

Ubiix VoIP SDK Manual for Windows

Version 16
Fri Feb 9 2018

Table of Contents

Welcome to Ubiix VoIP SDK.....	3
Getting Started.....	3
Contents.....	3
Website.....	3
Frequently Asked Questions.....	3
1. Where can I download the Ubiix VoIP SDK for test?.....	3
2. How can I compile the sample project?.....	3
3. How can I create a new project with Ubiix VoIP SDK?.....	3
4. How can I test the P2P call (without SIP server)?.....	4
5. Is the SDK is thread safe?.....	4
6. Does the SDK support native 64 bits?.....	5
Support.....	5
Module Index.....	6
Namespace Index.....	7
Class Index.....	8
Module Documentation.....	9
SDK functions.....	9
Initialize and register functions.....	9
NIC and local IP functions.....	11
Audio and video codecs functions.....	14
Additional setting functions.....	17
Audio and video functions.....	23
Access SIP message header functions.....	24
Call functions.....	28
Refer functions.....	33
Send audio and video stream functions.....	35
RTP packets, Audio stream and video stream callback functions.....	37
Record functions.....	39
Play audio and video file to remote functions.....	40
Conference functions.....	42
RTP and RTCP QOS functions.....	43
RTP statistics functions.....	44
Audio effect functions.....	46
Send OPTIONS/INFO/MESSAGE functions.....	49
Presence functions.....	52
Device Manage functions.....	54
SDK Callback events.....	58
Register events.....	59
Call events.....	59
Refer events.....	63
Signaling events.....	64
MWI events.....	65
DTMF events.....	66
INFO/OPTIONS message events.....	67
Presence events.....	68
Play audio and video file finished events.....	70
RTP callback events.....	71
Audio and video stream callback events.....	72
Ubiix.....	74
Class Documentation.....	79
Ubiix.Ubiix_Errors.....	79
Ubiix.UbiixLib.....	82
Ubiix.SIPCallbackEvents.....	89
Index.....	91

Welcome to Ubiix VoIP SDK

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

The rewarding [Ubiix](#) 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 [Ubiix](#) 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 Ubiix 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 [Ubiix](#) SDK: documentation, Dynamic/Static libraries, sources, headers, datasheet, and everything else a SDK user might need!

Website

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

Frequently Asked Questions

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

All sample projects of the Ubiix VoIP SDK can be found and downloaded at:
<http://www.Ubiix.com/downloads>
<http://www.Ubiix.com/Ubiix-voip-sdk>.

2. How can I compile the sample project?

1. Download the sample project from Ubiix website.
2. Extract the .zip file.
3. Open the project with your IDE:
C#, VB.NET, VC++: Visual Studio 2008 or higher.
Delphi: Delphi XE4 or higher.
4. Compile the sample project directly. The trial version SDK allows a 2-3 minutes conversation.

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

C#/VB.NET:
1) Download the Sample project and extract it for C#/VB.NET.
2) Create a new "Windows application" project.

```
3) Copy the Ubiix_sdk.dll to project output directory: bin and bin.
4) Copy the "Ubiix" folder to project folder and add into Solution.
5) Inherit the interface "SIPCallbackEvents" to process the callback events.
6) Right-click the project, choose "Properties". Click "Build" tab, and then check
the "Allow unsafe code" checkbox.
```

For more details please read the Sample project source code.

Delphi:

- 1) Download the Sample project and extract it.
- 2) Create a new "VCL Forms Application" project.
- 3) Copy the Ubiix_sdk.dll to project output directories.
- 4) Copy the "UbiixLib" folder to project folder and add into this new project.

For more details please read the Sample project source code.

VC++:

- 1) Download and extract the sample project.
- 2) Create a new "MFC Application" project.
- 3) Copy the Ubiix_sdk.dll to project output directories.
- 4) Copy the "include/UbiixLib" folder to project folder and add the ".hxx" files into project.
- 5) Copy the "lib" folder to project folder and link "Ubiix_sdk.lib" into project.

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

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

5. Is the SDK is 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 the "onAudioRawCallback", "onVideoRawCallback", "onReceivedRtpPacket", "onSendingRtpPacket" callbacks.

6. Does the SDK support native 64 bits?

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

Support

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

Module Index

Modules

Here is a list of all modules:

SDK functions.....	9
Initialize and register functions.....	9
NIC and local IP functions.....	11
Audio and video codecs functions.....	14
Additional setting functions.....	17
Audio and video functions.....	23
Access SIP message header functions.....	24
Call functions.....	28
Refer functions.....	33
Send audio and video stream functions.....	35
RTP packets, Audio stream and video stream callback functions.....	37
Record functions.....	39
Play audio and video file to remote functions.....	40
Conference functions.....	42
RTP and RTCP QOS functions.....	43
RTP statistics functions.....	44
Audio effect functions.....	46
Send OPTIONS/INFO/MESSAGE functions.....	49
Presence functions.....	52
Device Manage functions.....	54
 SDK Callback events.....	 58
Register events.....	59
Call events.....	59
Refer events.....	63
Signaling events.....	64
MWI events.....	65
DTMF events.....	66
INFO/OPTIONS message events.....	67
Presence events.....	68
Play audio and video file finished events.....	70
RTP callback events.....	71
Audio and video stream callback events.....	72

Namespace Index

Packages

Here are the packages with brief descriptions (if available):

Ubiix	74
-----------------------	----

Class Index

Class List

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

<u>Ubiix.Ubiix_Errors</u>	79
<u>Ubiix.UbiixLib</u>	82
<u>Ubiix.SIPCallbackEvents</u>	89

Module Documentation

SDK functions

Modules

- [Initialize and register functions](#)
 - [NIC and local IP functions](#)
 - [Audio and video codecs functions](#)
 - [Additional setting functions](#)
 - [Audio and video functions](#)
 - [Access SIP message header 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 file to remote functions](#)
 - [Conference functions](#)
 - [RTP and RTCP QOS functions](#)
 - [RTP statistics functions](#)
 - [Audio effect functions](#)
 - [Send OPTIONS/INFO/MESSAGE functions](#)
 - [Presence functions](#)
 - [Device Manage functions.](#)
-

Detailed Description

SDK functions

Initialize and register functions

Functions

- Int32 [Ubiix.UbiixLib.initialize](#) ([TRANSPORT_TYPE](#) transportType, String localIp, Int32 localSIPPort, [Ubiix_LOG_LEVEL](#) logLevel, String logFilePath, Int32 maxCallSessions, String sipAgentString, Int32 audioDeviceLayer, Int32 videoDeviceLayer, String TLSCertificatesRootPath, String TLSCipherList, Boolean verifyTLSCertificate)
Initialize the SDK.
 - void [Ubiix.UbiixLib.unInitialize](#) ()
Un-initialize the SDK and release resources.
 - Int32 [Ubiix.UbiixLib.getVersion](#) (out Int32 majorVersion, out Int32 minorVersion)
Get the current version number of the SDK.
 - Int32 [Ubiix.UbiixLib.setLicenseKey](#) (String key)
Set the license key. It must be called before setUser function.
-

Detailed Description

Initialize and register functions

Function Documentation

Int32 Ubiix.UbiixLib.initialize (TRANSPORT_TYPE transportType, String localIp, Int32 localSIPPort, Ubiix LOG_LEVEL logLevel, String logFilePath, Int32 maxCallSessions, String sipAgentString, Int32 audioDeviceLayer, Int32 videoDeviceLayer, String TLSCertificatesRootPath, String TLSCipherList, Boolean verifyTLSCertificate)

Initialize the SDK.

Parameters:

<i>transport</i>	Transport for SIP signaling. TRANSPORT_PERS is the Ubiix private transport for anti SIP blocking. It must be used with PERS.
<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 messages and RTP packets. If this IP is passed in IPv6 format, the SDK will be using 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>logLevel</i>	Set the application log level. The SDK will generate "Ubiix_Log_datetime.log" file if the log enabled.
<i>logFilePath</i>	The log file path. The path (folder) MUST be existent.
<i>maxCallLines</i>	Theoretically, unlimited lines could be supported depending on the device capability. For SIP client recommended value ranges 1 - 100;
<i>sipAgent</i>	The User-Agent header to be inserted into SIP messages.
<i>audioDeviceLayer</i>	Specify the audio device layer to 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.
<i>TLSCertificatesRootPath</i>	Specify the TLS certificate path, from which the SDK will load the certificates automatically. Note: On Windows, this path will be ignored, and SDK will read the certificates from Windows certificates stored area instead.
<i>TLSCipherList</i>	Specify the TLS cipher list. This parameter is usually passed as empty so that the SDK will offer all available ciphers.
<i>verifyTLSCertificate</i>	Indicate if SDK will verify the TLS certificate. By setting to false, the SDK will not verify the validity of TLS certificate.

Returns:

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

Int32 Ubiix.UbiixLib.getVersion (out Int32 majorVersion, out Int32 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 a specific error code.

Int32 Ubiix.UbiixLib.setLicenseKey (String key)

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

Parameters:

key	The SDK license key, please purchase from Ubiix .
-----	---

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

- Int32 [Ubiix.UbiixLib.getNICNums](#) ()
Get the Network Interface Card numbers.
- Int32 [Ubiix.UbiixLib.getLocalIpAddress](#) (Int32 index, StringBuilder ip, Int32 ipSize)
Get the local IP address by Network Interface Card index.
- Int32 [Ubiix.UbiixLib.setUser](#) (String userName, String displayName, String authName, String password, String sipDomain, String sipServerAddr, Int32 sipServerPort, String stunServerAddr, Int32 stunServerPort, String outboundServerAddr, Int32 outboundServerPort)
Set user account info.
- void [Ubiix.UbiixLib.removeUser](#) ()
Remove the user. It will un-register from SIP server given that the user is already registered.
- Int32 [Ubiix.UbiixLib.setInstanceId](#) (String uuid)
Set outbound (RFC5626) instanceId to be used in contact headers.
- Int32 [Ubiix.UbiixLib.registerServer](#) (Int32 expires, Int32 retryTimes)
Register to SIP proxy server (login to server).
- Int32 [Ubiix.UbiixLib.unregisterServer](#) ()
Un-register from the SIP proxy server.
- Int32 [Ubiix.UbiixLib.enableReliableProvisional](#) (Boolean enable)
Enable/disable PRACK.
- Int32 [Ubiix.UbiixLib.enable3GppTags](#) (Boolean enable)
Enable/disable the 3Gpp tags, including "ims.icsi.mmtel" and "g.3gpp.smsip".
- void [Ubiix.UbiixLib.enableCallbackSendingSignaling](#) (Boolean enable)
Enable/disable callback the sent SIP messages.
- void [Ubiix.UbiixLib.enableCallbackReceivedSignaling](#) (Boolean enable)
Enable/disable callback the received SIP messages.
- Int32 [Ubiix.UbiixLib.setRtpCallback](#) (Int32 callbackObject, Boolean enable)
Set the RTP callbacks to allow access to the sent and received RTP packets.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.getNICNums ()

Get the Network Interface Card numbers.

Returns:

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

Int32 Ubiix.UbiixLib.getLocalIpAddress (Int32 *index*, StringBuilder *ip*, Int32 *ipSize*)

Get the local IP address by Network Interface Card index.

Parameters:

<i>index</i>	The IP address index. For example, if the PC has two NICs, and we wish to obtain the second NIC IP. Set this parameter to 1 and the first NIC IP index is 0.
<i>ip</i>	The buffer that is used to receive the IP.
<i>ipSize</i>	The IP buffer size, which cannot be less than 32 bytes.

Returns:

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

Int32 Ubiix.UbiixLib.setUser (String *userName*, String *displayName*, String *authName*, String *password*, String *sipDomain*, String *sipServerAddr*, Int32 *sipServerPort*, String *stunServerAddr*, Int32 *stunServerPort*, String *outboundServerAddr*, Int32 *outboundServerPort*)

Set user account info.

Parameters:

<i>userName</i>	Account (User name) of the SIP. 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 pass this IP as the IPv6 format, the SDK will use IPv6.
<i>localSipPort</i>	The SIP message transport listener port. For example: 5060.
<i>userDomain</i>	User domain; this parameter is optional that allows to pass an empty string if you are not using the domain.
<i>sipServer</i>	SIP proxy server IP or domain. For example: xx.xxx.xx.x or sip.xxx.com.
<i>sipServerPort</i>	Port of the SIP proxy server. For example: 5060.
<i>stunServer</i>	Stun server used for NAT traversal. It's optional and can pass 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 a 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.

Int32 Ubiix.UbiixLib.setInstanceId (String *uuid*)

Set outbound (RFC5626) instanceId to be used in contact headers.

Parameters:

<i>uuid</i>	The ID that will appear in the contact header. Please make sure it's a unique ID.
-------------	---

Returns:

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

Int32 Ubiix.UbiixLib.registerServer (Int32 *expires*, Int32 *retryTimes*)

Register to SIP proxy server (login to server).

Parameters:

<i>expires</i>	Registration refresh Interval in seconds with maximum 3600. It will be inserted into SIP REGISTER message headers.
<i>retryTimes</i>	The retry times if failed to refresh the registration. If it's set to be less than or equal to 0, the retry will be disabled and onRegisterFailure callback triggered when retry failed.

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 succeeded, onRegisterSuccess will be triggered, otherwise onRegisterFailure triggered.

Int32 Ubiix.UbiixLib.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.

Int32 Ubiix.UbiixLib.enableReliableProvisional (Boolean *enable*)

Enable/disable PRACK.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.enable3GppTags (Boolean *enable*)

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

Parameters:

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

Returns:

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

void Ubiix.UbiixLib.enableCallbackSendingSignaling (Boolean *enable*)

Enable/disable callback the sent SIP messages.

Parameters:

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

void Ubiix.UbiixLib.enableCallbackReceivedSignaling (Boolean *enable*)

Enable/disable callback the received SIP messages.

Parameters:

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

Int32 Ubiix.UbiixLib.setRtpCallback (Int32 *callbackObject*, Boolean *enable*)

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

Parameters:

<i>callbackObject</i>	The callback object that you passed in can be accessed once the callback function triggered.
<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.

Audio and video codecs functions

Functions

- Int32 [Ubiix.UbiixLib.addAudioCodec](#) ([AUDIOCODEC_TYPE](#) codecType)
Enable an audio codec. It will be appears in SDP.
- Int32 [Ubiix.UbiixLib.addVideoCodec](#) ([VIDEOCODEC_TYPE](#) codecType)
Enable a video codec. It will appear in SDP.
- Boolean [Ubiix.UbiixLib.isAudioCodecEmpty](#) ()
Detect if enabled audio codecs is empty or not.
- Boolean [Ubiix.UbiixLib.isVideoCodecEmpty](#) ()
Detect if enabled video codecs is empty or not.

- Int32 [Ubiix.UbiixLib.setAudioCodecPayloadType](#) ([AUDIOCODEC_TYPE](#) codecType, Int32 payloadType)
Set the RTP payload type for dynamic audio codec.
- Int32 [Ubiix.UbiixLib.setVideoCodecPayloadType](#) ([VIDEOCODEC_TYPE](#) codecType, Int32 payloadType)
Set the RTP payload type for dynamic Video codec.
- void [Ubiix.UbiixLib.clearAudioCodec](#) ()
Remove all enabled audio codecs.
- void [Ubiix.UbiixLib.clearVideoCodec](#) ()
Remove all enabled video codecs.
- Int32 [Ubiix.UbiixLib.setAudioCodecParameter](#) ([AUDIOCODEC_TYPE](#) codecType, String parameter)
Set the codec parameter for audio codec.
- Int32 [Ubiix.UbiixLib.setVideoCodecParameter](#) ([VIDEOCODEC_TYPE](#) codecType, String parameter)
Set the codec parameter for video codec.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.addAudioCodec ([AUDIOCODEC_TYPE](#) codecType)

Enable an audio codec. It will be appears 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.

Int32 Ubiix.UbiixLib.addVideoCodec ([VIDEOCODEC_TYPE](#) codecType)

Enable a video codec. 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.

Boolean Ubiix.UbiixLib.isAudioCodecEmpty ()

Detect if enabled audio codecs is empty or not.

Returns:

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

Boolean Ubiix.UbiixLib.isVideoCodecEmpty ()

Detect if enabled video codecs is empty or not.

Returns:

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

Int32 Ubiix.UbiixLib.setAudioCodecPayloadType ([AUDIOCODEC_TYPE](#) codecType, Int32 payloadType)

Set the RTP payload type for dynamic audio codec.

Parameters:

<i>codecType</i>	Audio codec type, which is defined in the UbiixTypes 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 value a specific error code.

Int32 Ubiix.UbiixLib.setVideoCodecPayloadType ([VIDEOCODEC_TYPE](#) codecType, Int32 payloadType)

Set the RTP payload type for dynamic Video codec.

Parameters:

<i>codecType</i>	Video codec type, which is defined in the UbiixTypes 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.

Int32 Ubiix.UbiixLib.setAudioCodecParameter ([AUDIOCODEC_TYPE](#) codecType, String parameter)

Set the codec parameter for audio codec.

Parameters:

<i>codecType</i>	Audio codec type, defined in the UbiixTypes 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:

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

Int32 Ubiix.UbiixLib.setVideoCodecParameter ([VIDEOCODEC_TYPE](#) codecType, String parameter)

Set the codec parameter for video codec.

Parameters:

<i>codecType</i>	Video codec type, defined in the UbiixTypes 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:

```
setVideoCodecParameter(VIDEO_CODEC_H264, "profile-level-id=420033;  
packetization-mode=0");
```

Additional setting functions

Functions

- Int32 [Ubiix.UbiixLib.setSrtpPolicy](#) (SRTP_POLICY srtpPolicy, Boolean allowSrtpOverUnsecureTransport)
Set the SRTP policy.
- Int32 [Ubiix.UbiixLib.setRtpPortRange](#) (Int32 minimumRtpAudioPort, Int32 maximumRtpAudioPort, Int32 minimumRtpVideoPort, Int32 maximumRtpVideoPort)
Set the RTP ports range for audio and video streaming.
- Int32 [Ubiix.UbiixLib.setRtcpPortRange](#) (Int32 minimumRtcpAudioPort, Int32 maximumRtcpAudioPort, Int32 minimumRtcpVideoPort, Int32 maximumRtcpVideoPort)
Set the RTCP ports range for audio and video streaming.
- Int32 [Ubiix.UbiixLib.enableCallForward](#) (Boolean forBusyOnly, String forwardTo)
Enable call forward.
- Int32 [Ubiix.UbiixLib.disableCallForward](#) ()
Disable the call forwarding. The SDK is not forwarding any incoming call after this function is called.
- Int32 [Ubiix.UbiixLib.enableSessionTimer](#) (Int32 timerSeconds, [SESSION_REFRESH_MODE](#) refreshMode)
Allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending INVITE requests repeatedly.
- Int32 [Ubiix.UbiixLib.disableSessionTimer](#) ()
Disable the session timer.
- void [Ubiix.UbiixLib.setDoNotDisturb](#) (Boolean state)
Enable the "Do not disturb" to enable/disable.
- Int32 [Ubiix.UbiixLib.enableAutoCheckMwi](#) (Boolean state)
Allows to enable/disable the check MWI (Message Waiting Indication) automatically.
- Int32 [Ubiix.UbiixLib.setRtpKeepAlive](#) (Boolean state, Int32 keepAlivePayloadType, Int32 deltaTransmitTimeMS)
Enable or disable to send RTP keep-alive packet when the call is established.
- Int32 [Ubiix.UbiixLib.setKeepAliveTime](#) (Int32 keepAliveTime)
Enable or disable to send SIP keep-alive packet.
- Int32 [Ubiix.UbiixLib.getSipMessageHeaderValue](#) (String sipMessage, String headerName, StringBuilder headerValue, Int32 headerValueLength)
Access the SIP header of SIP message.
- Int32 [Ubiix.UbiixLib.addSipMessageHeader](#) (Int32 sessionId, String methodName, Int32 msgType, String headerName, String headerValue)
Add the SIP Message header into the specified outgoing SIP message.

- Int32 [Ubiix.UbiixLib.removeAddedSipMessageHeader](#) (Int32 sipMessageHeaderId)
Remove the headers (custom header) added by addSipMessageHeader.
- Int32 [Ubiix.UbiixLib.clearAddedSipMessageHeaders](#) ()
Clear the added extension headers (custom headers)
- Int32 [Ubiix.UbiixLib.modifySipMessageHeader](#) (Int32 sessionId, String methodName, Int32 msgType, String headerName, String headerValue)
Modify the special SIP header value for every outgoing SIP message.
- Int32 [Ubiix.UbiixLib.removeModifiedSipMessageHeader](#) (Int32 sipMessageHeaderId)
Remove the extension header (custom header) from every outgoing SIP message.
- Int32 [Ubiix.UbiixLib.clearModifiedSipMessageHeaders](#) ()
Clear the modified headers value, and do not modify every outgoing SIP message header values any longer.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.setSrtpPolicy ([SRTP_POLICY](#) *srtpPolicy*, Boolean *allowSrtpOverUnsecureTransport*)

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 value a specific error code.

Int32 Ubiix.UbiixLib.setRtpPortRange (Int32 *minimumRtpAudioPort*, Int32 *maximumRtpAudioPort*, Int32 *minimumRtpVideoPort*, Int32 *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.

Int32 Ubiix.UbiixLib.setRtcpPortRange (Int32 *minimumRtcpAudioPort*, Int32 *maximumRtcpAudioPort*, Int32 *minimumRtcpVideoPort*, Int32 *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.

Int32 Ubiix.UbiixLib.enableCallForward (Boolean *forBusyOnly*, String *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 like sip: xxxx@sip.Ubiix.com .

Returns:

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

Int32 Ubiix.UbiixLib.disableCallForward ()

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

Returns:

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

Int32 Ubiix.UbiixLib.enableSessionTimer (Int32 *timerSeconds*, [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 of 90 seconds required.
---------------------	---

<i>refreshMode</i>	Allow to set the session refresh by UAC or UAS: SESSION_REFRESH_UAC or SESSION_REFRESH_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 keepalive mechanism, proxies that remember incoming and outgoing requests (stateful proxies) may continue to retain call state in vain. If a UA fails to send a BYE message at the end of a session or if the BYE message is lost because of network problems, a stateful proxy will not know that the session has ended. The re-INVITES ensure that active sessions stay active and completed sessions are terminated.

Int32 Ubiix.UbiixLib.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 Ubiix.UbiixLib.setDoNotDisturb (Boolean 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.
--------------	--

Int32 Ubiix.UbiixLib.enableAutoCheckMwi (Boolean state)

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

Parameters:

<i>state</i>	If it is set as true, MWI will be checked 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.

Int32 Ubiix.UbiixLib.setRtpKeepAlive (Boolean state, Int32 keepAlivePayloadType, Int32 deltaTransmitTimeMS)

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

Parameters:

<i>state</i>	Set to true to allow to send the keep-alive packet during the conversation.
<i>keepAlivePayloadType</i>	The payload type of the keep-alive RTP packet. It's usually set to 126.
<i>deltaTransmitTimeMS</i>	The keep-alive RTP packet sending interval, in millisecond. Recommend 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.

Int32 Ubiix.UbiixLib.setKeepAliveTime (Int32 *keepAliveTime*)

Enable or disable to send SIP keep-alive packet.

Parameters:

<i>keepAliveTime</i>	This is the SIP keep alive time interval in seconds. Set it to 0 to disable the SIP keep alive. Recommend to set as 30 or 50.
----------------------	---

Returns:

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

Int32 Ubiix.UbiixLib.getSipMessageHeaderValue (String *sipMessage*, String *headerName*, StringBuilder *headerValue*, Int32 *headerValueLength*)

Access the SIP header of SIP message.

Parameters:

<i>sipMessage</i>	The SIP message.
<i>headerName</i>	The header which wishes to access the SIP message.
<i>headerValue</i>	The buffer to receive header value.
<i>headerValueLength</i>	The headerValue buffer size. Usually we recommend to set it more than 512 bytes.

Returns:

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

Remarks:

When receiving a SIP message in the onReceivedSignaling callback event, and wishes to get SIP message header value, please use getSipMessageHeaderValue:

```
StringBuilder value = new StringBuilder();  
value.Length = 512;  
getSipMessageHeaderValue(message, name, value);
```

Int32 Ubiix.UbiixLib.addSipMessageHeader (Int32 *sessionId*, String *methodName*, Int32 *msgType*, String *headerName*, String *headerValue*)

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

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.removeAddedSipMessageHeader (Int32 *sipMessageHeaderId*)

Remove the headers (custom header) added by `addSipMessageHeader`.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.clearAddedSipMessageHeaders ()

Clear the added extension headers (custom headers)

Remarks:

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

```
addExtensionHeader(-1, "ALL", 3, "Blling", "usd100.00");  
addExtensionHeader(-1, "ALL", 3, "ServiceId", "8873456");  
clearAddedSipMessageHeaders\(\);
```

Int32 Ubiix.UbiixLib.modifySipMessageHeader (Int32 *sessionId*, String *methodName*, Int32 *msgType*, String *headerName*, String *headerValue*)

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

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.removeModifiedSipMessageHeader (Int32 *sipMessageHeaderId*)

Remove the extension header (custom header) from every outgoing SIP message.

Parameters:

<i>modifiedSipMessageId</i>	The <code>modifiedSipMessageId</code> is returned by <code>modifySipMessageHeader</code> .
-----------------------------	--

Returns:

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

Int32 Ubiix.UbiixLib.clearModifiedSipMessageHeaders ()

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

Remarks:

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

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

Audio and video functions

Functions

- Int32 [Ubiix.UbiixLib.addSupportedMimeType](#) (String methodName, String mimeType, String subMimeType)
Set the SDK to receive the SIP messages that include special mime type.
- Int32 [Ubiix.UbiixLib.setAudioSamples](#) (Int32 ptime, Int32 maxPtime)
Set the audio capture sample.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.addSupportedMimeType (String *methodName*, String *mimeType*, String *subMimeType*)

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

Parameters:

<i>methodName</i>	Method name of the SIP message, such as INVITE, OPTION, INFO, MESSAGE, UPDATE, ACK etc. For more details please read 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, [Ubiix](#) VoIP SDK supports these media types (mime types) below for 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 send an INFO SIP message with its "Content-Type" header value "text/plain". SDK will reject this INFO message, as "text/plain" of INFO message is not included in the default support list. How should we enable the SDK to receive the SIP INFO message that includes "text/plain" mime type? The answer is `addSupportedMimeType`:

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

If we want to receive the NOTIFY message with "application/media_control+xml", please:
`addSupportedMimeType("NOTIFY", "application", "media_control+xml");`

For more details about the mime type, please visit this website:

<http://www.iana.org/assignments/media-types/>

Int32 Ubiix.UbiixLib.setAudioSamples (Int32 *ptime*, Int32 *maxPtime*)

Set the audio capture 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.

Access SIP message header functions

Functions

- Int32 [Ubiix.UbiixLib.setAudioDeviceId](#) (Int32 recordingDeviceId, Int32 playoutDeviceId)
Set the audio device that will be used for audio call.
- Int32 [Ubiix.UbiixLib.setVideoDeviceId](#) (Int32 deviceId)
Set the video device that will be used for video call.
- Int32 [Ubiix.UbiixLib.setVideoResolution](#) (Int32 width, Int32 height)
Set the video capturing resolution.
- Int32 [Ubiix.UbiixLib.setAudioBitrate](#) (Int32 sessionId, [AUDIOCODEC_TYPE](#) audioCodecType, Int32 bitrateKbps)
Set the audio bitrate.
- Int32 [Ubiix.UbiixLib.setVideoBitrate](#) (Int32 sessionId, Int32 bitrateKbps)
Set the video bitrate.
- Int32 [Ubiix.UbiixLib.setVideoFrameRate](#) (Int32 sessionId, Int32 frameRate)
Set the video frame rate.
- Int32 [Ubiix.UbiixLib.sendVideo](#) (Int32 sessionId, Boolean sendState)
Send the video to remote side.
- Int32 [Ubiix.UbiixLib.setVideoOrientation](#) (Int32 rotation)
Change the orientation of the video.
- void [Ubiix.UbiixLib.muteMicrophone](#) (Boolean mute)
Mute the device microphone. It's unavailable for Android and iOS.
- void [Ubiix.UbiixLib.muteSpeaker](#) (Boolean mute)
Mute the device speaker. It's unavailable for Android and iOS.

- void [Ubiix.UbiixLib.getDynamicVolumeLevel](#) (out Int32 speakerVolume, out Int32 microphoneVolume)
Obtain the dynamic microphone volume level from current call.
- void [Ubiix.UbiixLib.setChannelOutputVolumeScaling](#) (Int32 sessionId, Int32 scaling)
- void [Ubiix.UbiixLib.setLocalVideoWindow](#) (IntPtr localVideoWindow)
Set the window that is used to display the local video image.
- Int32 [Ubiix.UbiixLib.setRemoteVideoWindow](#) (Int32 sessionId, IntPtr remoteVideoWindow)
Set the window for a session that is used to display the received remote video image.
- Int32 [Ubiix.UbiixLib.displayLocalVideo](#) (Boolean state)
Start/stop displaying the local video image.
- Int32 [Ubiix.UbiixLib.setVideoNackStatus](#) (Boolean state)
Enable/disable the NACK feature (rfc6642) that helps to improve the video quality.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.setAudioDeviceId (Int32 *recordingDeviceId*, Int32 *playoutDeviceId*)

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

Parameters:

<i>recordingDeviceId</i>	Device ID (index) for audio recording. (Microphone).
<i>playoutDeviceId</i>	Device ID (index) for audio playback (Speaker).

Returns:

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

Int32 Ubiix.UbiixLib.setVideoDeviceId (Int32 *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.

Int32 Ubiix.UbiixLib.setVideoResolution (Int32 *width*, Int32 *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.

Int32 Ubiix.UbiixLib.setAudioBitrate (Int32 *sessionId*, [AUDIOCODEC_TYPE](#) *audioCodecType*, Int32 *bitrateKbps*)

Set the audio bitrate.

Parameters:

<i>bitrateKbps</i>	The audio bitrate in KBPS.
--------------------	----------------------------

Returns:

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

Int32 Ubiix.UbiixLib.setVideoBitrate (Int32 *sessionId*, Int32 *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.

Int32 Ubiix.UbiixLib.setVideoFrameRate (Int32 *sessionId*, Int32 *frameRate*)

Set the video frame rate.

Parameters:

<i>frameRate</i>	The frame rate value with minimum value 5, and maximum value 30. A greater value will enable you better video quality but 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.

Int32 Ubiix.UbiixLib.sendVideo (Int32 *sessionId*, Boolean *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.

Int32 Ubiix.UbiixLib.setVideoOrientation (Int32 *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 Ubiix.UbiixLib.muteMicrophone (Boolean *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 will be muted. You may also set it to false to un-mute it.
-------------	--

void Ubiix.UbiixLib.muteSpeaker (Boolean *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. You may also set it to false to un-mute it.
-------------	--

void Ubiix.UbiixLib.getDynamicVolumeLevel (out Int32 *speakerVolume*, out Int32 *microphoneVolume*)

Obtain the dynamic microphone volume level from current call.

Parameters:

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

Remarks:

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

void Ubiix.UbiixLib.setChannelOutputVolumeScaling (Int32 *sessionId*, Int32 *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.

void Ubiix.UbiixLib.setLocalVideoWindow (IntPtr *localVideoWindow*)

Set the window that is used to display the local video image.

Parameters:

<i>localVideoWindow</i>	The window on which the local video image from camera will be displayed.
-------------------------	--

Int32 Ubiix.UbiixLib.setRemoteVideoWindow (Int32 *sessionId*, IntPtr *remoteVideoWindow*)

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

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>remoteVideoWindow</i>	The window to display 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.

Int32 Ubiix.UbiixLib.displayLocalVideo (Boolean *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.

Int32 Ubiix.UbiixLib.setVideoNackStatus (Boolean *state*)

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

Parameters:

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

Returns:

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

Call functions

Functions

- Int32 [Ubiix.UbiixLib.call](#) (String callee, Boolean sendSdp, Boolean videoCall)
Make a call.
- Int32 [Ubiix.UbiixLib.rejectCall](#) (Int32 sessionId, int code)
rejectCall Reject the incoming call.
- Int32 [Ubiix.UbiixLib.hangUp](#) (Int32 sessionId)
hangUp Hang up the call.
- Int32 [Ubiix.UbiixLib.answerCall](#) (Int32 sessionId, Boolean videoCall)
answerCall Answer the incoming call.
- Int32 [Ubiix.UbiixLib.updateCall](#) (Int32 sessionId, bool enableAudio, bool enableVideo)
Use the re-INVITE to update the established call.

- Int32 [Ubiix.UbiixLib.hold](#) (Int32 sessionId)
To place a call on hold.
- Int32 [Ubiix.UbiixLib.unHold](#) (Int32 sessionId)
Take off hold.
- Int32 [Ubiix.UbiixLib.muteSession](#) (Int32 sessionId, Boolean muteIncomingAudio, Boolean muteOutgoingAudio, Boolean muteIncomingVideo, Boolean muteOutgoingVideo)
Mute the specified session audio or video.
- Int32 [Ubiix.UbiixLib.forwardCall](#) (Int32 sessionId, String forwardTo)
Forward call to another one when receiving the incoming call.
- Int32 [Ubiix.UbiixLib.pickupBLFCall](#) (String replaceDialogId, Boolean videoCall)
This function will be used for picking up a call based on the BLF (Busy Lamp Field) status.
- Int32 [Ubiix.UbiixLib.sendDtmf](#) (Int32 sessionId, [DTMF_METHOD](#) dtmfMethod, int code, int dtmfDuration, bool playDtmfTone)
Send DTMF tone.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.call (String *callee*, Boolean *sendSdp*, Boolean *videoCall*)

Make a call.

Parameters:

<i>callee</i>	The callee. It can be either name or full SIP URI. For example: user001, sip: user001@sip.iptel.org or sip: user002@sip.yourdomain.com :5068
<i>sendSdp</i>	If it's set to false, the outgoing call doesn't include the SDP in INVITE message.
<i>videoCall</i>	If it's set to true with at least one video codecs 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 that 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. You need to detect the call final state in onInviteTrying, onInviteRingin, onInviteFailure callback events.

Int32 Ubiix.UbiixLib.rejectCall (Int32 *sessionId*, int *code*)

rejectCall Reject the incoming call.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.hangUp (Int32 *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.

Int32 Ubiix.UbiixLib.answerCall (Int32 *sessionId*, Boolean *videoCall*)

answerCall Answer the incoming call.

Parameters:

<i>sessionId</i>	The session ID of call.
<i>videoCall</i>	If the incoming call is a video call and the video codec is matched, set it to true to answer the video call. If it's set to false, the answered 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.

Int32 Ubiix.UbiixLib.updateCall (Int32 *sessionId*, bool *enableAudio*, 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 value a specific error code.

Remarks:

Example usage:

Example 1: A called B with the audio only, B answered A, there has an audio conversation between A, B. Now A wants to see B visually, A could use these functions to do it.

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

Example 2: Remove video stream from current conversation.

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

Int32 Ubiix.UbiixLib.hold (Int32 *sessionId*)

To place a call on hold.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.unHold (Int32 *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.

Int32 Ubiix.UbiixLib.muteSession (Int32 *sessionId*, Boolean *muteIncomingAudio*, Boolean *muteOutgoingAudio*, Boolean *muteIncomingVideo*, Boolean *muteOutgoingVideo*)

Mute the specified session audio or video.

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>muteIncomingAudio</i>	Set it to true to mute incoming audio stream, and remote side audio cannot be heard.
<i>muteOutgoingAudio</i>	Set it to true to mute outgoing audio stream, and the remote side can't hear the audio.
<i>muteIncomingVideo</i>	Set it to true to mute incoming video stream, and the remote side video will be invisible.
<i>muteOutgoingVideo</i>	Set it to true to mute outgoing video stream, and the remote side can't see the video.

Returns:

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

Int32 Ubiix.UbiixLib.forwardCall (Int32 *sessionId*, String *forwardTo*)

Forward call to another one when receiving the incoming call.

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>forwardTo</i>	Target of the 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 value a specific error code.

Int32 Ubiix.UbiixLib.pickupBLFCall (String *replaceDialogId*, Boolean *videoCall*)

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

Parameters:

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

<i>videoCall</i>	Target of the forwarding. It can be "sip:number@sipserver.com" or "number" only.
------------------	--

Returns:

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

Remarks:

The scenario is:

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

Int32 Ubiix.UbiixLib.sendDtmf (Int32 *sessionId*, [DTMF_METHOD](#) *dtmfMethod*, int *code*, int *dtmfDuration*, bool *playDtmfTone*)

Send DTMF tone.

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>dtmfMethod</i>	DTMF tone could be sent with two methods: DTMF_RFC2833 and DTMF_INFO, of which 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>	If it is set 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

- Int32 [Ubiix.UbiixLib.refer](#) (Int32 sessionId, String referTo)
Refer the current call to another one.
- Int32 [Ubiix.UbiixLib.attendedRefer](#) (Int32 sessionId, Int32 replaceSessionId, String referTo)
Make an attended refer.
- Int32 [Ubiix.UbiixLib.attendedRefer2](#) (IntPtr libSDK, Int32 sessionId, Int32 replaceSessionId, String replaceMethod, String target, String referTo)
Make an attended refer with specified request line and specified method embedded into the "Refer-To" header.
- Int32 [Ubiix.UbiixLib.outOfDialogRefer](#) (Int32 replaceSessionId, String replaceMethod, String target, String referTo)
Send an out of dialog REFER to replace the specified call.
- Int32 [Ubiix.UbiixLib.acceptRefer](#) (Int32 referId, String referSignalingMessage)
Accept the REFER request, and a new call will be made if called this function. The function is usually called after onReceivedRefer callback event.
- Int32 [Ubiix.UbiixLib.rejectRefer](#) (Int32 referId)
Reject the REFER request.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.refer (Int32 sessionId, String 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, which can be either "sip:number@sipserver.com" or "number".

Returns:

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

Remarks:

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

You can download the demo AVI at

"<http://www.Ubiix.com/downloads/video/blindtransfer.rar>".

Use the Windows Media Player to play the AVI file after extracted, and it will demonstrate the transfer.

Int32 Ubiix.UbiixLib.attendedRefer (Int32 *sessionId*, Int32 *replaceSessionId*, String *referTo*)

Make an attended refer.

Parameters:

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

Returns:

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

Remarks:

Please read the sample project source code for more details, or download the demo AVI at:

<http://www.Ubiix.com/downloads/video/blindtransfer.rar>

Please use the Windows Media Player to play the AVI file after extracted, and it will demonstrate the transfer.

Int32 Ubiix.UbiixLib.attendedRefer2 (IntPtr *libSDK*, Int32 *sessionId*, Int32 *replaceSessionId*, String *replaceMethod*, String *target*, String *referTo*)

Make an attended refer with specified request line and specified method embedded into the "Refer-To" header.

Parameters:

<i>sessionId</i>	Session ID of the call.
<i>replaceSessionId</i>	Session ID of the replaced call.
<i>replaceMethod</i>	The SIP method name which will be embeded in the "Refer-To" header, usually INVITE or BYE.
<i>target</i>	The target to which the REFER message will be sent. It appears in the "Request Line" of REFER message.
<i>referTo</i>	Target of the refer that appears in the "Refer-To" header. It can be either "sip:number@sipserver.com" or "number".

Returns:

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

Remarks:

Please refer to the sample project source code for more details. Or you can watch the video on YouTube at https://www.youtube.com/watch?v=_2w9EGgr3FY. It will demonstrate the transmission.

Int32 Ubiix.UbiixLib.outOfDialogRefer (Int32 *replaceSessionId*, String *replaceMethod*, String *target*, String *referTo*)

Send an out of dialog REFER to replace the specified call.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.acceptRefer (Int32 *referId*, String *referSignalingMessage*)

Accept the REFER request, and 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>referSignalingMessage</i>	The SIP message of REFER request that comes from onReceivedRefer callback event.

Returns:

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

Int32 Ubiix.UbiixLib.rejectRefer (Int32 *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

- Int32 [Ubiix.UbiixLib.enableSendPcmStreamToRemote](#) (Int32 sessionId, Boolean state, Int32 streamSamplesPerSec)
Enable the SDK to send PCM stream data to remote side from another source instead of microphone.
- Int32 [Ubiix.UbiixLib.sendPcmStreamToRemote](#) (Int32 sessionId, byte[] data, Int32 dataLength)
Send the audio stream in PCM format from another source instead of audio device capturing (microphone).
- Int32 [Ubiix.UbiixLib.enableSendVideoStreamToRemote](#) (Int32 sessionId, Boolean state)
Enable the SDK send video stream data to remote side from another source instead of camera.
- Int32 [Ubiix.UbiixLib.sendVideoStreamToRemote](#) (Int32 sessionId, byte[] data, Int32 dataLength, Int32 width, Int32 height)
Send the video stream to remote side.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.enableSendPcmStreamToRemote (Int32 *sessionId*, Boolean *state*, Int32 *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:

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

Int32 Ubiix.UbiixLib.sendPcmStreamToRemote (Int32 *sessionId*, byte [] *data*, Int32 *dataLength*)

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

Parameters:

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

Returns:

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

Remarks:

Usually it should be used as below:

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

Int32 Ubiix.UbiixLib.enableSendVideoStreamToRemote (Int32 *sessionId*, Boolean *state*)

Enable the SDK 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.

Int32 Ubiix.UbiixLib.sendVideoStreamToRemote (Int32 *sessionId*, byte [] *data*, Int32 *dataLength*, Int32 *width*, Int32 *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>dataLength</i>	The size of data.
<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 it should be used as below:

```
enableSendVideoStreamToRemote(sessionId, true);  
sendVideoStreamToRemote(sessionId, data, dataSize, 352, 288);
```

RTP packets, Audio stream and video stream callback functions

Functions

- Int32 [Ubiix.UbiixLib.enableAudioStreamCallback](#) (Int32 callbackObject, Int32 sessionId, Boolean enable, [AUDIOSTREAM_CALLBACK_MODE](#) callbackMode)
Enable/disable the audio stream callback.
- Int32 [Ubiix.UbiixLib.enableVideoStreamCallback](#) (Int32 callbackObject, Int32 sessionId, [VIDEOSTREAM_CALLBACK_MODE](#) callbackMode)
Enable/disable the video stream callback.
- Int32 [Ubiix.UbiixLib.enableVideoDecoderCallback](#) (Int32 callbackObject, Boolean enable)
Enable/disable the video Decoder callback.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.enableAudioStreamCallback (Int32 *callbackObject*, Int32 *sessionId*, Boolean *enable*, [AUDIOSTREAM_CALLBACK_MODE](#) *callbackMode*)

Enable/disable the audio stream callback.

Parameters:

<i>callbackObject</i>	The callback object that you passed in can be accessed once callback function triggered.
<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

Mode	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 sessionId.
AUDIOSTREAM_REMOTE_MIX	Callback the received audio stream that mixed all included channels.
AUDIOSTREAM_REMOTE_PER_CHANNEL	Callback the received audio stream for one channel based on the given sessionId.

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.

Int32 Ubiix.UbiixLib.enableVideoStreamCallback (Int32 *callbackObject*, Int32 *sessionId*, [VIDEOSTREAM_CALLBACK_MODE](#) *callbackMode*)

Enable/disable the video stream callback.

Parameters:

<i>callbackObject</i>	The callback object that you passed in can be accessed once callback function triggered.
<i>sessionId</i>	The session ID of call.
<i>callbackMode</i>	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 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.

Int32 Ubiix.UbiixLib.enableVideoDecoderCallback (Int32 *callbackObject*, Boolean *enable*)

Enable/disable the video Decoder callback.

Parameters:

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

Returns:

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

Remarks:

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

Record functions

Functions

- Int32 [Ubiix.UbiixLib.startRecord](#) (Int32 sessionId, String recordFilePath, String recordFileName, Boolean appendTimestamp, [AUDIO_RECORDING_FILEFORMAT](#) audioFileFormat, [RECORD_MODE](#) audioRecordMode, [VIDEOCODEC_TYPE](#) videoFileCodecType, [RECORD_MODE](#) videoRecordMode)
Start recording the call.
- Int32 [Ubiix.UbiixLib.stopRecord](#) (Int32 sessionId)
Stop record.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.startRecord (Int32 sessionId, String recordFilePath, String recordFileName, Boolean appendTimestamp, [AUDIO_RECORDING_FILEFORMAT](#) audioFileFormat, [RECORD_MODE](#) audioRecordMode, [VIDEOCODEC_TYPE](#) videoFileCodecType, [RECORD_MODE](#) videoRecordMode)

Start recording the call.

Parameters:

<i>sessionId</i>	The session ID of call conversation.
<i>recordFilePath</i>	The file path to which the record file will be saved. It must be existent.
<i>recordFileName</i>	The file name of record file. For example: audiorecord.wav or videorecord.avi.
<i>appendTimestamp</i>	Set to true to append the timestamp to the recording file name.
<i>audioFileFormat</i>	The audio record file format.
<i>audioRecordMode</i>	The audio record mode.
<i>videoFileCodecType</i>	The codec which is used for compressing the video data to save into video record file.
<i>videoRecordMode</i>	Allow to set video record mode, with record received and/or sent supported.

Returns:

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

Int32 Ubiix.UbiixLib.stopRecord (Int32 sessionId)

Stop record.

Parameters:

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

Returns:

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

Play audio and video file to remote functions

Functions

- Int32 [Ubiix.UbiixLib.playVideoFileToRemote](#) (Int32 sessionId, String fileName, Boolean loop, Boolean playAudio)
Play an AVI file to remote party.
- Int32 [Ubiix.UbiixLib.stopPlayVideoFileToRemote](#) (Int32 sessionId)
Stop playing video file to remote side.
- Int32 [Ubiix.UbiixLib.playAudioFileToRemote](#) (Int32 sessionId, String fileName, Int32 fileSamplesPerSec, Boolean loop)
Play a wave file to remote party.
- Int32 [Ubiix.UbiixLib.stopPlayAudioFileToRemote](#) (Int32 sessionId)
Stop playing wave file to remote side.
- Int32 [Ubiix.UbiixLib.playAudioFileToRemoteAsBackground](#) (Int32 sessionId, String fileName, Int32 fileSamplesPerSec)
Play a wave file to remote party as conversation background sound.
- Int32 [Ubiix.UbiixLib.stopPlayAudioFileToRemoteAsBackground](#) (Int32 sessionId)
Stop playing wave file to remote party as conversation background sound.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.playVideoFileToRemote (Int32 sessionId, String fileName, Boolean loop, Boolean playAudio)

Play an AVI file to remote party.

Parameters:

<i>sessionId</i>	Session ID of the call.
<i>fileName</i>	The full file path, such as "c:\\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's set to true, audio and video will be played together; if false, only video will be played.

Returns:

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

Int32 Ubiix.UbiixLib.stopPlayVideoFileToRemote (Int32 sessionId)

Stop playing video file to remote side.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.playAudioFileToRemote (Int32 *sessionId*, String *fileName*, Int32 *fileSamplesPerSec*, Boolean *loop*)

Play a wave file to remote party.

Parameters:

<i>sessionId</i>	Session ID of the call.
<i>fileName</i>	The full filepath, such as "c:\\test.wav".
<i>fileSamplesPerSec</i>	The wave file sample 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.

Int32 Ubiix.UbiixLib.stopPlayAudioFileToRemote (Int32 *sessionId*)

Stop playing wave file to remote side.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.playAudioFileToRemoteAsBackground (Int32 *sessionId*, String *fileName*, Int32 *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 "c:\\test.wav".
<i>fileSamplesPerSec</i>	The wave file sample in seconds. It should be 8000, 16000 or 32000.

Returns:

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

Int32 Ubiix.UbiixLib.stopPlayAudioFileToRemoteAsBackground (Int32 *sessionId*)

Stop playing 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

- Int32 [Ubiix.UbiixLib.createAudioConference](#) ()
Create an audio conference. It will be failed if the existent conference is not ended yet.
- Int32 [Ubiix.UbiixLib.createVideoConference](#) (IntPtr conferenceVideoWindow, Int32 width, Int32 height, Boolean displayLocalVideoInConference)
Create a video conference. It will be failed if the existent conference is not ended yet.
- void [Ubiix.UbiixLib.destroyConference](#) ()
End the existent conference.
- Int32 [Ubiix.UbiixLib.setConferenceVideoWindow](#) (IntPtr videoWindow)
Set the window for a conference that is used to display the received remote video image.
- Int32 [Ubiix.UbiixLib.joinToConference](#) (Int32 sessionId)
Join a session into existent conference. If the call is in hold, it will be un-hold automatically.
- Int32 [Ubiix.UbiixLib.removeFromConference](#) (Int32 sessionId)
Remove a session from an existent conference.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.createAudioConference ()

Create an audio conference. It will be failed if the existent conference is not ended yet.

Returns:

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

Int32 Ubiix.UbiixLib.createVideoConference (IntPtr conferenceVideoWindow, Int32 width, Int32 height, Boolean displayLocalVideoInConference)

Create a video conference. It will be failed if the existent conference is not ended yet.

Parameters:

<i>conferenceVideoWindow</i>	The UIView used to display the conference video.
<i>videoResolution</i>	The conference video resolution.
<i>displayLocalVideoInConference</i>	Display the local video on video window or not.

Returns:

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

Int32 Ubiix.UbiixLib.setConferenceVideoWindow (IntPtr videoWindow)

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

Parameters:

<code>videoWindow</code>	The UIView 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.

Int32 Ubiix.UbiixLib.joinToConference (Int32 *sessionId*)

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

Parameters:

<code>sessionId</code>	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.

Int32 Ubiix.UbiixLib.removeFromConference (Int32 *sessionId*)

Remove a session from an existent conference.

Parameters:

<code>sessionId</code>	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

- Int32 [Ubiix.UbiixLib.setAudioRtcpBandwidth](#) (Int32 *sessionId*, Int32 *BitsRR*, Int32 *BitsRS*, Int32 *KBitsAS*)
Set the audio RTCP bandwidth parameters to the RFC3556.
- Int32 [Ubiix.UbiixLib.setVideoRtcpBandwidth](#) (Int32 *sessionId*, Int32 *BitsRR*, Int32 *BitsRS*, Int32 *KBitsAS*)
Set the video RTCP bandwidth parameters as the RFC3556.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.setAudioRtcpBandwidth (Int32 *sessionId*, Int32 *BitsRR*, Int32 *BitsRS*, Int32 *KBitsAS*)

Set the audio RTCP bandwidth parameters to 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.

Int32 Ubiix.UbiixLib.setVideoRtcpBandwidth (Int32 *sessionId*, Int32 *BitsRR*, Int32 *BitsRS*, Int32 *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.

RTP statistics functions

Functions

- Int32 [Ubiix.UbiixLib.getNetworkStatistics](#) (Int32 sessionId, out Int32 currentBufferSize, out Int32 preferredBufferSize, out Int32 currentPacketLossRate, out Int32 currentDiscardRate, out Int32 currentExpandRate, out Int32 currentPreemptiveRate, out Int32 currentAccelerateRate)
Get the "in-call" statistics. The statistics are reset after the query.
- Int32 [Ubiix.UbiixLib.getAudioRtpStatistics](#) (Int32 sessionId, out Int32 averageJitterMs, out Int32 maxJitterMs, out Int32 discardedPackets)
Obtain the RTP statistics of audio channel.
- Int32 [Ubiix.UbiixLib.getAudioRtcpStatistics](#) (Int32 sessionId, out Int32 bytesSent, out Int32 packetsSent, out Int32 bytesReceived, out Int32 packetsReceived, out Int32 sendFractionLost, out Int32 sendCumulativeLost, out Int32 recvFractionLost, out Int32 recvCumulativeLost)
Obtain the RTCP statistics of audio channel.
- Int32 [Ubiix.UbiixLib.getVideoRtpStatistics](#) (Int32 sessionId, out Int32 bytesSent, out Int32 packetsSent, out Int32 bytesReceived, out Int32 packetsReceived)
Obtain the RTP statistics of video.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.getNetworkStatistics (Int32 *sessionId*, out Int32 *currentBufferSize*, out Int32 *preferredBufferSize*, out Int32 *currentPacketLossRate*, out Int32 *currentDiscardRate*, out Int32 *currentExpandRate*, out Int32 *currentPreemptiveRate*, out Int32 *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 (optimal) 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.

Int32 Ubiix.UbiixLib.getAudioRtpStatistics (Int32 *sessionId*, out Int32 *averageJitterMs*, out Int32 *maxJitterMs*, out Int32 *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 value a specific error code.

Int32 Ubiix.UbiixLib.getAudioRtcpStatistics (Int32 *sessionId*, out Int32 *bytesSent*, out Int32 *packetsSent*, out Int32 *bytesReceived*, out Int32 *packetsReceived*, out Int32 *sendFractionLost*, out Int32 *sendCumulativeLost*, out Int32 *recvFractionLost*, out Int32 *recvCumulativeLost*)

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>recvFractionLost</i>	Fraction of received lost in percentage.
<i>recvCumulativeLost</i>	The number of received cumulative lost packets.

Returns:

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

Int32 Ubiix.UbiixLib.getVideoRtpStatistics (Int32 *sessionId*, out Int32 *bytesSent*, out Int32 *packetsSent*, out Int32 *bytesReceived*, out Int32 *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 [Ubiix.UbiixLib.enableVAD](#) (Boolean state)
Enable/disable Voice Activity Detection (VAD).
 - void [Ubiix.UbiixLib.enableAEC](#) ([EC_MODES](#) ecMode)
Enable/disable AEC (Acoustic Echo Cancellation).
 - void [Ubiix.UbiixLib.enableCNG](#) (Boolean state)
Enable/disable Comfort Noise Generator (CNG).
 - void [Ubiix.UbiixLib.enableAGC](#) ([AGC_MODES](#) agcMode)
Enable/disable Automatic Gain Control (AGC).
 - void [Ubiix.UbiixLib.enableANS](#) ([NS_MODES](#) nsMode)
Enable/disable Audio Noise Suppression (ANS).
 - Int32 [Ubiix.UbiixLib.setAudioQos](#) (Boolean state, Int32 DSCPValue, Int32 priority)
Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for audio channel.
 - Int32 [Ubiix.UbiixLib.setVideoQos](#) (Boolean state, Int32 DSCPValue)
Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for video channel.
 - Int32 [Ubiix.UbiixLib.setVideoMTU](#) (Int32 mtu)
Set the MTU size for video RTP packet.
-

Detailed Description

Function Documentation

void Ubiix.UbiixLib.enableVAD (Boolean *state*)

Enable/disable Voice Activity Detection (VAD).

Parameters:

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

void Ubiix.UbiixLib.enableAEC ([EC MODES](#) *ecMode*)

Enable/disable AEC (Acoustic Echo Cancellation).

Parameters:

<i>ecMode</i>	Allow to set the AEC mode to influence different scenarios.
---------------	---

Mode	Description
EC_NONE	Disable AEC.
EC_DEFAULT	Platform default AEC.
EC_CONFERENCE	Desktop platform (Windows, MAC) Conferencing default (aggressive AEC).

void Ubiix.UbiixLib.enableCNG (Boolean *state*)

Enable/disable Comfort Noise Generator (CNG).

Parameters:

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

void Ubiix.UbiixLib.enableAGC ([AGC MODES](#) *agcMode*)

Enable/disable Automatic Gain Control (AGC).

Parameters:

<i>agcMode</i>	Allow to set the AGC mode to influence different scenarios.
----------------	---

Mode	Description
AGC_DEFAULT	Disable AGC.
AGC_DEFAULT	Platform default.
AGC_ADAPTIVE_ANALOG	Desktop platform (Windows, MAC) adaptive mode for use when analog volume control exists.
AGC_ADAPTIVE_DIGITAL	Scaling takes place in the digital domain (e.g. for conference servers and embedded devices).
AGC_FIXED_DIGITAL	It can be used on embedded devices where the capture signal level is predictable.

void Ubiix.UbiixLib.enableANS ([NS MODES](#) *nsMode*)

Enable/disable Audio Noise Suppression (ANS).

Parameters:

<i>nsMode</i>	Allow to set the NS mode to influence different scenarios.
Mode	Description
NS_NONE	Disable NS.
NS_DEFAULT	Platform default.
NS_Conference	Conferencing default.
NS_LOW_SUPPRESSION	Lowest suppression.
NS_MODERATE_SUPPRESSION	Moderate suppression.
NS_HIGH_SUPPRESSION	High suppression
NS_VERY_HIGH_SUPPRESSION	Highest suppression.

Int32 Ubiix.UbiixLib.setAudioQos (Boolean *state*, Int32 *DSCPValue*, Int32 *priority*)

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

Parameters:

<i>state</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 an 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.

Int32 Ubiix.UbiixLib.setVideoQos (Boolean *state*, Int32 *DSCPValue*)

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

Parameters:

<i>state</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.

Int32 Ubiix.UbiixLib.setVideoMTU (Int32 *mtu*)

Set the MTU size for video RTP packet.

Parameters:

<i>mtu</i>	Set MTU value. Allow value ranges (512-65507). Other value will be
------------	--

	modified to the default 1450.
--	-------------------------------

Returns:

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

Send OPTIONS/INFO/MESSAGE functions

Functions

- Int32 [Ubiix.UbiixLib.sendOptions](#) (String to, String sdp)
Send OPTIONS message.
- Int32 [Ubiix.UbiixLib.sendInfo](#) (Int32 sessionId, String mimeType, String subMimeType, String infoContents)
Send a INFO message to remote side in a call.
- Int32 [Ubiix.UbiixLib.sendSubscription](#) (String to, String eventName)
Send a SUBSCRIBE message to subscribe an event.
- Int32 [Ubiix.UbiixLib.terminateSubscription](#) (Int32 subscribeId)
Terminate the given subscription.
- Int32 [Ubiix.UbiixLib.sendMessage](#) (Int32 sessionId, String mimeType, String subMimeType, byte[] message, Int32 messageLength)
Send a MESSAGE message to remote side in dialog.
- Int32 [Ubiix.UbiixLib.sendOutOfDialogMessage](#) (String to, String mimeType, String subMimeType, byte[] message, Int32 messageLength)
Send an out of dialog MESSAGE message to remote side.
- Int32 [Ubiix.UbiixLib.setDefaultSubscriptionTime](#) (Int32 secs)
Set the default expiration time to be used when creating a subscription.
- Int32 [Ubiix.UbiixLib.setDefaultPublicationTime](#) (Int32 secs)
Set the default expiration time to be used when creating a publication.
- Int32 [Ubiix.UbiixLib.setPresenceMode](#) (Int32 mode)
Indicate the SDK uses the P2P mode for presence or presence agent mode.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.sendOptions (String to, String sdp)

Send OPTIONS message.

Parameters:

to	The recipient of OPTIONS message.
sdp	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.

Int32 Ubiix.UbiixLib.sendInfo (Int32 *sessionId*, String *contentType*, String *subMimeType*, String *infoContents*)

Send a INFO message to remote side in a call.

Parameters:

<i>sessionId</i>	The session ID of call.
<i>contentType</i>	The mime type of INFO message.
<i>subMimeType</i>	The sub mime type of INFO message.
<i>infoContents</i>	The contents that is 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.

Int32 Ubiix.UbiixLib.sendSubscription (String *to*, String *eventName*)

Send a SUBSCRIBE message to subscribe an event.

Parameters:

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

Returns:

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

Remarks:

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

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

Int32 Ubiix.UbiixLib.terminateSubscription (Int32 *subscribeId*)

Terminate the given subscription.

Parameters:

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

Returns:

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

Remarks:

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

```
terminateSubscription (mwiSubId) ;
```

Int32 Ubiix.UbiixLib.sendMessage (Int32 *sessionId*, String *contentType*, String *subMimeType*, byte [] *message*, Int32 *messageLength*)

Send a MESSAGE message to remote side in dialog.

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>contentType</i>	The mime type of MESSAGE message.
<i>subContentType</i>	The sub mime type of MESSAGE message.
<i>message</i>	The contents which is 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 text in other languages, please use the UTF-8 to encode the message before sending.

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

Example 2: send a binary message.

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

Int32 Ubiix.UbiixLib.sendOutOfDialogMessage (String to, String contentType, String subContentType, byte [] message, Int32 messageLength)

Send an out of dialog MESSAGE message to remote side.

Parameters:

<i>to</i>	The message recipient, such as sip: receiver@Ubiix.com
<i>contentType</i>	The mime type of MESSAGE message.
<i>subContentType</i>	The sub mime type of MESSAGE message.
<i>message</i>	The contents which is 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 the UTF-8 to encode the message before sending.

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

Example 2: send a binary message.

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

Int32 Ubiix.UbiixLib.setDefaultSubscriptionTime (Int32 secs)

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

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.setDefaultPublicationTime (Int32 secs)

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

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.setPresenceMode (Int32 *mode*)

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

Parameters:

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

Returns:

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

Remarks:

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

Presence functions

Functions

- Int32 [Ubiix.UbiixLib.presenceSubscribe](#) (String *to*, String *subject*)
Send a SUBSCRIBE message for subscribing the contact's presence status.
- Int32 [Ubiix.UbiixLib.presenceTerminateSubscribe](#) (Int32 *subscribeId*)
Terminate the given presence subscription.
- Int32 [Ubiix.UbiixLib.presenceRejectSubscribe](#) (Int32 *subscribeId*)
Reject a presence SUBSCRIBE request which is received from contact.
- Int32 [Ubiix.UbiixLib.presenceAcceptSubscribe](#) (Int32 *subscribeId*)
Accept the presence SUBSCRIBE request which is received from contact.
- Int32 [Ubiix.UbiixLib.setPresenceStatus](#) (Int32 *subscribeId*, String *stateText*)
Set the presence status.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.presenceSubscribe (String *to*, String *subject*)

Send a SUBSCRIBE message for subscribing the contact's presence status.

Parameters:

<i>to</i>	The target contact. It must be like sip: contact001@sip.Ubiix.com .
<i>subject</i>	This subject text will be inserted into the SUBSCRIBE message. For example: "Hello, I'm Jason". The subject maybe in UTF-8 format. You should use UTF-8 to decode it.

Returns:

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

Int32 Ubiix.UbiixLib.presenceTerminateSubscribe (Int32 *subscribeId*)

Terminate the given presence subscription.

Parameters:

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

Returns:

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

Int32 Ubiix.UbiixLib.presenceRejectSubscribe (Int32 *subscribeId*)

Reject a presence SUBSCRIBE request which is received from contact.

Parameters:

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

Returns:

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

Remarks:

If the P2P presence mode is enabled, when someone subscribe your presence status, you will receive the subscribe request in the callback, and you can use this function to accept it.

Int32 Ubiix.UbiixLib.presenceAcceptSubscribe (Int32 *subscribeId*)

Accept the presence SUBSCRIBE request which is received from contact.

Parameters:

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

Returns:

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

Remarks:

If the P2P presence mode is enabled, when someone subscribes your presence status, you will receive the subscription request in the callback, and you can use this function to reject it.

Int32 Ubiix.UbiixLib.setPresenceStatus (Int32 *subscribeId*, String *stateText*)

Set the presence status.

Parameters:

<i>subscribeId</i>	Subscription ID.
<i>stateText</i>	The state text of presence online. For example: "I'm here". If you want to appear as offline to others, please pass the "Offline" to "statusText" parameter.

Returns:

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

Remarks:

With P2P presence mode, when receiving a SUBSCRIBE request from contact, the event onPresenceRecvSubscribe will be triggered. The event includes the subscription ID. This function will cause the SDK sending a NOTIFY message to update your presence status, and you must pass the correct subscribeId.

With presence agent mode, this function will cause the SDK to send a PUBLISH message to update your presence status, and you must pass 0 to the "subscribeId" parameter.

Device Manage functions.

Functions

- Int32 [Ubiix.UbiixLib.getNumOfRecordingDevices \(\)](#)
Gets the count of audio devices available for audio recording.
- Int32 [Ubiix.UbiixLib.getNumOfPlayOutDevices \(\)](#)
Gets the number of audio devices available for audio playOut.
- Int32 [Ubiix.UbiixLib.getRecordingDeviceName](#) (Int32 deviceIndex, StringBuilder nameUTF8, Int32 nameUTF8Length)
Gets the name of a specific recording device given by an index.
- Int32 [Ubiix.UbiixLib.getPlayOutDeviceName](#) (Int32 deviceIndex, StringBuilder nameUTF8, Int32 nameUTF8Length)
Get the name of a specific playOut device given by an index.
- Int32 [Ubiix.UbiixLib.setSpeakerVolume](#) (Int32 volume)
Set the speaker volume level.
- Int32 [Ubiix.UbiixLib.getSpeakerVolume](#) ()
Gets the speaker volume level.
- Int32 [Ubiix.UbiixLib.setSystemOutputMute](#) (Boolean enable)
Mute the speaker device completely in the OS.
- Boolean [Ubiix.UbiixLib.getSystemOutputMute](#) ()
Retrieves the output device mute state in the operating system.
- Int32 [Ubiix.UbiixLib.setMicVolume](#) (Int32 volume)
Sets the microphone volume level.
- Int32 [Ubiix.UbiixLib.getMicVolume](#) ()
Retrieves the current microphone volume.
- Int32 [Ubiix.UbiixLib.setSystemInputMute](#) (Boolean enable)
Mute the microphone input device completely in the OS.
- Boolean [Ubiix.UbiixLib.getSystemInputMute](#) ()
Get the mute state of the input device in the operating system.
- void [Ubiix.UbiixLib.audioPlayLoopbackTest](#) (Boolean enable)
Use it for the audio device loop back test.
- Int32 [Ubiix.UbiixLib.getNumOfVideoCaptureDevices](#) ()
Get the number of available capturing devices.

- Int32 [Ubiix.UbiixLib.getVideoCaptureDeviceName](#) (Int32 deviceIndex, StringBuilder uniqueIdUTF8, Int32 uniqueIdUTF8Length, StringBuilder deviceNameUTF8, Int32 deviceNameUTF8Length)
Get the name of a specific video capture device given by an index.
- Int32 [Ubiix.UbiixLib.showVideoCaptureSettingsDialogBox](#) (String uniqueIdUTF8, Int32 uniqueIdUTF8Length, String dialogTitle, IntPtr parentWindow, Int32 x, Int32 y)
Display the capture device property dialog box for the specified capture device.

Detailed Description

Function Documentation

Int32 Ubiix.UbiixLib.getNumOfRecordingDevices ()

Gets the count of audio devices available for audio recording.

Returns:

It will return the count of recording devices. If the function fails, it will return a specific error code less than 0.

Int32 Ubiix.UbiixLib.getNumOfPlayoutDevices ()

Gets the number of audio devices available for audio playout.

Returns:

It will return the count of playout devices. If the function fails, it will return a specific error code less than 0.

Int32 Ubiix.UbiixLib.getRecordingDeviceName (Int32 *deviceIndex*, StringBuilder *nameUTF8*, Int32 *nameUTF8Length*)

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

Parameters:

<i>deviceIndex</i>	Device index (0, 1, 2, ..., N-1), where N is given by <code>getNumOfRecordingDevices ()</code> . Also -1 is a valid value and will return the name of the default recording device.
<i>nameUTF8</i>	A character buffer to which the device name will be copied as a null-terminated string in UTF-8 format.
<i>nameUTF8Length</i>	The size of nameUTF8 buffer. It cannot be less than 128.

Returns:

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

Int32 Ubiix.UbiixLib.getPlayoutDeviceName (Int32 *deviceIndex*, StringBuilder *nameUTF8*, Int32 *nameUTF8Length*)

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

Parameters:

<i>deviceIndex</i>	
<i>deviceIndex</i>	Device index (0, 1, 2, ..., N-1), where N is given by <code>getNumOfRecordingDevices ()</code> . Also -1 is a valid value and will return the name of the default recording device.
<i>nameUTF8</i>	A character buffer to which the device name will be copied as a null-terminated string in UTF-8 format.
<i>nameUTF8Length</i>	The size of <i>nameUTF8</i> buffer. It cannot be less than 128.

Returns:

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

Int32 Ubiix.UbiixLib.setSpeakerVolume (Int32 *volume*)

Set the speaker volume level.

Parameters:

<i>volume</i>	Volume level of speaker. Valid value ranges 0 - 255.
---------------	--

Returns:

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

Int32 Ubiix.UbiixLib.getSpeakerVolume ()

Gets the speaker volume level.

Returns:

If the function succeeds, it will return the speaker volume with valid range 0 - 255. If the function fails, it will return a specific error code.

Int32 Ubiix.UbiixLib.setSystemOutputMute (Boolean *enable*)

Mute the speaker device completely in the OS.

Parameters:

<i>enable</i>	If it's set to true, the device output is muted. If false, the output is unmuted.
---------------	---

Returns:

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

Boolean Ubiix.UbiixLib.getSystemOutputMute ()

Retrieves the output device mute state in the operating system.

Returns:

If it returns value true, the output device is muted. If false, the output device is not muted.

Int32 Ubiix.UbiixLib.setMicVolume (Int32 *volume*)

Sets the microphone volume level.

Parameters:

<i>volume</i>	The microphone volume. Valid value ranges 0 - 255.
---------------	--

Returns:

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

Int32 Ubiix.UbiixLib.getMicVolume ()

Retrieves the current microphone volume.

Returns:

If the function succeeds, it will return the microphone volume. If the function fails, it will return a specific error code.

Int32 Ubiix.UbiixLib.setSystemInputMute (Boolean *enable*)

Mute the microphone input device completely in the OS.

Parameters:

<i>enable</i>	If it is set to true, the input device is muted, or set to false to unmute.
---------------	---

Returns:

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

Boolean Ubiix.UbiixLib.getSystemInputMute ()

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

Returns:

If it returns value true, the input device is muted. If false, the input device is not muted.

void Ubiix.UbiixLib.audioPlayLoopbackTest (Boolean *enable*)

Use it for the audio device loop back test.

Parameters:

<i>enable</i>	Set to true to start audio look back test; or false to stop.
---------------	--

Int32 Ubiix.UbiixLib.getNumOfVideoCaptureDevices ()

Get the number of available capturing devices.

Returns:

It will return the count of video capturing devices. If it fails, it will return a specific error code less than 0.

Int32 Ubiix.UbiixLib.getVideoCaptureDeviceName (Int32 *deviceIndex*, StringBuilder *uniqueIdUTF8*, Int32 *uniqueIdUTF8Length*, StringBuilder *deviceNameUTF8*, Int32 *deviceNameUTF8Length*)

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

Parameters:

<i>deviceIndex</i>	Device index (0, 1, 2, ..., N-1), where N is given by <code>getNumOfVideoCaptureDevices()</code> . Also -1 is a valid value and will return the name of the default capturing device.
<i>uniqueIdUTF8</i>	Unique identifier of the capturing device.
<i>uniqueIdUTF8Length</i>	Size in bytes of <i>uniqueIdUTF8</i> .
<i>deviceNameUTF8</i>	A character buffer to which the device name will be copied as a null-terminated string in UTF-8 format.
<i>deviceNameUTF8Length</i>	The size of <i>nameUTF8</i> buffer. It cannot be less than 128.

Returns:

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

Int32 Ubiix.UbiixLib.showVideoCaptureSettingsDialogBox (String *uniqueIdUTF8*, Int32 *uniqueIdUTF8Length*, String *dialogTitle*, IntPtr *parentWindow*, Int32 *x*, Int32 *y*)

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

Parameters:

<i>uniqueIdUTF8</i>	Unique identifier of the capture device.
<i>uniqueIdUTF8Length</i>	Size in bytes of <i>uniqueIdUTF8</i> .
<i>dialogTitle</i>	The title of the video settings dialog.
<i>parentWindow</i>	Parent window used for the dialog box. It should originally be a HWND.
<i>x</i>	Horizontal position for the dialog relative to the parent window, in pixels.
<i>y</i>	Vertical position for the dialog relative to the parent window, in pixels.

Returns:

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

SDK Callback events

Modules

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

Detailed Description

SDK Callback events

Register events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onRegisterSuccess](#) (Int32 callbackIndex, Int32 callbackObject, String statusText, Int32 statusCode, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onRegisterFailure](#) (Int32 callbackIndex, Int32 callbackObject, String statusText, Int32 statusCode, StringBuilder sipMessage)

Detailed Description

Register events

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onRegisterSuccess (Int32 *callbackIndex*, Int32 *callbackObject*, String *statusText*, Int32 *statusCode*, StringBuilder *sipMessage*)

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

Parameters:

<i>callbackIndex</i>	This is a callback index passed in when creating the SDK library.
<i>callbackObject</i>	This is a callback object passed in when creating the SDK library.
<i>statusText</i>	The status text.
<i>statusCode</i>	The status code.
<i>sipMessage</i>	The SIP message received.

Int32 Ubiix.SIPCallbackEvents.onRegisterFailure (Int32 *callbackIndex*, Int32 *callbackObject*, String *statusText*, Int32 *statusCode*, StringBuilder *sipMessage*)

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

Parameters:

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

Call events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onInviteIncoming](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onInviteTrying](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [Ubiix.SIPCallbackEvents.onInviteSessionProgress](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsEarlyMedia, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onInviteRinging](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String statusText, Int32 statusCode, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onInviteAnswered](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)

- Int32 [Ubiix.SIPCallbackEvents.onInviteFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String reason, Int32 code, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onInviteUpdated](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [Ubiix.SIPCallbackEvents.onInviteConnected](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [Ubiix.SIPCallbackEvents.onInviteBeginingForward](#) (Int32 callbackIndex, Int32 callbackObject, String forwardTo)
- Int32 [Ubiix.SIPCallbackEvents.onInviteClosed](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [Ubiix.SIPCallbackEvents.onDialogStateUpdated](#) (Int32 callbackIndex, Int32 callbackObject, String BLFMonitoredUri, String BLFDialogState, String BLFDialogId, String BLFDialogDirection)
- Int32 [Ubiix.SIPCallbackEvents.onRemoteHold](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [Ubiix.SIPCallbackEvents.onRemoteUnHold](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onInviteIncoming (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *callerDisplayName*, String *caller*, String *calleeDisplayName*, String *callee*, String *audioCodecNames*, String *videoCodecNames*, Boolean *existsAudio*, Boolean *existsVideo*, StringBuilder *sipMessage*)

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>audioCodecNames</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecNames</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	If it's true, it indicates that this call includes the audio.
<i>existsVideo</i>	If it's true, it indicates that this call includes the video.
<i>sipMessage</i>	The SIP message received.

Int32 Ubiix.SIPCallbackEvents.onInviteTrying (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onInviteSessionProgress (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *audioCodecNames*, String *videoCodecNames*, Boolean *existsEarlyMedia*, Boolean *existsAudio*, Boolean *existsVideo*, StringBuilder *sipMessage*)

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

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>audioCodecNames</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecNames</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsEarlyMedia</i>	If it's true, it indicates that the call has early media.
<i>existsAudio</i>	If it's true, it indicates that this call includes the audio.
<i>existsVideo</i>	If it's true, it indicates that this call includes the video.
<i>sipMessage</i>	The SIP message received.

Int32 Ubiix.SIPCallbackEvents.onInviteRingin (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *statusText*, Int32 *statusCode*, StringBuilder *sipMessage*)

If the outgoing call was ringing, this event would be triggered.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onInviteAnswered (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *callerDisplayName*, String *caller*, String *calleeDisplayName*, String *callee*, String *audioCodecNames*, String *videoCodecNames*, Boolean *existsAudio*, Boolean *existsVideo*, StringBuilder *sipMessage*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onInviteFailure (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *reason*, Int32 *code*, StringBuilder *sipMessage*)

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

Int32 Ubiix.SIPCallbackEvents.onInviteUpdated (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *audioCodecNames*, String *videoCodecNames*, Boolean *existsAudio*, Boolean *existsVideo*, StringBuilder *sipMessage*)

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

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>audioCodecNames</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecNames</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	If it's true, it indicates that this call includes the audio.
<i>existsVideo</i>	If it's true, it indicates that this call includes the video.
<i>sipMessage</i>	The SIP message received.

Int32 Ubiix.SIPCallbackEvents.onInviteConnected (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onInviteBeginingForward (Int32 *callbackIndex*, Int32 *callbackObject*, String *forwardTo*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onInviteClosed (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

This event is triggered once remote side closes the call.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onDialogStateUpdated (Int32 *callbackIndex*, Int32 *callbackObject*, String *BLFMonitoredUri*, String *BLFDialogState*, String *BLFDialogId*, String *BLFDialogDirection*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onRemoteHold (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onRemoteUnHold (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *audioCodecNames*, String *videoCodecNames*, Boolean *existsAudio*, Boolean *existsVideo*)

If the remote side un-hold the call, this event would be triggered.

Parameters:

<i>sessionId</i>	The session ID of the call.
<i>audioCodecNames</i>	The matched audio codecs. It's separated by "#" if there are more than one codecs.
<i>videoCodecNames</i>	The matched video codecs. It's separated by "#" if there are more than one codecs.
<i>existsAudio</i>	If it's true, it indicates that this call includes the audio.
<i>existsVideo</i>	If it's true, it indicates that this call includes the video.

Refer events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onReceivedRefer](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *referId*, String *to*, String *from*, StringBuilder *referSipMessage*)
- Int32 [Ubiix.SIPCallbackEvents.onReferAccepted](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)
- Int32 [Ubiix.SIPCallbackEvents.onReferRejected](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *reason*, Int32 *code*)
- Int32 [Ubiix.SIPCallbackEvents.onTransferTrying](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)
- Int32 [Ubiix.SIPCallbackEvents.onTransferRinging](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)
- Int32 [Ubiix.SIPCallbackEvents.onACTVTransferSuccess](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)
- Int32 [Ubiix.SIPCallbackEvents.onACTVTransferFailure](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *reason*, Int32 *code*)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onReceivedRefer (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *referId*, String *to*, String *from*, StringBuilder *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 <code>acceptRefer</code> or <code>rejectRefer</code>
<i>to</i>	The refer target.
<i>from</i>	The sender of REFER message.
<i>referSipMessage</i>	The SIP message of "REFER". Pass it to "acceptRefer" function.

Int32 Ubiix.SIPCallbackEvents.onReferAccepted (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onReferRejected (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *reason*, Int32 *code*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onTransferTrying (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onTransferRinging (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onACTVTransferSuccess (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onACTVTransferFailure (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *reason*, Int32 *code*)

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

- Int32 [Ubiix.SIPCallbackEvents.onReceivedSignaling](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, StringBuilder *signaling*)

- Int32 [Ubiix.SIPCallbackEvents.onSendingSignaling](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, StringBuilder signaling)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onReceivedSignaling (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, StringBuilder *signaling*)

This event will be triggered when receiving an SIP message.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onSendingSignaling (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, StringBuilder *signaling*)

This event will be triggered when a SIP message sent.

Parameters:

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

MWI events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onWaitingVoiceMessage](#) (Int32 callbackIndex, Int32 callbackObject, String messageAccount, Int32 urgentNewMessageCount, Int32 urgentOldMessageCount, Int32 newMessageCount, Int32 oldMessageCount)
- Int32 [Ubiix.SIPCallbackEvents.onWaitingFaxMessage](#) (Int32 callbackIndex, Int32 callbackObject, String messageAccount, Int32 urgentNewMessageCount, Int32 urgentOldMessageCount, Int32 newMessageCount, Int32 oldMessageCount)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onWaitingVoiceMessage (Int32 *callbackIndex*, Int32 *callbackObject*, String *messageAccount*, Int32 *urgentNewMessageCount*, Int32 *urgentOldMessageCount*, Int32 *newMessageCount*, Int32 *oldMessageCount*)

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

Parameters:

<i>messageAccount</i>	Voice message account
<i>urgentNewMessageCount</i>	Urgent new message count.
<i>urgentOldMessageCount</i>	Urgent old message count.

<i>newMessageCount</i>	New message count.
<i>oldMessageCount</i>	Old message count.

Int32 Ubiix.SIPCallbackEvents.onWaitingFaxMessage (Int32 *callbackIndex*, Int32 *callbackObject*, String *messageAccount*, Int32 *urgentNewMessageCount*, Int32 *urgentOldMessageCount*, Int32 *newMessageCount*, Int32 *oldMessageCount*)

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

Parameters:

<i>messageAccount</i>	Fax message account
<i>urgentNewMessageCount</i>	Urgent new message count.
<i>urgentOldMessageCount</i>	Urgent old message count.
<i>newMessageCount</i>	New message count.
<i>oldMessageCount</i>	Old message count.

DTMF events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onRecvDtmfTone](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *tone*)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onRecvDtmfTone (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *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

- Int32 [Ubiix.SIPCallbackEvents.onRecvOptions](#) (Int32 callbackIndex, Int32 callbackObject, StringBuilder optionsMessage)
- Int32 [Ubiix.SIPCallbackEvents.onRecvInfo](#) (Int32 callbackIndex, Int32 callbackObject, StringBuilder infoMessage)
- Int32 [Ubiix.SIPCallbackEvents.onRecvNotifyOfSubscription](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId, StringBuilder notifyMsg, byte[] contentData, Int32 contentLenght)
- Int32 [Ubiix.SIPCallbackEvents.onSubscriptionFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId, Int32 statusCode)
- Int32 [Ubiix.SIPCallbackEvents.onSubscriptionTerminated](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onRecvOptions (Int32 *callbackIndex*, Int32 *callbackObject*, StringBuilder *optionsMessage*)

This event will be triggered when receiving the OPTIONS message.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onRecvInfo (Int32 *callbackIndex*, Int32 *callbackObject*, StringBuilder *infoMessage*)

This event will be triggered when receiving the INFO message.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onRecvNotifyOfSubscription (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *subscribeId*, StringBuilder *notifyMsg*, byte [] *contentData*, Int32 *contentLenght*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onSubscriptionFailure (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *subscribeId*, Int32 *statusCode*)

This event will be triggered on sending SUBSCRIBE failure.

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onSubscriptionTerminated (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *subscribeId*)

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

Parameters:

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

Presence events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onPresenceRecvSubscribe](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *subscribeId*, String *fromDisplayName*, String *from*, String *subject*)
- Int32 [Ubiix.SIPCallbackEvents.onPresenceOnline](#) (Int32 *callbackIndex*, Int32 *callbackObject*, String *fromDisplayName*, String *from*, String *stateText*)
- Int32 [Ubiix.SIPCallbackEvents.onPresenceOffline](#) (Int32 *callbackIndex*, Int32 *callbackObject*, String *fromDisplayName*, String *from*)
- Int32 [Ubiix.SIPCallbackEvents.onRecvMessage](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *contentType*, String *subMimeType*, byte[] *messageData*, Int32 *messageDataLength*)
- Int32 [Ubiix.SIPCallbackEvents.onRecvOutOfDialogMessage](#) (Int32 *callbackIndex*, Int32 *callbackObject*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*, String *contentType*, String *subMimeType*, byte[] *messageData*, Int32 *messageDataLength*)
- Int32 [Ubiix.SIPCallbackEvents.onSendMessageSuccess](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *messageId*)
- Int32 [Ubiix.SIPCallbackEvents.onSendMessageFailure](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, Int32 *messageId*, String *reason*, Int32 *code*)
- Int32 [Ubiix.SIPCallbackEvents.onSendOutOfDialogMessageSuccess](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *messageId*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*)
- Int32 [Ubiix.SIPCallbackEvents.onSendOutOfDialogMessageFailure](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *messageId*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*, String *reason*, Int32 *code*)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onPresenceRecvSubscribe (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *subscribeId*, String *fromDisplayName*, String *from*, String *subject*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onPresenceOnline (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from, String stateText)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onPresenceOffline (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from)

When the contact goes 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

Int32 Ubiix.SIPCallbackEvents.onRecvMessage (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String mimeType, String subMimeType, byte [] messageData, Int32 messageDataLength)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onRecvOutOfDialogMessage (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from, String toDisplayName, String to, String mimeType, String subMimeType, byte [] messageData, Int32 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 recipient.
<i>to</i>	The recipient.
<i>mimeType</i>	The message mime type.
<i>subMimeType</i>	The message sub mime type.
<i>messageData</i>	The received message body. It can be text or binary data.
<i>messageDataLength</i>	The length of "messageData".

Int32 Ubiix.SIPCallbackEvents.onSendMessageSuccess (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 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.

Int32 Ubiix.SIPCallbackEvents.onSendMessageFailure (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 messageId, String reason, Int32 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.

Int32 Ubiix.SIPCallbackEvents.onSendOutOfDialogMessageSuccess (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *messageId*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*)

If the message is sent succeeded 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 recipient.
<i>to</i>	The message receiver.

Int32 Ubiix.SIPCallbackEvents.onSendOutOfDialogMessageFailure (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *messageId*, String *fromDisplayName*, String *from*, String *toDisplayName*, String *to*, String *reason*, Int32 *code*)

If the message was sent failure 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 recipient.
<i>to</i>	The message recipient.
<i>reason</i>	The failure reason.
<i>code</i>	The failure code.

Play audio and video file finished events

Functions

- Int32 [Ubiix.SIPCallbackEvents.onPlayAudioFileFinished](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *fileName*)
- Int32 [Ubiix.SIPCallbackEvents.onPlayVideoFileFinished](#) (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onPlayAudioFileFinished (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *sessionId*, String *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 name of the file played.

Int32 Ubiix.SIPCallbackEvents.onPlayVideoFileFinished (Int32 *callbackIndex*, Int32 *callbackObject*, Int32 *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

- Int32 [Ubiix.SIPCallbackEvents.onReceivedRtpPacket](#) (IntPtr *callbackObject*, Int32 *sessionId*, Boolean *isAudio*, byte[] *RTPPacket*, Int32 *packetSize*)
- Int32 [Ubiix.SIPCallbackEvents.onSendingRtpPacket](#) (IntPtr *callbackObject*, Int32 *sessionId*, Boolean *isAudio*, byte[] *RTPPacket*, Int32 *packetSize*)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onReceivedRtpPacket (IntPtr *callbackObject*, Int32 *sessionId*, Boolean *isAudio*, byte [] *RTPPacket*, Int32 *packetSize*)

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

Parameters:

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

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.

Int32 Ubiix.SIPCallbackEvents.onSendingRtpPacket (IntPtr *callbackObject*, Int32 *sessionId*, Boolean *isAudio*, byte [] *RTPPacket*, Int32 *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

- Int32 [Ubiix.SIPCallbackEvents.onAudioRawCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 callbackType, byte[] data, Int32 dataLength, Int32 samplingFreqHz)
- Int32 [Ubiix.SIPCallbackEvents.onVideoRawCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 callbackType, Int32 width, Int32 height, byte[] data, Int32 dataLength)
- Int32 [Ubiix.SIPCallbackEvents.onVideoDecoderCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 width, Int32 height, Int32 framerate, Int32 bitrate)

Detailed Description

Function Documentation

Int32 Ubiix.SIPCallbackEvents.onAudioRawCallback (IntPtr *callbackObject*, Int32 *sessionId*, Int32 *callbackType*, byte [] *data*, Int32 *dataLength*, Int32 *samplingFreqHz*)

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

Parameters:

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

Int32 Ubiix.SIPCallbackEvents.onVideoRawCallback (IntPtr *callbackObject*, Int32 *sessionId*, Int32 *callbackType*, Int32 *width*, Int32 *height*, byte [] *data*, Int32 *dataLength*)

This event will be triggered once receiving 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, 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.

Int32 Ubiix.SIPCallbackEvents.onVideoDecoderCallback (IntPtr *callbackObject*, Int32 *sessionId*, Int32 *width*, Int32 *height*, Int32 *framerate*, Int32 *bitrate*)

This event will be triggered once per second, with the frame rate and bit rate for the incoming stream or new incoming Video size is detected, on the premise that enableVideoDecoderCallback function is called.

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.

Namespace Documentation

Ubiix Namespace Reference

Classes

- class [Ubiix_Errors](#)
- class [UbiixLib](#)
- interface [SIPCallbackEvents](#)

Enumerations

- enum [AUDIOCODEC_TYPE](#) : int { [AUDIOCODEC_TYPE.AUDIOCODEC_NONE](#) = -1, [AUDIOCODEC_TYPE.AUDIOCODEC_G729](#) = 18, [AUDIOCODEC_TYPE.AUDIOCODEC_PCMA](#) = 8, [AUDIOCODEC_TYPE.AUDIOCODEC_PCMU](#) = 0, [AUDIOCODEC_TYPE.AUDIOCODEC_GSM](#) = 3, [AUDIOCODEC_TYPE.AUDIOCODEC_G722](#) = 9, [AUDIOCODEC_TYPE.AUDIOCODEC_ILBC](#) = 97, [AUDIOCODEC_TYPE.AUDIOCODEC_AMR](#) = 98, [AUDIOCODEC_TYPE.AUDIOCODEC_AMRWB](#) = 99, [AUDIOCODEC_TYPE.AUDIOCODEC_SPEEX](#) = 100, [AUDIOCODEC_TYPE.AUDIOCODEC_SPEEXWB](#) = 102, [AUDIOCODEC_TYPE.AUDIOCODEC_ISACWB](#) = 103, [AUDIOCODEC_TYPE.AUDIOCODEC_ISACSWB](#) = 104, [AUDIOCODEC_TYPE.AUDIOCODEC_G7221](#) = 121, [AUDIOCODEC_TYPE.AUDIOCODEC_OPUS](#) = 105, [AUDIOCODEC_TYPE.AUDIOCODEC_DTMF](#) = 101 } *Audio codec type.*
- enum [VIDEOCODEC_TYPE](#) : int { [VIDEOCODEC_TYPE.VIDEO_CODE_NONE](#) = -1, [VIDEOCODEC_TYPE.VIDEO_CODE_I420](#) = 113, [VIDEOCODEC_TYPE.VIDEO_CODE_H263](#) = 34, [VIDEOCODEC_TYPE.VIDEO_CODE_H263_1998](#) = 115, [VIDEOCODEC_TYPE.VIDEO_CODE_H264](#) = 125, [VIDEOCODEC_TYPE.VIDEO_CODE_VP8](#) = 120 } *Video codec type.*
- enum [AUDIO_RECORDING_FILEFORMAT](#) : int { [AUDIO_RECORDING_FILEFORMAT.FILEFORMAT_WAVE](#) = 1, [AUDIO_RECORDING_FILEFORMAT.FILEFORMAT_AMR](#) } *The audio record file format.*
- enum [RECORD_MODE](#) : int { [RECORD_MODE.RECORD_NONE](#) = 0, [RECORD_MODE.RECORD_RECV](#) = 1, [RECORD_MODE.RECORD_SEND](#), [RECORD_MODE.RECORD_BOTH](#) } *The audio/Video record mode.*
- enum [CALLBACK_SESSION_ID](#) : int { [Ubiix_LOCAL_MIX_ID](#) = -1, [Ubiix_REMOTE_MIX_ID](#) = -2 }
- enum [AUDIOSTREAM_CALLBACK_MODE](#) : int { [AUDIOSTREAM_CALLBACK_MODE.AUDIOSTREAM_NONE](#) = 0, [AUDIOSTREAM_CALLBACK_MODE.AUDIOSTREAM_LOCAL_MIX](#), [AUDIOSTREAM_CALLBACK_MODE.AUDIOSTREAM_LOCAL_PER_CHANNEL](#), [AUDIOSTREAM_CALLBACK_MODE.AUDIOSTREAM_REMOTE_MIX](#), [AUDIOSTREAM_CALLBACK_MODE.AUDIOSTREAM_REMOTE_PER_CHANNEL](#) } *The audio stream callback mode.*
- enum [VIDEOSTREAM_CALLBACK_MODE](#) : int { [VIDEOSTREAM_CALLBACK_MODE.VIDEOSTREAM_NONE](#) = 0, [VIDEOSTREAM_CALLBACK_MODE.VIDEOSTREAM_LOCAL](#), [VIDEOSTREAM_CALLBACK_MODE.VIDEOSTREAM_REMOTE](#), [VIDEOSTREAM_CALLBACK_MODE.VIDEOSTREAM_BOTH](#) } *The video stream callback mode.*
- enum [Ubiix_LOG_LEVEL](#) : int { [Ubiix_LOG_NONE](#) = -1, [Ubiix_LOG_ERROR](#) = 1, [Ubiix_LOG_WARNING](#) = 2, [Ubiix_LOG_INFO](#) = 3, [Ubiix_LOG_DEBUG](#) = 4 } *Log level.*
- enum [SRTP_POLICY](#) : int { [SRTP_POLICY.SRTP_POLICY_NONE](#) = 0, [SRTP_POLICY.SRTP_POLICY_FORCE](#), [SRTP_POLICY.SRTP_POLICY_PREFER](#) } *SRTP Policy.*

- enum [TRANSPORT_TYPE](#) : int { [TRANSPORT_TYPE.TRANSPORT_UDP](#) = 0, [TRANSPORT_TYPE.TRANSPORT_TLS](#), [TRANSPORT_TYPE.TRANSPORT_TCP](#), [TRANSPORT_TYPE.TRANSPORT_PERS](#) } *Transport for SIP signaling.*
- enum [SESSION_REFRESH_MODE](#) : int { [SESSION_REFRESH_MODE.SESSION_REFERESH_UAC](#) = 0, [SESSION_REFRESH_MODE.SESSION_REFERESH_UAS](#) } *The session refreshment by UAC or UAS.*
- enum [DTMF_METHOD](#) { [DTMF_METHOD.DTMF_RFC2833](#) = 0, [DTMF_METHOD.DTMF_INFO](#) = 1 } *send DTMF tone with two methods*
- enum [EC_MODES](#) { [EC_NONE](#) = 0, [EC_DEFAULT](#) = 1, [EC_CONFERENCE](#) = 2, [EC_AEC](#) = 3, [EC_AECM_1](#) = 4, [EC_AECM_2](#) = 5, [EC_AECM_3](#) = 6, [EC_AECM_4](#) = 7 } *type of Echo Control*
- enum [AGC_MODES](#) { [AGC_NONE](#) = 0, [AGC_DEFAULT](#), [AGC_ADAPTIVE_ANALOG](#), [AGC_ADAPTIVE_DIGITAL](#), [AGC_FIXED_DIGITAL](#) } *type of Automatic Gain Control*
- enum [NS_MODES](#) { [NS_NONE](#) = 0, [NS_DEFAULT](#), [NS_Conference](#), [NS_LOW_SUPPRESSION](#), [NS_MODERATE_SUPPRESSION](#), [NS_HIGH_SUPPRESSION](#), [NS_VERY_HIGH_SUPPRESSION](#) } *type of Noise Suppression*

Detailed Description

[Ubiix](#) The [Ubiix](#) VoIP SDK namespace

Enumeration Type Documentation

enum [Ubiix.AUDIOCODEC_TYPE](#) : int[strong]

Audio codec type.

Enumerator:

AUDIOCODEC_G729	G729 8KHZ 8kbit/s.
AUDIOCODEC_PCMA	PCMA/G711 A-law 8KHZ 64kbit/s.
AUDIOCODEC_PCMU	PCMU/G711 1¼-law 8KHZ 64kbit/s.
AUDIOCODEC_GSM	GSM 8KHZ 13kbit/s.
AUDIOCODEC_G722	G722 16KHZ 64kbit/s.
AUDIOCODEC_ILBC	iLBC 8KHZ 30ms-13kbit/s 20 ms-15kbit/s
AUDIOCODEC_AMR	Adaptive Multi-Rate (AMR) 8KHZ (4.75,5.15,5.90,6.70,7.40,7.95,10.20,12.20)kbit/s.
AUDIOCODEC_AMRWB	Adaptive Multi-Rate Wideband (AMR-WB)16KHZ (6.60,8.85,12.65,14.25,15.85,18.25,19.85,23.05,23.85)kbit/s.
AUDIOCODEC_SPEEX	SPEEX 8KHZ (2-24)kbit/s.
AUDIOCODEC_SPEEXWB	SPEEX 16KHZ (4-42)kbit/s.

AUDIOCODEC_I SACWB	internet Speech Audio Codec(iSAC) 16KHZ (32-54)kbit/s
AUDIOCODEC_I SACSWB	internet Speech Audio Codec(iSAC) 16KHZ (32-160)kbit/s
AUDIOCODEC_ G7221	G722.1 16KHZ (16,24,32)kbit/s.
AUDIOCODEC_ OPUS	OPUS 48KHZ 32kbit/s.
AUDIOCODEC_ DTMF	DTMF RFC 2833.

enum [Ubix.VIDEOCODEC_TYPE](#) : int[strong]

Video codec type.

Enumerator:

VIDEO_CODE_N ONE	Do not use Video codec.
VIDEO_CODEC_ I420	I420/YUV420 Raw Video format. Used with startRecord only.
VIDEO_CODEC_ H263	H263 video codec.
VIDEO_CODEC_ H263_1998	H263+/H263 1998 video codec.
VIDEO_CODEC_ H264	H264 video codec.
VIDEO_CODEC_ VP8	VP8 video code.

enum [Ubix.AUDIO_RECORDING_FILEFORMAT](#) : int[strong]

The audio record file format.

Enumerator:

FILEFORMAT_W AVE	The record audio file is in WAVE format.
FILEFORMAT_A MR	The record audio file is in AMR format - all voice data are compressed by AMR codec.

enum [Ubix.RECORD_MODE](#) : int[strong]

The audio/Video record mode.

Enumerator:

RECORD_NONE	Not Record.
-------------	-------------

RECORD_RECV	Only record the received data.
RECORD_SEND	Only record the sent data.
RECORD_BOTH	Record both received and sent data.

enum [Ubix.AUDIOSTREAM_CALLBACK_MODE](#) : int[strong]

The audio stream callback mode.

Enumerator:

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 session ID.
AUDIOSTREAM_REMOTE_MIX	Callback the received audio stream that mixed all included channels.
AUDIOSTREAM_REMOTE_PER_CHANNEL	Callback the received audio stream for one channel based on the session ID.

enum [Ubix.VIDEOSTREAM_CALLBACK_MODE](#) : int[strong]

The video stream callback mode.

Enumerator:

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.

enum [Ubix.SRTP_POLICY](#) : int[strong]

SRTP Policy.

Enumerator:

SRTP_POLICY_NONE	Do not use SRTP. The SDK can receive the encrypted call(SRTP) and unencrypted call both, but can't place outgoing encrypted call.
SRTP_POLICY_FORCE	All calls must use SRTP. The SDK allows to receive encrypted call and place outgoing encrypted call only.

SRTP_POLICY_PREFER	Top priority for using SRTP. The SDK allows to receive encrypted and decrypted call, and to place outgoing encrypted call and unencrypted call.
--------------------	---

enum [Ubiix.TRANSPORT_TYPE](#) : int[strong]

Transport for SIP signaling.

Enumerator:

TRANSPORT_UDP	UDP Transport.
TRANSPORT_TLS	Tls Transport.
TRANSPORT_TCP	TCP Transport.
TRANSPORT_PERS	PERS is the Ubiix private transport for anti SIP blocking. It must be used with the PERS Server http://www.Ubiix.com/pers.html .

enum [Ubiix.SESSION_REFRESH_MODE](#) : int[strong]

The session refreshment by UAC or UAS.

Enumerator:

SESSION_REFRESH_UAC	The session refreshment by UAC.
SESSION_REFRESH_UAS	The session refreshment by UAS.

enum [Ubiix.DTMF_METHOD](#) [strong]

send DTMF tone with two methods

Enumerator:

DTMF_RFC2833	Send DTMF tone with RFC 2833. Recommended.
DTMF_INFO	Send DTMF tone with SIP INFO.

Class Documentation

Ubiix.Ubiix_Errors Class Reference

Static Public Attributes

- static readonly int **INVALID_SESSION_ID** = -1
- static readonly int **ECoreAlreadyInitialized** = -60000
- static readonly int **ECoreNotInitialized** = -60001
- static readonly int **ECoreSDKObjectNull** = -60002
- static readonly int **ECoreArgumentNull** = -60003
- static readonly int **ECoreInitializeWinsockFailure** = -60004
- static readonly int **ECoreUserNameAuthNameEmpty** = -60005
- static readonly int **ECoreInitiazeStackFailure** = -60006
- static readonly int **ECorePortOutOfRange** = -60007
- static readonly int **ECoreAddTcpTransportFailure** = -60008
- static readonly int **ECoreAddTlsTransportFailure** = -60009
- static readonly int **ECoreAddUdpTransportFailure** = -60010
- static readonly int **ECoreMiniAudioPortOutOfRange** = -60011
- static readonly int **ECoreMaxAudioPortOutOfRange** = -60012
- static readonly int **ECoreMiniVideoPortOutOfRange** = -60013
- static readonly int **ECoreMaxVideoPortOutOfRange** = -60014
- static readonly int **ECoreMiniAudioPortNotEvenNumber** = -60015
- static readonly int **ECoreMaxAudioPortNotEvenNumber** = -60016
- static readonly int **ECoreMiniVideoPortNotEvenNumber** = -60017
- static readonly int **ECoreMaxVideoPortNotEvenNumber** = -60018
- static readonly int **ECoreAudioVideoPortOverlapped** = -60019
- static readonly int **ECoreAudioVideoPortRangeTooSmall** = -60020
- static readonly int **ECoreAlreadyRegistered** = -60021
- static readonly int **ECoreSIPServerEmpty** = -60022
- static readonly int **ECoreExpiresValueTooSmall** = -60023
- static readonly int **ECoreCallIdNotFound** = -60024
- static readonly int **ECoreNotRegistered** = -60025
- static readonly int **ECoreCalleeEmpty** = -60026
- static readonly int **ECoreInvalidUri** = -60027
- static readonly int **ECoreAudioVideoCodecEmpty** = -60028
- static readonly int **ECoreNoFreeDialogSession** = -60029
- static readonly int **ECoreCreateAudioChannelFailed** = -60030
- static readonly int **ECoreSessionTimerValueTooSmall** = -60040
- static readonly int **ECoreAudioHandleNull** = -60041
- static readonly int **ECoreVideoHandleNull** = -60042
- static readonly int **ECoreCallsClosed** = -60043
- static readonly int **ECoreCallAlreadyHold** = -60044
- static readonly int **ECoreCallNotEstablished** = -60045
- static readonly int **ECoreCallNotHold** = -60050
- static readonly int **ECoreSipMessaegEmpty** = -60051
- static readonly int **ECoreSipHeaderNotExist** = -60052
- static readonly int **ECoreSipHeaderValueEmpty** = -60053
- static readonly int **ECoreSipHeaderBadFormed** = -60054
- static readonly int **ECoreBufferTooSmall** = -60055
- static readonly int **ECoreSipHeaderValueListEmpty** = -60056
- static readonly int **ECoreSipHeaderParserEmpty** = -60057
- static readonly int **ECoreSipHeaderValueListNull** = -60058
- static readonly int **ECoreSipHeaderNameEmpty** = -60059
- static readonly int **ECoreAudioSampleNotmultiple** = -60060
- static readonly int **ECoreAudioSampleOutOfRange** = -60061
- static readonly int **ECoreInviteSessionNotFound** = -60062
- static readonly int **ECoreStackException** = -60063

- static readonly int **ECoreMimeTypeUnknown** = -60064
- static readonly int **ECoreDataSizeTooLarge** = -60065
- static readonly int **ECoreSessionNumsOutOfRange** = -60066
- static readonly int **ECoreNotSupportCallbackMode** = -60067
- static readonly int **ECoreNotFoundSubscribed** = -60068
- static readonly int **ECoreCodecNotSupport** = -60069
- static readonly int **ECoreCodecParameterNotSupport** = -60070
- static readonly int **ECorePayloadOutOfRange** = -60071
- static readonly int **ECorePayloadHasExist** = -60072
- static readonly int **ECoreFixPayloadCantChange** = -60073
- static readonly int **ECoreCodecTypeInvalid** = -60074
- static readonly int **ECoreCodecWasExist** = -60075
- static readonly int **ECorePayloadTypeInvalid** = -60076
- static readonly int **ECoreArgumentTooLong** = -60077
- static readonly int **ECoreMiniRtpPortMustIsEvenNum** = -60078
- static readonly int **ECoreCallInHold** = -60079
- static readonly int **ECoreNotIncomingCall** = -60080
- static readonly int **ECoreCreateMediaEngineFailure** = -60081
- static readonly int **ECoreAudioCodecEmptyButAudioEnabled** = -60082
- static readonly int **ECoreVideoCodecEmptyButVideoEnabled** = -60083
- static readonly int **ECoreNetworkInterfaceUnavailable** = -60084
- static readonly int **ECoreWrongDTMFTone** = -60085
- static readonly int **ECoreWrongLicenseKey** = -60086
- static readonly int **ECoreTrialVersionLicenseKey** = -60087
- static readonly int **ECoreOutgoingAudioMuted** = -60088
- static readonly int **ECoreOutgoingVideoMuted** = -60089
- static readonly int **ECoreFailedCreateSdp** = -60090
- static readonly int **ECoreTrialVersionExpired** = -60091
- static readonly int **ECoreStackFailure** = -60092
- static readonly int **ECoreTransportExists** = -60093
- static readonly int **ECoreUnsupportTransport** = -60094
- static readonly int **ECoreAllowOnlyOneUser** = -60095
- static readonly int **ECoreUserNotFound** = -60096
- static readonly int **ECoreTransportsIncorrect** = -60097
- static readonly int **ECoreCreateTransportFailure** = -60098
- static readonly int **ECoreTransportNotSet** = -60099
- static readonly int **ECoreECreateSignalingFailure** = -60100
- static readonly int **ECoreArgumentIncorrect** = -60101
- static readonly int **EAudioFileNameEmpty** = -70000
- static readonly int **EAudioChannelNotFound** = -70001
- static readonly int **EAudioStartRecordFailure** = -70002
- static readonly int **EAudioRegisterRecodingFailure** = -70003
- static readonly int **EAudioRegisterPlaybackFailure** = -70004
- static readonly int **EAudioGetStatisticsFailure** = -70005
- static readonly int **EAudioIsPlaying** = -70006
- static readonly int **EAudioPlayObjectNotExist** = -70007
- static readonly int **EAudioPlaySteamNotEnabled** = -70008
- static readonly int **EAudioRegisterCallbackFailure** = -70009
- static readonly int **EAudioCreateAudioConferenceFailure** = -70010
- static readonly int **EAudioOpenPlayFileFailure** = -70011
- static readonly int **EAudioPlayFileModeNotSupport** = -70012
- static readonly int **EAudioPlayFileFormatNotSupport** = -70013
- static readonly int **EAudioPlaySteamAlreadyEnabled** = -70014
- static readonly int **EAudioCreateRecordFileFailure** = -70015
- static readonly int **EAudioCodecNotSupport** = -70016
- static readonly int **EAudioPlayFileNotEnabled** = -70017
- static readonly int **EAudioPlayFileUnknowSeekOrigin** = -70018
- static readonly int **EAudioCantSetDeviceIdDuringCall** = -70019
- static readonly int **EAudioVolumeOutOfRange** = -70020
- static readonly int **EVideoFileNameEmpty** = -80000

- static readonly int **EVideoGetDeviceNameFailure** = -80001
- static readonly int **EVideoGetDeviceIdFailure** = -80002
- static readonly int **EVideoStartCaptureFailure** = -80003
- static readonly int **EVideoChannelNotFound** = -80004
- static readonly int **EVideoStartSendFailure** = -80005
- static readonly int **EVideoGetStatisticsFailure** = -80006
- static readonly int **EVideoStartPlayAviFailure** = -80007
- static readonly int **EVideoSendAviFileFailure** = -80008
- static readonly int **EVideoRecordUnknowCodec** = -80009
- static readonly int **EVideoCantSetDeviceIdDuringCall** = -80010
- static readonly int **EVideoUnsupportCaptureRotate** = -80011
- static readonly int **EVideoUnsupportCaptureResolution** = -80012
- static readonly int **ECameraSwitchTooOften** = -80013
- static readonly int **EMTUOutOfRange** = -80014
- static readonly int **EDeviceGetDeviceNameFailure** = -90001

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

- `Ubiix_Errors.cs`

Ubiix.UbiixLib Class Reference

Public Member Functions

- **UbiixLib** (Int32 callbackIndex, Int32 callbackObject, [SIPCallbackEvents](#) callbackevents)
- Boolean **createCallbackHandlers** ()
- void **releaseCallbackHandlers** ()
- Int32 **initialize** ([TRANSPORT_TYPE](#) transportType, String localIp, Int32 localSIPPort, [Ubiix_LOG_LEVEL](#) logLevel, String logFilePath, Int32 maxCallSessions, String sipAgentString, Int32 audioDeviceLayer, Int32 videoDeviceLayer, String TLSCertificatesRootPath, String TLSCipherList, Boolean verifyTLSCertificate)
Initialize the SDK.
- void **unInitialize** ()
Un-initialize the SDK and release resources.
- Int32 **getVersion** (out Int32 majorVersion, out Int32 minorVersion)
Get the current version number of the SDK.
- Int32 **setLicenseKey** (String key)
Set the license key. It must be called before setUser function.
- Int32 **getNICNums** ()
Get the Network Interface Card numbers.
- Int32 **getLocalIpAddress** (Int32 index, StringBuilder ip, Int32 ipSize)
Get the local IP address by Network Interface Card index.
- Int32 **setUser** (String userName, String displayName, String authName, String password, String sipDomain, String sipServerAddr, Int32 sipServerPort, String stunServerAddr, Int32 stunServerPort, String outboundServerAddr, Int32 outboundServerPort)
Set user account info.
- void **removeUser** ()
Remove the user. It will un-register from SIP server given that the user is already registered.
- Int32 **setInstancelId** (String uuid)
Set outbound (RFC5626) instancelId to be used in contact headers.
- Int32 **registerServer** (Int32 expires, Int32 retryTimes)
Register to SIP proxy server (login to server).
- Int32 **unRegisterServer** ()
Un-register from the SIP proxy server.
- Int32 **enableReliableProvisional** (Boolean enable)
Enable/disable PRACK.
- Int32 **enable3GppTags** (Boolean enable)
Enable/disable the 3Gpp tags, including "ims.icsi.mmtel" and "g.3gpp.smsip".
- void **enableCallbackSendingSignaling** (Boolean enable)
Enable/disable callback the sent SIP messages.
- void **enableCallbackReceivedSignaling** (Boolean enable)
Enable/disable callback the received SIP messages.
- Int32 **setRtpCallback** (Int32 callbackObject, Boolean enable)
Set the RTP callbacks to allow access to the sent and received RTP packets.
- Int32 **addAudioCodec** ([AUDIOCODEC_TYPE](#) codecType)
Enable an audio codec. It will be appears in SDP.
- Int32 **addVideoCodec** ([VIDEOCODEC_TYPE](#) codecType)
Enable a video codec. It will appear in SDP.
- Boolean **isAudioCodecEmpty** ()
Detect if enabled audio codecs is empty or not.
- Boolean **isVideoCodecEmpty** ()
Detect if enabled video codecs is empty or not.

- Int32 [setAudioCodecPayloadType](#) ([AUDIOCODEC_TYPE](#) codecType, Int32 payloadType)
Set the RTP payload type for dynamic audio codec.
- Int32 [setVideoCodecPayloadType](#) ([VIDEOCODEC_TYPE](#) codecType, Int32 payloadType)
Set the RTP payload type for dynamic Video codec.
- void [clearAudioCodec](#) ()
Remove all enabled audio codecs.
- void [clearVideoCodec](#) ()
Remove all enabled video codecs.
- Int32 [setAudioCodecParameter](#) ([AUDIOCODEC_TYPE](#) codecType, String parameter)
Set the codec parameter for audio codec.
- Int32 [setVideoCodecParameter](#) ([VIDEOCODEC_TYPE](#) codecType, String parameter)
Set the codec parameter for video codec.
- Int32 [setSrtpPolicy](#) ([SRTP_POLICY](#) srtpPolicy, Boolean allowSrtpOverUnsecureTransport)
Set the SRTP policy.
- Int32 [setRtpPortRange](#) (Int32 minimumRtpAudioPort, Int32 maximumRtpAudioPort, Int32 minimumRtpVideoPort, Int32 maximumRtpVideoPort)
Set the RTP ports range for audio and video streaming.
- Int32 [setRtcpPortRange](#) (Int32 minimumRtcpAudioPort, Int32 maximumRtcpAudioPort, Int32 minimumRtcpVideoPort, Int32 maximumRtcpVideoPort)
Set the RTCP ports range for audio and video streaming.
- Int32 [enableCallForward](#) (Boolean forBusyOnly, String forwardTo)
Enable call forward.
- Int32 [disableCallForward](#) ()
Disable the call forwarding. The SDK is not forwarding any incoming call after this function is called.
- Int32 [enableSessionTimer](#) (Int32 timerSeconds, [SESSION_REFRESH_MODE](#) refreshMode)
Allows to periodically refresh Session Initiation Protocol (SIP) sessions by sending INVITE requests repeatedly.
- Int32 [disableSessionTimer](#) ()
Disable the session timer.
- void [setDoNotDisturb](#) (Boolean state)
Enable the "Do not disturb" to enable/disable.
- Int32 [enableAutoCheckMwi](#) (Boolean state)
Allows to enable/disable the check MWI (Message Waiting Indication) automatically.
- Int32 [setRtpKeepAlive](#) (Boolean state, Int32 keepAlivePayloadType, Int32 deltaTransmitTimeMS)
Enable or disable to send RTP keep-alive packet when the call is established.
- Int32 [setKeepAliveTime](#) (Int32 keepAliveTime)
Enable or disable to send SIP keep-alive packet.
- Int32 [getSipMessageHeaderValue](#) (String sipMessage, String headerName, StringBuilder headerValue, Int32 headerValueLength)
Access the SIP header of SIP message.
- Int32 [addSipMessageHeader](#) (Int32 sessionId, String methodName, Int32 msgType, String headerName, String headerValue)
Add the SIP Message header into the specified outgoing SIP message.
- Int32 [removeAddedSipMessageHeader](#) (Int32 sipMessageHeaderId)
Remove the headers (custom header) added by addSipMessageHeader.
- Int32 [clearAddedSipMessageHeaders](#) ()
Clear the added extension headers (custom headers)
- Int32 [modifySipMessageHeader](#) (Int32 sessionId, String methodName, Int32 msgType, String headerName, String headerValue)
Modify the special SIP header value for every outgoing SIP message.

- Int32 [removeModifiedSipMessageHeader](#) (Int32 sipMessageHeaderId)
Remove the extension header (custom header) from every outgoing SIP message.
- Int32 [clearModifiedSipMessageHeaders](#) ()
Clear the modified headers value, and do not modify every outgoing SIP message header values any longer.
- Int32 [addSupportedMimeType](#) (String methodName, String mimeType, String subMimeType)
Set the SDK to receive the SIP messages that include special mime type.
- Int32 [setAudioSamples](#) (Int32 ptime, Int32 maxPtime)
Set the audio capture sample.
- Int32 [setAudioDeviceId](#) (Int32 recordingDeviceId, Int32 playoutDeviceId)
Set the audio device that will be used for audio call.
- Int32 [setVideoDeviceId](#) (Int32 deviceId)
Set the video device that will be used for video call.
- Int32 [setVideoResolution](#) (Int32 width, Int32 height)
Set the video capturing resolution.
- Int32 [setAudioBitrate](#) (Int32 sessionId, [AUDIOCODEC_TYPE](#) audioCodecType, Int32 bitrateKbps)
Set the audio bitrate.
- Int32 [setVideoBitrate](#) (Int32 sessionId, Int32 bitrateKbps)
Set the video bitrate.
- Int32 [setVideoFrameRate](#) (Int32 sessionId, Int32 frameRate)
Set the video frame rate.
- Int32 [sendVideo](#) (Int32 sessionId, Boolean sendState)
Send the video to remote side.
- Int32 [setVideoOrientation](#) (Int32 rotation)
Change the orientation of the video.
- void [muteMicrophone](#) (Boolean mute)
Mute the device microphone. It's unavailable for Android and iOS.
- void [muteSpeaker](#) (Boolean mute)
Mute the device speaker. It's unavailable for Android and iOS.
- void [getDynamicVolumeLevel](#) (out Int32 speakerVolume, out Int32 microphoneVolume)
Obtain the dynamic microphone volume level from current call.
- void [setChannelOutputVolumeScaling](#) (Int32 sessionId, Int32 scaling)
- void [setLocalVideoWindow](#) (IntPtr localVideoWindow)
Set the window that is used to display the local video image.
- Int32 [setRemoteVideoWindow](#) (Int32 sessionId, IntPtr remoteVideoWindow)
Set the window for a session that is used to display the received remote video image.
- Int32 [displayLocalVideo](#) (Boolean state)
Start/stop displaying the local video image.
- Int32 [setVideoNackStatus](#) (Boolean state)
Enable/disable the NACK feature (rfc6642) that helps to improve the video quality.
- Int32 [call](#) (String callee, Boolean sendSdp, Boolean videoCall)
Make a call.
- Int32 [rejectCall](#) (Int32 sessionId, int code)
rejectCall Reject the incoming call.
- Int32 [hangUp](#) (Int32 sessionId)
hangUp Hang up the call.
- Int32 [answerCall](#) (Int32 sessionId, Boolean videoCall)
answerCall Answer the incoming call.
- Int32 [updateCall](#) (Int32 sessionId, bool enableAudio, bool enableVideo)
Use the re-INVITE to update the established call.

- Int32 [hold](#) (Int32 sessionId)
To place a call on hold.
- Int32 [unHold](#) (Int32 sessionId)
Take off hold.
- Int32 [muteSession](#) (Int32 sessionId, Boolean muteIncomingAudio, Boolean muteOutgoingAudio, Boolean muteIncomingVideo, Boolean muteOutgoingVideo)
Mute the specified session audio or video.
- Int32 [forwardCall](#) (Int32 sessionId, String forwardTo)
Forward call to another one when receiving the incoming call.
- Int32 [pickupBLFCall](#) (String replaceDialogId, Boolean videoCall)
This function will be used for picking up a call based on the BLF (Busy Lamp Field) status.
- Int32 [sendDtmf](#) (Int32 sessionId, [DTMF_METHOD](#) dtmfMethod, int code, int dtmfDuration, bool playDtmfTone)
Send DTMF tone.
- Int32 [refer](#) (Int32 sessionId, String referTo)
Refer the current call to another one.
- Int32 [attendedRefer](#) (Int32 sessionId, Int32 replaceSessionId, String referTo)
Make an attended refer.
- Int32 [attendedRefer2](#) (IntPtr libSDK, Int32 sessionId, Int32 replaceSessionId, String replaceMethod, String target, String referTo)
Make an attended refer with specified request line and specified method embedded into the "Refer-To" header.
- Int32 [outOfDialogRefer](#) (Int32 replaceSessionId, String replaceMethod, String target, String referTo)
Send an out of dialog REFER to replace the specified call.
- Int32 [acceptRefer](#) (Int32 referId, String referSignalingMessage)
Accept the REFER request, and a new call will be made if called this function. The function is usually called after onReceivedRefer callback event.
- Int32 [rejectRefer](#) (Int32 referId)
Reject the REFER request.
- Int32 [enableSendPcmStreamToRemote](#) (Int32 sessionId, Boolean state, Int32 streamSamplesPerSec)
Enable the SDK to send PCM stream data to remote side from another source instead of microphone.
- Int32 [sendPcmStreamToRemote](#) (Int32 sessionId, byte[] data, Int32 dataLength)
Send the audio stream in PCM format from another source instead of audio device capturing (microphone).
- Int32 [enableSendVideoStreamToRemote](#) (Int32 sessionId, Boolean state)
Enable the SDK send video stream data to remote side from another source instead of camera.
- Int32 [sendVideoStreamToRemote](#) (Int32 sessionId, byte[] data, Int32 dataLength, Int32 width, Int32 height)
Send the video stream to remote side.
- Int32 [enableAudioStreamCallback](#) (Int32 callbackObject, Int32 sessionId, Boolean enable, [AUDIOSTREAM_CALLBACK_MODE](#) callbackMode)
Enable/disable the audio stream callback.
- Int32 [enableVideoStreamCallback](#) (Int32 callbackObject, Int32 sessionId, [VIDEOSTREAM_CALLBACK_MODE](#) callbackMode)
Enable/disable the video stream callback.
- Int32 [enableVideoDecoderCallback](#) (Int32 callbackObject, Boolean enable)
Enable/disable the video Decoder callback.

- Int32 [startRecord](#) (Int32 sessionId, String recordFilePath, String recordFileName, Boolean appendTimestamp, [AUDIO_RECORDING_FILEFORMAT](#) audioFileFormat, [RECORD_MODE](#) audioRecordMode, [VIDEOCODEC_TYPE](#) videoFileCodecType, [RECORD_MODE](#) videoRecordMode)
Start recording the call.
- Int32 [stopRecord](#) (Int32 sessionId)
Stop record.
- Int32 [playVideoFileToRemote](#) (Int32 sessionId, String fileName, Boolean loop, Boolean playAudio)
Play an AVI file to remote party.
- Int32 [stopPlayVideoFileToRemote](#) (Int32 sessionId)
Stop playing video file to remote side.
- Int32 [playAudioFileToRemote](#) (Int32 sessionId, String fileName, Int32 fileSamplesPerSec, Boolean loop)
Play a wave file to remote party.
- Int32 [stopPlayAudioFileToRemote](#) (Int32 sessionId)
Stop playing wave file to remote side.
- Int32 [playAudioFileToRemoteAsBackground](#) (Int32 sessionId, String fileName, Int32 fileSamplesPerSec)
Play a wave file to remote party as conversation background sound.
- Int32 [stopPlayAudioFileToRemoteAsBackground](#) (Int32 sessionId)
Stop playing wave file to remote party as conversation background sound.
- Int32 [createAudioConference](#) ()
Create an audio conference. It will be failed if the existent conference is not ended yet.
- Int32 [createVideoConference](#) (IntPtr conferenceVideoWindow, Int32 width, Int32 height, Boolean displayLocalVideoInConference)
Create a video conference. It will be failed if the existent conference is not ended yet.
- void [destroyConference](#) ()
End the existent conference.
- Int32 [setConferenceVideoWindow](#) (IntPtr videoWindow)
Set the window for a conference that is used to display the received remote video image.
- Int32 [joinToConference](#) (Int32 sessionId)
Join a session into existent conference. If the call is in hold, it will be un-hold automatically.
- Int32 [removeFromConference](#) (Int32 sessionId)
Remove a session from an existent conference.
- Int32 [setAudioRtcpBandwidth](#) (Int32 sessionId, Int32 BitsRR, Int32 BitsRS, Int32 KBitsAS)
Set the audio RTCP bandwidth parameters to the RFC3556.
- Int32 [setVideoRtcpBandwidth](#) (Int32 sessionId, Int32 BitsRR, Int32 BitsRS, Int32 KBitsAS)
Set the video RTCP bandwidth parameters as the RFC3556.
- Int32 [getNetworkStatistics](#) (Int32 sessionId, out Int32 currentBufferSize, out Int32 preferredBufferSize, out Int32 currentPacketLossRate, out Int32 currentDiscardRate, out Int32 currentExpandRate, out Int32 currentPreemptiveRate, out Int32 currentAccelerateRate)
Get the "in-call" statistics. The statistics are reset after the query.
- Int32 [getAudioRtpStatistics](#) (Int32 sessionId, out Int32 averageJitterMs, out Int32 maxJitterMs, out Int32 discardedPackets)
Obtain the RTP statistics of audio channel.
- Int32 [getAudioRtcpStatistics](#) (Int32 sessionId, out Int32 bytesSent, out Int32 packetsSent, out Int32 bytesReceived, out Int32 packetsReceived, out Int32 sendFractionLost, out Int32 sendCumulativeLost, out Int32 rcvFractionLost, out Int32 rcvCumulativeLost)
Obtain the RTCP statistics of audio channel.
- Int32 [getVideoRtpStatistics](#) (Int32 sessionId, out Int32 bytesSent, out Int32 packetsSent, out Int32 bytesReceived, out Int32 packetsReceived)
Obtain the RTP statistics of video.

- void [enableVAD](#) (Boolean state)
Enable/disable Voice Activity Detection (VAD).
- void [enableAEC](#) ([EC_MODES](#) ecMode)
Enable/disable AEC (Acoustic Echo Cancellation).
- void [enableCNG](#) (Boolean state)
Enable/disable Comfort Noise Generator (CNG).
- void [enableAGC](#) ([AGC_MODES](#) agcMode)
Enable/disable Automatic Gain Control (AGC).
- void [enableANS](#) ([NS_MODES](#) nsMode)
Enable/disable Audio Noise Suppression (ANS).
- Int32 [setAudioQos](#) (Boolean state, Int32 DSCPValue, Int32 priority)
Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for audio channel.
- Int32 [setVideoQos](#) (Boolean state, Int32 DSCPValue)
Set the DSCP (differentiated services code point) value of QoS (Quality of Service) for video channel.
- Int32 [setVideoMTU](#) (Int32 mtu)
Set the MTU size for video RTP packet.
- Int32 [sendOptions](#) (String to, String sdp)
Send OPTIONS message.
- Int32 [sendInfo](#) (Int32 sessionId, String mimeType, String subMimeType, String infoContents)
Send a INFO message to remote side in a call.
- Int32 [sendSubscription](#) (String to, String eventName)
Send a SUBSCRIBE message to subscribe an event.
- Int32 [terminateSubscription](#) (Int32 subscribeId)
Terminate the given subscription.
- Int32 [sendMessage](#) (Int32 sessionId, String mimeType, String subMimeType, byte[] message, Int32 messageLength)
Send a MESSAGE message to remote side in dialog.
- Int32 [sendOutOfDialogMessage](#) (String to, String mimeType, String subMimeType, byte[] message, Int32 messageLength)
Send an out of dialog MESSAGE message to remote side.
- Int32 [setDefaultSubscriptionTime](#) (Int32 secs)
Set the default expiration time to be used when creating a subscription.
- Int32 [setDefaultPublicationTime](#) (Int32 secs)
Set the default expiration time to be used when creating a publication.
- Int32 [setPresenceMode](#) (Int32 mode)
Indicate the SDK uses the P2P mode for presence or presence agent mode.
- Int32 [presenceSubscribe](#) (String to, String subject)
Send a SUBSCRIBE message for subscribing the contact's presence status.
- Int32 [presenceTerminateSubscribe](#) (Int32 subscribeId)
Terminate the given presence subscription.
- Int32 [presenceRejectSubscribe](#) (Int32 subscribeId)
Reject a presence SUBSCRIBE request which is received from contact.
- Int32 [presenceAcceptSubscribe](#) (Int32 subscribeId)
Accept the presence SUBSCRIBE request which is received from contact.
- Int32 [setPresenceStatus](#) (Int32 subscribeId, String stateText)
Set the presence status.
- Int32 [getNumOfRecordingDevices](#) ()
Gets the count of audio devices available for audio recording.
- Int32 [getNumOfPlayoutDevices](#) ()

- Gets the number of audio devices available for audio playout.*

 - Int32 [getRecordingDeviceName](#) (Int32 deviceIndex, StringBuilder nameUTF8, Int32 nameUTF8Length)
Gets the name of a specific recording device given by an index.
 - Int32 [getPlayoutDeviceName](#) (Int32 deviceIndex, StringBuilder nameUTF8, Int32 nameUTF8Length)
Get the name of a specific playout device given by an index.
 - Int32 [setSpeakerVolume](#) (Int32 volume)
Set the speaker volume level.
 - Int32 [getSpeakerVolume](#) ()
Gets the speaker volume level.
 - Int32 [setSystemOutputMute](#) (Boolean enable)
Mute the speaker device completely in the OS.
 - Boolean [getSystemOutputMute](#) ()
Retrieves the output device mute state in the operating system.
 - Int32 [setMicVolume](#) (Int32 volume)
Sets the microphone volume level.
 - Int32 [getMicVolume](#) ()
Retrieves the current microphone volume.
 - Int32 [setSystemInputMute](#) (Boolean enable)
Mute the microphone input device completely in the OS.
 - Boolean [getSystemInputMute](#) ()
Get the mute state of the input device in the operating system.
 - void [audioPlayLoopbackTest](#) (Boolean enable)
Use it for the audio device loop back test.
 - Int32 [getNumOfVideoCaptureDevices](#) ()
Get the number of available capturing devices.
 - Int32 [getVideoCaptureDeviceName](#) (Int32 deviceIndex, StringBuilder uniqueIdUTF8, Int32 uniqueIdUTF8Length, StringBuilder deviceNameUTF8, Int32 deviceNameUTF8Length)
Get the name of a specific video capture device given by an index.
 - Int32 [showVideoCaptureSettingsDialogBox](#) (String uniqueIdUTF8, Int32 uniqueIdUTF8Length, String dialogTitle, IntPtr parentWindow, Int32 x, Int32 y)
Display the capture device property dialog box for the specified capture device.

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

- UbiixLib.cs

Ubiix.SIPCallbackEvents Interface Reference

Public Member Functions

- Int32 [onRegisterSuccess](#) (Int32 callbackIndex, Int32 callbackObject, String statusText, Int32 statusCode, StringBuilder sipMessage)
- Int32 [onRegisterFailure](#) (Int32 callbackIndex, Int32 callbackObject, String statusText, Int32 statusCode, StringBuilder sipMessage)
- Int32 [onInviteIncoming](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [onInviteTrying](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onInviteSessionProgress](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsEarlyMedia, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [onInviteRinging](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String statusText, Int32 statusCode, StringBuilder sipMessage)
- Int32 [onInviteAnswered](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String callerDisplayName, String caller, String calleeDisplayName, String callee, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [onInviteFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String reason, Int32 code, StringBuilder sipMessage)
- Int32 [onInviteUpdated](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo, StringBuilder sipMessage)
- Int32 [onInviteConnected](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onInviteBeginingForward](#) (Int32 callbackIndex, Int32 callbackObject, String forwardTo)
- Int32 [onInviteClosed](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onDialogStateUpdated](#) (Int32 callbackIndex, Int32 callbackObject, String BLFMonitoredUri, String BLFDialogState, String BLFDialogId, String BLFDialogDirection)
- Int32 [onRemoteHold](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onRemoteUnHold](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String audioCodecNames, String videoCodecNames, Boolean existsAudio, Boolean existsVideo)
- Int32 [onReceivedRefer](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 referId, String to, String from, StringBuilder referSipMessage)
- Int32 [onReferAccepted](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onReferRejected](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String reason, Int32 code)
- Int32 [onTransferTrying](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onTransferRinging](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onACTVTransferSuccess](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onACTVTransferFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String reason, Int32 code)
- Int32 [onReceivedSignaling](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, StringBuilder signaling)
- Int32 [onSendingSignaling](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, StringBuilder signaling)
- Int32 [onWaitingVoiceMessage](#) (Int32 callbackIndex, Int32 callbackObject, String messageAccount, Int32 urgentNewMessageCount, Int32 urgentOldMessageCount, Int32 newMessageCount, Int32 oldMessageCount)
- Int32 [onWaitingFaxMessage](#) (Int32 callbackIndex, Int32 callbackObject, String messageAccount, Int32 urgentNewMessageCount, Int32 urgentOldMessageCount, Int32 newMessageCount, Int32 oldMessageCount)
- Int32 [onRecvDtmfTone](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 tone)
- Int32 [onRecvOptions](#) (Int32 callbackIndex, Int32 callbackObject, StringBuilder optionsMessage)
- Int32 [onRecvInfo](#) (Int32 callbackIndex, Int32 callbackObject, StringBuilder infoMessage)

- Int32 [onRecvNotifyOfSubscription](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId, StringBuilder notifyMsg, byte[] contentData, Int32 contentLength)
- Int32 [onSubscriptionFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId, Int32 statusCode)
- Int32 [onSubscriptionTerminated](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId)
- Int32 [onPresenceRecvSubscribe](#) (Int32 callbackIndex, Int32 callbackObject, Int32 subscribeId, String fromDisplayName, String from, String subject)
- Int32 [onPresenceOnline](#) (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from, String stateText)
- Int32 [onPresenceOffline](#) (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from)
- Int32 [onRecvMessage](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String mimeType, String subMimeType, byte[] messageData, Int32 messageDataLength)
- Int32 [onRecvOutOfDialogMessage](#) (Int32 callbackIndex, Int32 callbackObject, String fromDisplayName, String from, String toDisplayName, String to, String mimeType, String subMimeType, byte[] messageData, Int32 messageDataLength)
- Int32 [onSendMessageSuccess](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 messageId)
- Int32 [onSendMessageFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, Int32 messageId, String reason, Int32 code)
- Int32 [onSendOutOfDialogMessageSuccess](#) (Int32 callbackIndex, Int32 callbackObject, Int32 messageId, String fromDisplayName, String from, String toDisplayName, String to)
- Int32 [onSendOutOfDialogMessageFailure](#) (Int32 callbackIndex, Int32 callbackObject, Int32 messageId, String fromDisplayName, String from, String toDisplayName, String to, String reason, Int32 code)
- Int32 [onPlayAudioFileFinished](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId, String fileName)
- Int32 [onPlayVideoFileFinished](#) (Int32 callbackIndex, Int32 callbackObject, Int32 sessionId)
- Int32 [onReceivedRtpPacket](#) (IntPtr callbackObject, Int32 sessionId, Boolean isAudio, byte[] RTPPacket, Int32 packetSize)
- Int32 [onSendingRtpPacket](#) (IntPtr callbackObject, Int32 sessionId, Boolean isAudio, byte[] RTPPacket, Int32 packetSize)
- Int32 [onAudioRawCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 callbackType, byte[] data, Int32 dataLength, Int32 samplingFreqHz)
- Int32 [onVideoRawCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 callbackType, Int32 width, Int32 height, byte[] data, Int32 dataLength)
- Int32 [onVideoDecoderCallback](#) (IntPtr callbackObject, Int32 sessionId, Int32 width, Int32 height, Int32 framerate, Int32 bitrate)

Detailed Description

[SIPCallbackEvents](#) [Ubiix](#) VoIP SDK Callback events

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

- SIPCallbackEvents.cs

Index

- acceptRefer
 - Refer functions, 36
- Access SIP message header functions, 25
 - displayLocalVideo, 29
 - getDynamicVolumeLevel, 28
 - muteMicrophone, 28
 - muteSpeaker, 28
 - sendVideo, 27
 - setAudioBitrate, 26
 - setAudioDeviceId, 26
 - setChannelOutputVolumeScaling, 28
 - setLocalVideoWindow, 28
 - setRemoteVideoWindow, 28
 - setVideoBitrate, 27
 - setVideoDeviceId, 26
 - setVideoFrameRate, 27
 - setVideoNackStatus, 29
 - setVideoOrientation, 27
 - setVideoResolution, 26
- addAudioCodec
 - Audio and video codecs functions, 15
- Additional setting functions, 17
 - addSipMessageHeader, 22
 - clearAddedSipMessageHeaders, 22
 - clearModifiedSipMessageHeaders, 23
 - disableCallForward, 20
 - disableSessionTimer, 20
 - enableAutoCheckMwi, 21
 - enableCallForward, 19
 - enableSessionTimer, 20
 - getSipMessageHeaderValue, 21
 - modifySipMessageHeader, 23
 - removeAddedSipMessageHeader, 22
 - removeModifiedSipMessageHeader, 23
 - setDoNotDisturb, 20
 - setKeepAliveTime, 21
 - setRtcpPortRange, 19
 - setRtpKeepAlive, 21
 - setRtpPortRange, 19
 - setSrtpPolicy, 18
- addSipMessageHeader
 - Additional setting functions, 22
- addSupportedMimeType
 - Audio and video functions, 24
- addVideoCodec
 - Audio and video codecs functions, 15
- answerCall
 - Call functions, 31
- attendedRefer
 - Refer functions, 35
- attendedRefer2
 - Refer functions, 35
- Audio and video codecs functions, 15
 - addAudioCodec, 15
 - addVideoCodec, 15
 - isAudioCodecEmpty, 16
 - isVideoCodecEmpty, 16
 - setAudioCodecParameter, 17
 - setAudioCodecPayloadType, 16
 - setVideoCodecParameter, 17
 - setVideoCodecPayloadType, 16
- Audio and video functions, 24
 - addSupportedMimeType, 24
 - setAudioSamples, 24
- Audio and video stream callback events, 75
 - onAudioRawCallback, 75
 - onVideoDecoderCallback, 76
 - onVideoRawCallback, 75
- Audio effect functions, 48
 - enableAEC, 48
 - enableAGC, 49
 - enableANS, 49
 - enableCNG, 49
 - enableVAD, 48
 - setAudioQos, 49
 - setVideoMTU, 50
 - setVideoQos, 50
- AUDIO_RECORDING_FILEFORMAT
 - Ubiix, 79
- AUDIOCODEC_AMR
 - Ubiix, 78
- AUDIOCODEC_AMRWB
 - Ubiix, 78
- AUDIOCODEC_DTMF
 - Ubiix, 79
- AUDIOCODEC_G722
 - Ubiix, 78
- AUDIOCODEC_G7221
 - Ubiix, 79
- AUDIOCODEC_G729
 - Ubiix, 78
- AUDIOCODEC_GSM
 - Ubiix, 78
- AUDIOCODEC_ILBC
 - Ubiix, 78
- AUDIOCODEC_ISACSWB
 - Ubiix, 79
- AUDIOCODEC_ISACWB
 - Ubiix, 79
- AUDIOCODEC_OPUS
 - Ubiix, 79
- AUDIOCODEC_PCMA
 - Ubiix, 78
- AUDIOCODEC_PCMU
 - Ubiix, 78
- AUDIOCODEC_SPEEX
 - Ubiix, 79
- AUDIOCODEC_SPEEXWB
 - Ubiix, 79
- AUDIOCODEC_TYPE
 - Ubiix, 78
- audioPlayLoopbackTest

- Device Manage functions., 59
- AUDIOSTREAM_CALLBACK_MODE
 - Ubiix, 80
- AUDIOSTREAM_LOCAL_MIX
 - Ubiix, 80
- AUDIOSTREAM_LOCAL_PER_CHANNEL
 - Ubiix, 80
- AUDIOSTREAM_REMOTE_MIX
 - Ubiix, 80
- AUDIOSTREAM_REMOTE_PER_CHANNEL
 - Ubiix, 80
- call
 - Call functions, 30
- Call events, 61
 - onDialogStateUpdated, 64
 - onInviteAnswered, 63
 - onInviteBeginingForward, 64
 - onInviteClosed, 64
 - onInviteConnected, 64
 - onInviteFailure, 64
 - onInviteIncoming, 62
 - onInviteRinging, 63
 - onInviteSessionProgress, 63
 - onInviteTrying, 63
 - onInviteUpdated, 64
 - onRemoteHold, 65
 - onRemoteUnHold, 65
- Call functions, 29
 - answerCall, 31
 - call, 30
 - forwardCall, 32
 - hangUp, 30
 - hold, 31
 - muteSession, 32
 - pickupBLFCall, 32
 - rejectCall, 30
 - sendDtmf, 33
 - unHold, 32
 - updateCall, 31
- clearAddedSipMessageHeaders
 - Additional setting functions, 22
- clearModifiedSipMessageHeaders
 - Additional setting functions, 23
- Conference functions, 43
 - createAudioConference, 43
 - createVideoConference, 44
 - joinToConference, 44
 - removeFromConference, 44
 - setConferenceVideoWindow, 44
- createAudioConference
 - Conference functions, 43
- createVideoConference
 - Conference functions, 44
- Device Manage functions., 56
 - audioPlayLoopbackTest, 59
 - getMicVolume, 59
 - getNumOfPlayoutDevices, 57
 - getNumOfRecordingDevices, 57
 - getNumOfVideoCaptureDevices, 59
 - getPlayoutDeviceName, 57
 - getRecordingDeviceName, 57
 - getSpeakerVolume, 58
 - getSystemInputMute, 59
 - getSystemOutputMute, 58
 - getVideoCaptureDeviceName, 59
 - setMicVolume, 58
 - setSpeakerVolume, 58
 - setSystemInputMute, 59
 - setSystemOutputMute, 58
 - showVideoCaptureSettingsDialogBox, 60
- disableCallForward
 - Additional setting functions, 20
- disableSessionTimer
 - Additional setting functions, 20
- displayLocalVideo
 - Access SIP message header functions, 29
- DTMF events, 68
 - onRecvDtmfTone, 69
- DTMF_INFO
 - Ubiix, 81
- DTMF_METHOD
 - Ubiix, 81
- DTMF_RFC2833
 - Ubiix, 81
- enable3GppTags
 - NIC and local IP functions, 14
- enableAEC
 - Audio effect functions, 48
- enableAGC
 - Audio effect functions, 49
- enableANS
 - Audio effect functions, 49
- enableAudioStreamCallback
 - RTP packets, Audio stream and video stream callback functions, 38
- enableAutoCheckMwi
 - Additional setting functions, 21
- enableCallbackReceivedSignaling
 - NIC and local IP functions, 14
- enableCallbackSendingSignaling
 - NIC and local IP functions, 14
- enableCallForward
 - Additional setting functions, 19
- enableCNG
 - Audio effect functions, 49
- enableReliableProvisional
 - NIC and local IP functions, 13
- enableSendPcmStreamToRemote
 - Send audio and video stream functions, 37
- enableSendVideoStreamToRemote
 - Send audio and video stream functions, 37
- enableSessionTimer
 - Additional setting functions, 20
- enableVAD
 - Audio effect functions, 48
- enableVideoDecoderCallback

- RTP packets, Audio stream and video stream callback functions, 39
- enableVideoStreamCallback
 - RTP packets, Audio stream and video stream callback functions, 39
- FILEFORMAT_AMR
 - Ubiix, 79
- FILEFORMAT_WAVE
 - Ubiix, 79
- forwardCall
 - Call functions, 32
- getAudioRtpStatistics
 - RTP statistics functions, 47
- getAudioRtpStatistics
 - RTP statistics functions, 47
- getDynamicVolumeLevel
 - Access SIP message header functions, 28
- getLocalIpAddress
 - NIC and local IP functions, 12
- getMicVolume
 - Device Manage functions., 59
- getNetworkStatistics
 - RTP statistics functions, 46
- getNICNums
 - NIC and local IP functions, 12
- getNumOfPlayoutDevices
 - Device Manage functions., 57
- getNumOfRecordingDevices
 - Device Manage functions., 57
- getNumOfVideoCaptureDevices
 - Device Manage functions., 59
- getPlayoutDeviceName
 - Device Manage functions., 57
- getRecordingDeviceName
 - Device Manage functions., 57
- getSipMessageHeaderValue
 - Additional setting functions, 21
- getSpeakerVolume
 - Device Manage functions., 58
- getSystemInputMute
 - Device Manage functions., 59
- getSystemOutputMute
 - Device Manage functions., 58
- getVersion
 - Initialize and register functions, 10
- getVideoCaptureDeviceName
 - Device Manage functions., 59
- getVideoRtpStatistics
 - RTP statistics functions, 47
- hangUp
 - Call functions, 30
- hold
 - Call functions, 31
- INFO/OPTIONS message events, 69
 - onRecvInfo, 70
 - onRecvNotifyOfSubscription, 70
 - onRecvOptions, 70
 - onSubscriptionFailure, 70
 - onSubscriptionTerminated, 70
- initialize

- Initialize and register functions, 10
- Initialize and register functions, 9
 - getVersion, 10
 - initialize, 10
 - setLicenseKey, 11
- isAudioCodecEmpty
 - Audio and video codecs functions, 16
- isVideoCodecEmpty
 - Audio and video codecs functions, 16
- joinToConference
 - Conference functions, 44
- modifySipMessageHeader
 - Additional setting functions, 23
- muteMicrophone
 - Access SIP message header functions, 28
- muteSession
 - Call functions, 32
- muteSpeaker
 - Access SIP message header functions, 28
- MWI events, 67
 - onWaitingFaxMessage, 68
 - onWaitingVoiceMessage, 68
- NIC and local IP functions, 11
 - enable3GppTags, 14
 - enableCallbackReceivedSignaling, 14
 - enableCallbackSendingSignaling, 14
 - enableReliableProvisional, 13
 - getLocalIpAddress, 12
 - getNICNums, 12
 - registerServer, 13
 - setInstanceId, 13
 - setRtpCallback, 14
 - setUser, 12
 - unRegisterServer, 13
- onACTVTransferFailure
 - Refer events, 67
- onACTVTransferSuccess
 - Refer events, 66
- onAudioRawCallback
 - Audio and video stream callback events, 75
- onDialogStateUpdated
 - Call events, 64
- onInviteAnswered
 - Call events, 63
- onInviteBeginingForward
 - Call events, 64
- onInviteClosed
 - Call events, 64
- onInviteConnected
 - Call events, 64
- onInviteFailure
 - Call events, 64
- onInviteIncoming
 - Call events, 62
- onInviteRinging
 - Call events, 63
- onInviteSessionProgress
 - Call events, 63
- onInviteTrying

- Call events, 63
- onInviteUpdated
 - Call events, 64
- onPlayAudioFileFinished
 - Play audio and video file finished events, 73
- onPlayVideoFileFinished
 - Play audio and video file finished events, 73
- onPresenceOffline
 - Presence events, 71
- onPresenceOnline
 - Presence events, 71
- onPresenceRecvSubscribe
 - Presence events, 71
- onReceivedRefer
 - Refer events, 66
- onReceivedRtpPacket
 - RTP callback events, 74
- onReceivedSignaling
 - Signaling events, 67
- onRecvDtmfTone
 - DTMF events, 69
- onRecvInfo
 - INFO/OPTIONS message events, 70
- onRecvMessage
 - Presence events, 71
- onRecvNotifyOfSubscription
 - INFO/OPTIONS message events, 70
- onRecvOptions
 - INFO/OPTIONS message events, 70
- onRecvOutOfDialogMessage
 - Presence events, 72
- onReferAccepted
 - Refer events, 66
- onReferRejected
 - Refer events, 66
- onRegisterFailure
 - Register events, 61
- onRegisterSuccess
 - Register events, 61
- onRemoteHold
 - Call events, 65
- onRemoteUnHold
 - Call events, 65
- onSendingRtpPacket
 - RTP callback events, 74
- onSendingSignaling
 - Signaling events, 67
- onSendMessageFailure
 - Presence events, 72
- onSendMessageSuccess
 - Presence events, 72
- onSendOutOfDialogMessageFailure
 - Presence events, 73
- onSendOutOfDialogMessageSuccess
 - Presence events, 72
- onSubscriptionFailure
 - INFO/OPTIONS message events, 70
- onSubscriptionTerminated
 - INFO/OPTIONS message events, 70
- onTransferRinging
 - Refer events, 66
- onTransferTrying
 - Refer events, 66
- onVideoDecoderCallback
 - Audio and video stream callback events, 76
- onVideoRawCallback
 - Audio and video stream callback events, 75
- onWaitingFaxMessage
 - MWI events, 68
- onWaitingVoiceMessage
 - MWI events, 68
- outOfDialogRefer
 - Refer functions, 35
- pickupBLFCall
 - Call functions, 32
- Play audio and video file finished events, 73
 - onPlayAudioFileFinished, 73
 - onPlayVideoFileFinished, 73
- Play audio and video file to remote functions, 41
 - playAudioFileToRemote, 42
 - playAudioFileToRemoteAsBackground, 42
 - playVideoFileToRemote, 41
 - stopPlayAudioFileToRemote, 42
 - stopPlayAudioFileToRemoteAsBackground, 43
 - stopPlayVideoFileToRemote, 42
- playAudioFileToRemote
 - Play audio and video file to remote functions, 42
- playAudioFileToRemoteAsBackground
 - Play audio and video file to remote functions, 42
- playVideoFileToRemote
 - Play audio and video file to remote functions, 41
- Ubiix, 77
 - AUDIO_RECORDING_FILEFORMAT, 79
 - AUDIOCODEC_AMR, 78
 - AUDIOCODEC_AMRWB, 78
 - AUDIOCODEC_DTMF, 79
 - AUDIOCODEC_G722, 78
 - AUDIOCODEC_G7221, 79
 - AUDIOCODEC_G729, 78
 - AUDIOCODEC_GSM, 78
 - AUDIOCODEC_ILBC, 78
 - AUDIOCODEC_ISACSWB, 79
 - AUDIOCODEC_ISACWB, 79
 - AUDIOCODEC_OPUS, 79
 - AUDIOCODEC_PCMA, 78
 - AUDIOCODEC_PCMU, 78
 - AUDIOCODEC_SPEEX, 79
 - AUDIOCODEC_SPEEXWB, 79
 - AUDIOCODEC_TYPE, 78

- AUDIOSTREAM_CALLBACK_MODE, 80
- AUDIOSTREAM_LOCAL_MIX, 80
- AUDIOSTREAM_LOCAL_PER_CHANNEL, 80
- AUDIOSTREAM_REMOTE_MIX, 80
- AUDIOSTREAM_REMOTE_PER_CHANNEL, 80
- DTMF_INFO, 81
- DTMF_METHOD, 81
- DTMF_RFC2833, 81
- FILEFORMAT_AMR, 79
- FILEFORMAT_WAVE, 79
- RECORD_BOTH, 80
- RECORD_MODE, 79
- RECORD_NONE, 80
- RECORD_RECV, 80
- RECORD_SEND, 80
- SESSION_REFERESH_UAC, 81
- SESSION_REFERESH_UAS, 81
- SESSION_REFRESH_MODE, 81
- SRTP_POLICY, 80
- SRTP_POLICY_FORCE, 81
- SRTP_POLICY_NONE, 81
- SRTP_POLICY_PREFER, 81
- TRANSPORT_PERS, 81
- TRANSPORT_TCP, 81
- TRANSPORT_TLS, 81
- TRANSPORT_TYPE, 81
- TRANSPORT_UDP, 81
- VIDEO_CODE_NONE, 79
- VIDEO_CODEC_H263, 79
- VIDEO_CODEC_H263_1998, 79
- VIDEO_CODEC_H264, 79
- VIDEO_CODEC_I420, 79
- VIDEO_CODEC_VP8, 79
- VIDEOCODEC_TYPE, 79
- VIDEOSTREAM_BOTH, 80
- VIDEOSTREAM_CALLBACK_MODE, 80
- VIDEOSTREAM_LOCAL, 80
- VIDEOSTREAM_NONE, 80
- VIDEOSTREAM_REMOTE, 80
- Ubiix.Ubiix_Errors, 82
- Ubiix.UbiixLib, 85
- Ubiix.SIPCallbackEvents, 92
- Presence events, 70
 - onPresenceOffline, 71
 - onPresenceOnline, 71
 - onPresenceRecvSubscribe, 71
 - onRecvMessage, 71
 - onRecvOutOfDialogMessage, 72
 - onSendMessageFailure, 72
 - onSendMessageSuccess, 72
 - onSendOutOfDialogMessageFailure, 73
 - onSendOutOfDialogMessageSuccess, 72
- Presence functions, 54
 - presenceAcceptSubscribe, 55
 - presenceRejectSubscribe, 55
 - presenceSubscribe, 54
 - Presence functions, 54
 - presenceTerminateSubscribe, 54
 - setPresenceStatus, 55
- presenceAcceptSubscribe
 - Presence functions, 55
- presenceRejectSubscribe
 - Presence functions, 55
- presenceSubscribe
 - Presence functions, 54
- presenceTerminateSubscribe
 - Presence functions, 54
- Record functions, 40
 - startRecord, 40
 - stopRecord, 40
- RECORD_BOTH
 - Ubiix, 80
- RECORD_MODE
 - Ubiix, 79
- RECORD_NONE
 - Ubiix, 80
- RECORD_RECV
 - Ubiix, 80
- RECORD_SEND
 - Ubiix, 80
- refer
 - Refer functions, 34
- Refer events, 65
 - onACTVTransferFailure, 67
 - onACTVTransferSuccess, 66
 - onReceivedRefer, 66
 - onReferAccepted, 66
 - onReferRejected, 66
 - onTransferRinging, 66
 - onTransferTrying, 66
- Refer functions, 34
 - acceptRefer, 36
 - attendedRefer, 35
 - attendedRefer2, 35
 - outOfDialogRefer, 35
 - refer, 34
 - rejectRefer, 36
- Register events, 61
 - onRegisterFailure, 61
 - onRegisterSuccess, 61
- registerServer
 - NIC and local IP functions, 13
- rejectCall
 - Call functions, 30
- rejectRefer
 - Refer functions, 36
- removeAddedSipMessageHeader
 - Additional setting functions, 22
- removeFromConference
 - Conference functions, 44
- removeModifiedSipMessageHeader
 - Additional setting functions, 23
- RTP and RTCP QOS functions, 45
 - setAudioRtcpBandwidth, 45
 - setVideoRtcpBandwidth, 45
- RTP callback events, 74
 - onReceivedRtpPacket, 74

- onSendingRtpPacket, 74
- RTP packets, Audio stream and video stream callback functions, 38
 - enableAudioStreamCallback, 38
 - enableVideoDecoderCallback, 39
 - enableVideoStreamCallback, 39
- RTP statistics functions, 46
 - getAudioRtcpStatistics, 47
 - getAudioRtpStatistics, 47
 - getNetworkStatistics, 46
 - getVideoRtpStatistics, 47
- SDK Callback events, 60
- SDK functions, 9
- Send audio and video stream functions, 36
 - enableSendPcmStreamToRemote, 37
 - enableSendVideoStreamToRemote, 37
 - sendPcmStreamToRemote, 37
 - sendVideoStreamToRemote, 38
- Send OPTIONS/INFO/MESSAGE functions, 50
 - sendInfo, 51
 - sendMessage, 52
 - sendOptions, 51
 - sendOutOfDialogMessage, 53
 - sendSubscription, 51
 - setDefaultPublicationTime, 53
 - setDefaultSubscriptionTime, 53
 - setPresenceMode, 53
 - terminateSubscription, 52
- sendDtmf
 - Call functions, 33
- sendInfo
 - Send OPTIONS/INFO/MESSAGE functions, 51
- sendMessage
 - Send OPTIONS/INFO/MESSAGE functions, 52
- sendOptions
 - Send OPTIONS/INFO/MESSAGE functions, 51
- sendOutOfDialogMessage
 - Send OPTIONS/INFO/MESSAGE functions, 53
- sendPcmStreamToRemote
 - Send audio and video stream functions, 37
- sendSubscription
 - Send OPTIONS/INFO/MESSAGE functions, 51
- sendVideo
 - Access SIP message header functions, 27
- sendVideoStreamToRemote
 - Send audio and video stream functions, 38
- SESSION_REFRESH_UAC
 - Ubiix, 81
- SESSION_REFRESH_UAS
 - Ubiix, 81
- SESSION_REFRESH_MODE
 - Ubiix, 81

- setAudioBitrate
 - Access SIP message header functions, 26
- setAudioCodecParameter
 - Audio and video codecs functions, 17
- setAudioCodecPayloadType
 - Audio and video codecs functions, 16
- setAudioDeviceId
 - Access SIP message header functions, 26
- setAudioQos
 - Audio effect functions, 49
- setAudioRtcpBandwidth
 - RTP and RTCP QOS functions, 45
- setAudioSamples
 - Audio and video functions, 24
- setChannelOutputVolumeScaling
 - Access SIP message header functions, 28
- setConferenceVideoWindow
 - Conference functions, 44
- setDefaultPublicationTime
 - Send OPTIONS/INFO/MESSAGE functions, 53
- setDefaultSubscriptionTime
 - Send OPTIONS/INFO/MESSAGE functions, 53
- setDoNotDisturb
 - Additional setting functions, 20
- setInstanceId
 - NIC and local IP functions, 13
- setKeepAliveTime
 - Additional setting functions, 21
- setLicenseKey
 - Initialize and register functions, 11
- setLocalVideoWindow
 - Access SIP message header functions, 28
- setMicVolume
 - Device Manage functions., 58
- setPresenceMode
 - Send OPTIONS/INFO/MESSAGE functions, 53
- setPresenceStatus
 - Presence functions, 55
- setRemoteVideoWindow
 - Access SIP message header functions, 28
- setRtcpPortRange
 - Additional setting functions, 19
- setRtpCallback
 - NIC and local IP functions, 14
- setRtpKeepAlive
 - Additional setting functions, 21
- setRtpPortRange
 - Additional setting functions, 19
- setSpeakerVolume
 - Device Manage functions., 58
- setSrtpPolicy
 - Additional setting functions, 18
- setSystemInputMute
 - Device Manage functions., 59
- setSystemOutputMute
 - Device Manage functions., 58
- setUser

- NIC and local IP functions, 12
- setVideoBitrate
 - Access SIP message header functions, 27
- setVideoCodecParameter
 - Audio and video codecs functions, 17
- setVideoCodecPayloadType
 - Audio and video codecs functions, 16
- setVideoDeviceId
 - Access SIP message header functions, 26
- setVideoFrameRate
 - Access SIP message header functions, 27
- setVideoMTU
 - Audio effect functions, 50
- setVideoNackStatus
 - Access SIP message header functions, 29
- setVideoOrientation
 - Access SIP message header functions, 27
- setVideoQos
 - Audio effect functions, 50
- setVideoResolution
 - Access SIP message header functions, 26
- setVideoRtcpBandwidth
 - RTP and RTCP QOS functions, 45
- showVideoCaptureSettingsDialogBox
 - Device Manage functions., 60
- Signaling events, 67
 - onReceivedSignaling, 67
 - onSendingSignaling, 67
- SRTP_POLICY
 - Ubiix, 80
- SRTP_POLICY_FORCE
 - Ubiix, 81
- SRTP_POLICY_NONE
 - Ubiix, 81
- SRTP_POLICY_PREFER
 - Ubiix, 81
- startRecord
 - Record functions, 40
- stopPlayAudioFileToRemote
 - Play audio and video file to remote functions, 42
- stopPlayAudioFileToRemoteAsBackground
 - Play audio and video file to remote functions, 43
- stopPlayVideoFileToRemote
 - Play audio and video file to remote functions, 42
- stopRecord
 - Record functions, 40
- terminateSubscription
 - Send OPTIONS/INFO/MESSAGE functions, 52
- TRANSPORT_PERS
 - Ubiix, 81
- TRANSPORT_TCP
 - Ubiix, 81
- TRANSPORT_TLS
 - Ubiix, 81
- TRANSPORT_TYPE
 - Ubiix, 81
- TRANSPORT_UDP
 - Ubiix, 81
- unHold
 - Call functions, 32
- unRegisterServer
 - NIC and local IP functions, 13
- updateCall
 - Call functions, 31
- VIDEO_CODE_NONE
 - Ubiix, 79
- VIDEO_CODEC_H263
 - Ubiix, 79
- VIDEO_CODEC_H263_1998
 - Ubiix, 79
- VIDEO_CODEC_H264
 - Ubiix, 79
- VIDEO_CODEC_I420
 - Ubiix, 79
- VIDEO_CODEC_VP8
 - Ubiix, 79
- VIDEOCODEC_TYPE
 - Ubiix, 79
- VIDEOSTREAM_BOTH
 - Ubiix, 80
- VIDEOSTREAM_CALLBACK_MODE
 - Ubiix, 80
- VIDEOSTREAM_LOCAL
 - Ubiix, 80
- VIDEOSTREAM_NONE
 - Ubiix, 80
- VIDEOSTREAM_REMOTE
 - Ubiix, 80