystemInputMute(void * SIPCoreLib, bool enable);
```

This function mutes the microphone input device completely in the OS.

Parameters

enable – If set to true, the microphone device is muted. If set to false, the microphone device is unmuted..

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
bool PortSIP_getSystemInputMute(void * SIPCoreLib);
```

Gets the mute state of the input device in the operating system.

Return Value

If return the true, the input device is muted. If false, the input device is not muted.

```
int PortSIP_playLocalWaveFile(void * SIPCoreLib,  const char * waveFile, bool loop);
```

This function plays wave file in locally.

Parameters

waveFile – The wave file which is should playing, if set to NULL then stop the playing.

loop – If set to true the wave file be play as repeat, set to false justp play once.

Return Value

If the function succeeds, the return value is 0, otherwise the return value is a specific error code can be found in PortSIP_Errors.hxx file.

```
void PortSIP_audioPlayLoopbackTest(void * SIPCoreLib, bool enable);
```

This function use to do the audio device loop back test.

Parameters

enable – Set to true start audio look back test; Set to fase to stop.

```
int PortSIP_getNumOfVideoCaptureDevices(void * SIPCoreLib);
```

Gets the number of available capture devices.

Return Value

The return value is number of video capture devices, if failed the return value is less than 0 which is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_getVideoCaptureDeviceName(void * SIPCoreLib,
                                      int index,
                                      char * uniqueIdUTF8,
                                      const unsigned int uniqueIdUTF8Length,
                                      char* deviceNameUTF8,
                                      const int deviceNameUTF8Length);
```

Gets the name of a specific video capture device given by an index.

Parameters

index - Device index (0, 1, 2, …, N-1), where N is given by `PortSIP_getNumOfVideoCaptureDevices ()`. Also -1 is a valid value and will return the name of the default capture device.

uniqueIdUTF8 - Unique identifier of the capture device.

UniqueIdUTF8Length - Size in bytes of uniqueIdUTF8.

NameUTF8 - A pointer to a character buffer to which the device name will be copied as a null-terminated string in UTF8 format.

NameUTF8Length – The size of nameUTF8 buffer, don't let it less than 128.

Return Value

If the function succeeds, the return value is 0, otherwise is a specific error code can be found in PortSIP_Errors.hxx file.

```
int PortSIP_showVideoCaptureSettingsDialogBox(void * SIPCoreLib,
                                              const char* uniqueIdUTF8,
                                              const int uniqueIdUTF8Length,
                                              const char* dialogTitle,
                                              void* parentWindow = NULL,
                                              const int x = 200,
                                              const int y = 200);
```

Displays the capture device property dialog box for the specified capture device.

Parameters

uniqueIdUTF8 - Unique identifier of the capture device.

uniqueIdUTF8Length - Size in bytes of uniqueIdUTF8.

DialogTitle –The title to use for the dialog.

parentWindow – Parent window to use for the dialog box, should originally be a HWND.

X - Horizontal position for the dialog relative to the parent window, in pixels.

Y - Vertical position for the dialog relative to the parent window, in pixels.

Return Value

If the function succeeds, the return value is 0, otherwise is a specific error code can be found in PortSIP_Errors.hxx file.

Section 2: The XMPP Core SDK

The XMPP Core SDK is help you to implement IM features as XMPP protocol.

Callback events:

```
typedef enum
{
  XMPP_UNKNOWN = 0,                 //  Unknown event, ignore it
  XMPP_LOGIN_SUCCESS = 1,           //  Logged in the JABBER server
  XMPP_LOGIN_FAILURE,               //  Logged faild

  XMPP_CONNECTION_ERROR,            //  Connection Error

  XMPP_REGISTER_ACCOUNT_SUCCESS,    //  Register a new account success
  XMPP_REGISTER_ACCOUNT_FAILURE,    //  Register a new account faild

  XMPP_RECV_MESSAGE,    // If you received a message from your buddy, then will got this event
  XMPP_SEND_MESSAGE_FAILURE,    // If you send a message failed, then will got this event

  XMPP_PRESENCE_REFRESH,        // If your buddy changes his status, then will got this event
  XMPP_PRESENCE_LOGOUT,         // The buddy who has logout


  XMPP_BUDDYLIST_ADD,               // Need to add a person into your buddy list
  XMPP_BUDDYLIST_DELETE,            // Need to delete a person from your buddy list

  XMPP_BUDDY_REQUEST_ADD,           // Someone request to add you
  XMPP_BUDDY_REQUEST_REJECT         // Someone reject you invite/Subscription
```

```
}XMPP_EVENT;
```

Remark:

For example, the XMPP_REGISTER_ACCOUNT_SUCCESS will be received when you register a new account succeeds. XMPP_REGISTER_ACCOUNT_FAILURE will be received when you register a new account failed.

Presence state:

```
typedef enum
{
  PRESENCEAvailable = 0,            // online and available.
  PRESENCEChat,                 // actively interested in chatting.
  PRESENCEAway,                 // temporarily away.
  PRESENCEDnd,                  // busy (dnd = "Do Not Disturb"). */
  PRESENCEXa,                   // away for an extended period (xa = "eXtended Away").
  PRESENCEUnavailable,           // offline.
  PRESENCEProbe,                // This is a presence probe.
  PRESENCEError,                // This is a presence error.
  PRESENCEInvalid               // The stanza is invalid.
}XMPP_PRESENCE;
```

TLS policy:

```
enum _TLSPolicy
{
  TLSDisabled = 0,   /**< Don't use TLS. */
  TLSOptional,  /**< Use TLS if compiled in and offered by the server. */
/**< Don't attempt to log in if the server didn't offer TLS
* or if TLS was not compiled in. */
  TLSRequired
};
```

Callback Event Mechanism:

XMPP Core SDK using an abstract struct(XMPPCallbackEvent) to implement callback event report. The abstract struct

XMPPCallbackEvent define as (In XMPPEvent.hxx file):

```
struct XMPPCallbackEvent
{
  virtual void onCommand(XMPPCallbackCommand * command) = 0;
};
```

All of the callback events was reports by **onCommand()** member function of the abstract struct **XMPPCallbackEvent**, and the callback events information transfered by parameter **command** of **onCommand()**.

The **XMPPCallbackEvent** class was define in the XMPPEvent.hxx file .

User should derive and define onCommand() function to process callback events. such as:

```
struct MyXMPPEvent : public XMPPCallbackEvent
{
  virtual void onCommand(XMPPCallbackCommand * command)
  {
      if (!command)
      {
          return;
      }

  switch(command->getEventType())
  {
  case XMPP_LOGIN_SUCCESS :
      m_Registered = true;
      break;
  case XMPP_LOGIN_FAILURE:
      break;
  ......
  case XMPP_BUDDY_REQUEST_ADD:
      onXMPPBuddyRequestAdd(command);
      break;
  }
  delete command;
};
```

```
MyXMPPEvent * event = new MyXMPPEvent;
HANDLE XMPPCore = PortXMPP_initialize(event, ......);
```

XMPP SDK initialize and login functions

```
HANDLE PortXMPP_initialize(
        XMPPCallbackEvent * callbackEvent,
        const char * userName,
        const char * password,
        const char * server,
        int port,
        const char * resource,
        _TLSPolicy tls
      );
```

To initialize the XMPP Core SDK.

Parameters

callbackEvent - The callback event pointer.

userName - Jabber account name, likes: testuser10

password - Password for the user name.

server - Jabber server, for example: jabber.com

port - Jabber server port, for example: 5222

resource - The Jabber client name, you can set it as you like.

tls - TLS options, if pass it as TLSDisabled, then SDK does not using TLS. If set as TLS optional, it will be depends on server, if server tequires the TLS, then will use TLS, otherwise, does not using TLS. Pass it as TLSRequired to force SDK using the TLS anyway.

Return Value

If the function succeeds, the return value is a handle to the XMPP Core SDK object. If the function fails, the return value is NULL.

```
void PortXMPP_unInitialize(HANDLE XMPPCoreLib);
```

To un-initialize the XMPP Core SDK and release resources.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

```
void PortXMPP_getVersion(HANDLE XMPPCoreLib, int * majorVersion, int * minorVersion);
```

The getVersion function returns the current version number of the SDK.

Parameters

marjorVersion - Pointer to a “int” data that the function fills with SDK major version number.

minorVersion - Pointer to a “int” data that the function fills with SDK minor version number.

```
bool PortXMPP_login(HANDLE XMPPCoreLib);
```

To login in server.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

Return Value

If the function succeeds, the return value is true, otherwise false.

```
void PortXMPP_unLogin(HANDLE XMPPCoreLib);
```

To logout in server.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize

XMPP SDK account manage functions

```
bool PortXMPP_createNewAccount(HANDLE XMPPCoreLib);
```

Use this function to register a new account from the server.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

Return Value

If the function succeeds, the return value is true, otherwise false.

```
void PortXMPP_changePassword(HANDLE XMPPCoreLib, const char * password );
```

Change current user password.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

password - New password.

XMPP SDK send message function

```
bool PortXMPP_sendMessage(HANDLE XMPPCoreLib, const char * toJid, const char * message);
```

Use this function to send text message to your buddy.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

toJid - Message receiver, likes: testuser2@jabber.com

message - Text message.

Return Value

If the function succeeds, the return value is true, otherwise false.

Remark

Exmaple: ***PortXMPP_sendMessage(CoreLib, "testuser2@jabber.com", "How are you?");***

XMPP SDK presence state function

```
void PortXMPP_setPresenceState(HANDLE XMPPCoreLib,XMPP_PRESENCE presence, const char * stateText);
```

Use this function to change your presence status and your buddy will saw it.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

presence - Set presence state, it can be using blow values:

```
typedef enum
{
    PRESENCEUnknown = 0,          // Unknown status.
    PRESENCEAvailable,            // online and available.
    PRESENCEChat,                 // actively interested in chatting.
    PRESENCEAway,                 // temporarily away.
    PRESENCEDnd,                  // busy (dnd = "Do Not Disturb"). */
    PRESENCEXa,                   // away for an extended period (xa = "eXtended Away").
    PRESENCEUnavailable           // offline.
}XMPP_PRESENCE;
```

stateText - The presence status text.

Remark

Exmaple: ***PortXMPP_ setPresenceState (CoreLib, "PRESENCEAway", "Be right back");***

XMPP SDK buddy manage functions

```
void PortXMPP_addBuddy(
        HANDLE XMPPCoreLib,
        const char * jid,
        const char * group,
        const char * message
    );
```

To add someone into your buddy list.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

jid - The buddy jid(xxxx@jabber.com) which need to adding into buddy list.

group - Buddy group

message - A message send to the buddy with the add invite.

Remark

For example, adding testuser3@jabber.com into “Friends” group

PortXMPP_addBuddy(CoreLib, “testuser3@jabber.com”, “Friends”, “Hello, I’m Jason”);

If just adding it without group:

PortXMPP_addBuddy(CoreLib, “testuser3@jabber.com”, NULL, NULL);

```
void PortXMPP_deleteBuddy(HANDLE XMPPCoreLib, const char * jid, bool both);
```

To delete the buddy from buddy list.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

Jid - The buddy jid(xxxx@jabber.com) which need to delete from buddy list.

both - If set this parameter as true, then will delete buddy from buddy list both side.

Remark

For example:

1: testuser2@jabber.com delete testuser3@jabber.com from buddy list.

PortXMPP_ deleteBuddy (CoreLib, "testuser3@jabber.com", false);

2: testuser2@jabber.com delete testuser3@jabber.com from buddy list and force testuser3@jabber.com delete testuser2@jabber.com from buddy list: ***PortXMPP_ deleteBuddy (CoreLib, "testuser3@jabber.com", true);***

```
void PortXMPP_rejectAddBuddyInvite(HANDLE XMPPCoreLib, const char * jid);
```

Use this function to rejected when received add buddy request from another one.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

jid - The buddy jid that you want to rejected.

Remark

For example:

The testuser7@jabber.com want to add testuser1@jabber.com into his buddy list, testuser1@jabber.com received this request and want to reject testuser7@jabber.com: ***PortXMPP_rejectAddBuddyInvite(CoreLib, "testuser7@jabber.com");***

```
void PortXMPP_acceptAddBuddyInvite(HANDLE XMPPCoreLib, const char * jid);
```

Use this function to accepted when received add buddy request from another one.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

jid - The buddy jid that you want to accepted.

Remark

For example:

The testuser7@jabber.com want to add testuser1@jabber.com into his buddy list, testuser1@jabber.com received this request and want to accept: ***PortXMPP_ acceptAddBuddyInvite (CoreLib, "testuser7@jabber.com");***

```
void PortXMPP_setLicenseKey(HANDLE XMPPCoreLib, const char * licenseKey);
```

Set the license key.

Parameters

XMPPCoreLib - Handle to the SIP Core SDK object. The PortXMPP_initialize function returns this handle.

Key - The license key, provide by PortSIP Solutions, Inc. (http://www.portsip.com)