MizuTech

MVoIPSDK -VoIP SDK for Windows

Powerful SIP client SDK for Windows

The Mizu VoIP SDK for Windows (MVoIPSDK) is a SIP standard based VoIP client that can be used as a SDK to build any SIP client application on Windows or to add VoIP call functionality to any existing application. It can be used from any framework or language such as .NET, C#, C++, Delphi and others. Based on the industry standard SIP protocol, it is compatible with all common VoIP devices, servers and softphones providing easy integration capabilities with an easy to use API for Windows desktop applications.

Contents

About

The Mizu VoIP SDK for Windows (MVoIPSDK) is a SIP client implemented as a Windows NT service which can be used from any framework or programming language including .NET, C#, C++, Delphi and many more. Since it is based on the open standard Session Initiation Protocol, it can inter-operate with any other SIP-based device (servers and clients).

Instead of a strict SDK as a library, the MVoIPSDK expose all its functionality via TCP, thus it can be used in any framework capable to initiate a simple local TCP connection. Instead of calling API functions you will have to send commands via TCP and receive the answers and other notifications on the same socket.

With the Windows VoIP SDK you have an easy to use full featured SIP/media stack easy to integrate or embed in your desktop application. For example it can be integrated with callcenter software or you can use it to add VoIP call capabilities into any software not directly related to VoIP (such as games or CRM's) or to perform any kind of VoIP automation (auto dialer, auto answer machine, etc).

Usage

The MVoIPSDK exposes the exact same API as the webphone.
The only difference is that instead of JavaScript function call, you need to send the same over TCP.

First you need download and install the MVoIPSDK (later most probably you will need to integrate that with your app installer).
Once the SDK service is running you can connect to its TCP listening port 18620 and start to send API command and receive answers and other notifications.
With other words, all you have to do is the followings:

-connect to 127.0.0.1:18620 with a tcp client socket
-send valid API commands
-parse the received notifications and adjust your app state accordingly

That's all you have to do to add VoIP capabilities to your Windows application in any framework or language.

Pseudocode:

```
//connect via TCP
tcp.Connect("127.0.0.1", 18620);
//…once connected, start the SIP stack
tcp.Send("API_Start");
//wait 1-2 seconds here to let the sip stack to initialize (or wait for the "START,api" notification)
//set multiple parameters at once (all parameters have to be separated by "CRLF")
tcp.Send("API_SetParameters,serveraddress=YOURSIPSERVER CRLF username=SIPUSERNAME CRLF password=SIPPASSWORD CRLF loglevel=5 \r\n")
//set one parameter
tcp.Send("API_SetParameter,mediatimeout,0\r\n")
//make outgoing call
tcp.end("API_Call,-1, DESTINATION\r\n");
//hangup
tcp.Send("API_Hangup\r\n");
//send IM
tcp.Send("API_SendChat,-1,DESTINATION,MESSAGE\r\n")
//on close:
//api_stop is optional, because it will time-out on no usage anyway after 3 minutes
tcp.Send("API_Stop\r\n");
//TCP disconnect
tcp.Disconnect();

//handle incoming messages
OnSocketReceivedCallback()
String recv = tcp.ReceiveString();
//parse line-by-line (split by \r\n)
//adjust your GUI based on the sip stack state machine received in STATUS messages
```

Answer for API requests are received in the following format:
For string or number return values: you will receive the return value as-is
For boolean return values: OK: FUNCTIONNAME succeed for true or ERROR: FUNCTIONNAME failed for false

Notifications are received in the following format:
WPNOTIFICATION,STATUS,-1,Registered.,NEOL

Alternative stricter format:
Instead of sending messages as simple lines, you can encapsulate lines between BOFCOMMAND / EOFCOMMAND and parameters between BOFLINE / EOFLINE. This is useful to be more strict with the message format and to easily pass multi-line parameters (such as parameters for API_SetParameters or the chat message text for API_SendChat).
Example:
Instead of using API_Call,-1,123456 you can use the following format (no need for spaces):

BOFCOMMAND BOFLINE function=API_Call EOFLINE BOFLINE line=-1 EOFLINE BOFLINE peer=123456 EOFLINE EOFCOMMAND
In this case the answer is received in the following format:
APIREQUEST: API_Call RPARAM1:-1 RPARAM2:123456 APIRESULT:true
So in the APIRESULT value you can have true/false for boolean return values, a number for int return values or a string, exactly as the API functions are defined.

See the API and the parameters sections below for the details.
Download the demo package for usage example which contains the followings:

- VoIPSDK_Test.exe: ready to use test application (run the MVoIPSDKService_Install.exe before launching this)
- ExampleC++.cpp: the important part of the C++ example test application (look only at this if you don't wish to compile the test project)
- ExampleC#.cs: the important part of the C# example test application (look only at this if you don't wish to compile the test project)
- Source folder: contains a C++ (with GUI) and a C# (console application) example with full source code

The examples are written in C++ and C#, however the code is very easy to understand and to be translated to other languages.

Features

- Standard SIP client for voice calls (in/out), chat, conference and others
- **SIP** and RTP stack compatible with any VoIP server or client (Cisco, Asterisk, gateways, ATA, softphones, IP Phones, X-Lite and many others)
- Protocols: SIP/SIPS, RTP/SRTP. Transport: UDP, TCP, TLS, TCP tunnel, SOCKS proxy traversal, HTTP proxy traversal, HTTP, VPN tunneling, **tunnel***
- NAT/Firewall support: stable SIP and RTP ports ,keep-alive, rport support, fast ICE/fast STUN protocols and auto configuration
- Peer to peer encrypted media
- RFC's: 2543, 3261, 2976, 3892, 2778, 2779, 3428, 3265, 3515, 3311, 3911, 3581, 3842, 1889, 2327, 3550, 3960, 4028, 3824, 3966, 2663, 3022 and others
- Supported methods: REGISTER, INVITE, reINVITE, ACK, PRACK, BYE, CANCEL, UPDATE, MESSAGE, INFO, OPTIONS, SUBSCRIBE, NOTIFY, REFER
- Audio codec: PCMU, PCMA, **G.729**, GSM, iLBC, SPEEX, **OPUS**
- Video codec: H264, H263, H261, MPEG1, MPEG4, MPEG2, VP8,Theora
- HD Audio: Wideband, **ultra-wideband** and full-band codecs (speex, opus)
- Audio enhancements: Stereo output (will convert mono sources to stereo) , PLC (packet loss concealment), AEC (acoustic echo canceller), Noise suppression, Silence suppression, AGC (automatic gain control) and auto QoS
- **Conference** calls (built-in RTP mixer)
- **Voice recording** (local and/or ftp upload), custom audio streaming
- DTMF (RFC2833, SIP INFO and in-band)
- **IM/Chat** (RFC 3428), BLF (Busy Lamp Field), offline messaging, group chat and **presence**
- Redial, call **hold**, mute, **forward** and **transfer** (attended and unattended)
- Call park and pickup, barge-in
- Balance display, call timer, inbound/outbound calls, Caller-ID display, Voicemail (MWI)
- Additional features: call parking, early media, local ring-back, PRACK and 100rel, replaces
- Unlimited lines
- Cross-platform
- Flexibility (all parameters/behavior can be changed/controlled by parameters and/or the API)

Requirements

OS: XP, 2000, Windows Vista, 7, 8, 10, 11. Server editions are also supported.
Audio device: Headset or microphone/speaker for audio (will work also without these, but there will be no audio)
CPU: minimum 400 MHz P3 or similar for the more complex codec's (e.g. G.729 or speex wideband) or better for conferencing
RAM: minimum 15 MB
Disk space: 15 MB
A SIP account (At any VoIP service provider or your own PBX. Can be also used without registration for peer to peer calls)
Supported languages and environments: All (C++, VB, .NET, C#, Delphi and many others)

Licensing

The Mizu VoIP SDK for Windows (MVoIPSDK) is sold with unlimited client license (Advanced and Gold) or restricted number of licenses (Basic and Standard). You can use it with any VoIP server(s) which belongs to you or your company. Your VoIP server(s) address (IP or domain name) will be hardcoded into the software to protect the licensing. After successful tests please ask for your final version at info@mizu-voip.com. Mizutech will deliver the MVoIPSDK build within one workday after your payment.

Release versions don't have any of the demo limitations and can be fully customized with your branding with "mizu" and "mizutech" words and links removed. Your final build must be used only for you company needs (including your direct sip endusers).

Title, ownership rights, and intellectual property rights in the Software shall remain with MizuTech and/or its suppliers.
The agreement and the license granted hereunder will terminate automatically if you fail to comply with the limitations described herein. Upon termination, you must destroy all copies of the Software. The software is provided "as is" without any warranty of any kind.

Some audio codecs that can be used with MVoIPSDK (e.g. G.729) can be patented in your country and require you to pay royalties to their licensors (patent license per channel). Mizutech doesn't sell codec patent licenses (but the codecs are implemented in MVoIPSDK).

You may:

Use MVoIPSDK on any number of computers
Use MVoIPSDK as an SDK embedded in your project
Give the access to MVoIPSDK for your customers or use within your company
Use MVoIPSDK on VoIP servers for which you have license for (after the agreement with Mizutech). All the VoIP servers must be owned by you or your company. Otherwise please contact our support to check the possibilities

You may not:

Resell MVoIPSDK
Sell MVoIPSDK as a standalone application (you are allowed to use it only coupled with your project which purpose should not be the same as MVoIPSDK's)
Sell "MVoIPSDK" services for third party VoIP providers and other companies
Use MVoIPSDK with VoIP servers not communicated with Mizutech
Reverse engineer, decompile or disassemble MVoIPSDK
Modify the software in any way (except modifying the parameters, skins and to add digital certificate if needed)

Demo version:

We are providing a demo version which you can try and test before any payment. The demo version has all features enabled but with some restrictions to prevent commercial usage. The limitations are the followings:

- maximum 10 simultaneous MVoIPSDK in the same time
- will expire after several months of usage (usually 2 or 3 months)
- maximum ~100 sec call duration restriction
- maximum 10 calls / session limitation (after 10 calls you will have to restart the service)
- will work maximum 20 minute after that you have to restart it or restart the service
- verifications against the mizu license service

Note: for the first few calls the limitations might be weaker than described above.
On request we can also send out demo with only the trial period limitation (will expire after several weeks of usage) and without the other limitations.

Install

Download the demo version from our website or use your own licensed build (sent to you by Mizutech on payment).
For development, just double click on the MVoIPSDKService_Install.exe and you can start working with it immediately (This will install and start the SDK service).
For production most likely you will need to integrate the SDK with your own application installer.
For this all you have to do is to add the MVoIPSDKService_Install.exe into your installer and launch it as part of the install process with the –q flag.
Alternatively you can install it on demand from your app using ShellExecute, CreateProcess or similar OS API call (when your app is launched first time or first time when the VoIP SDK is needed; see the command line examples below).
To specify a destination directory (for example to be placed in a sub-folder in your installer target directory) just pass a –dir "path" flag.
You can find a complete inno-setup example for all these in the demo package SetupExample.iis file. You can easily apply the same concepts to your favorite install builder.

Command Line Options:
-c: console mode
-q: unattended mode

The following additional options can be used with the -q flag:
-splash [text]: display a small window with progress bar instead of completely quiet
-overwrite: auto overwrites old binary files
-dir [directory]: specify installation directory (surrounded in quation marks. Ex: -dir "C:\Program Files (x86)\YourAppFolder\MVoIPSDK")

Command line examples:

- GUI install: MVoIPSDKService_Install.exe (just launch it without any parameters)
- Unattended install: MVoIPSDKService_Install.exe -q -splash "Installing VoIP Engine..."

- Silent install: MVoIPSDKService_Install.exe -q -overwrite -dir "C:\Program Files (x86)\YourAppFolder\MVoIPSDK"

For uninstall, just run it's "uninstall.exe" (which is created at install time).

API

Functions

The MVoIPSDK exposes more than 20 API entries (functions) which can be used to manipulate the SIP stack: register to server, make calls, send dtmf and chat messages and others.

These API functions are described in their internal/native format. Over TCP you will need to serialize the API requests as plain strings sent over UDP/TCP/HTTP.

Simple format:
You can issue functions calls by sending the function name and its parameters separated by comma via TCP to port 18620.
Example: API_Call,-1, 123456 \r\n

Encapsulated format:
Optionally you can use the encapsulated format when each request is encapsulated by BOFCOMMAND / EOFCOMMAND and parameters by BOFLINE / EOFLINE.
Example: BOFCOMMAND BOFLINE API_Call EOFLINE BOFLINE -1 EOFLINE BOFLINE 123456 EOFLINE EOFCOMMAND \r\n

Answers for API requests are received in the following format:
APIREQUEST:API_SetParametersAPIRESULT:true
As you can see, your API request entry is also returned in the answer. This can be useful if you are sending the API TCP request in asynchronous way, so you can always see for which request you have received the answer.
In the APIRESULT value you will receive the API function return value which can have true/false for bool return values, a number for int return values or a string, exactly as the API functions are defined below. Basically you can just extract the result as answer.GetStringBetween("APIRESULT:" , "\r") (or with similar string parsing code available in your framework).

Most of the functions return a boolean value. True when the operation was completed or initiated successfully, otherwise false.
Most of the functions are executed asynchronously. This means that it can return a true value immediately and fail later so you will need to watch the notifications about feedback from the sip stack. This it is perfectly OK to just omit all the function return values and parse only the notifications.

The **line** parameter is part of most of the function calls and means the channel number. The following values are defined:

- -2: all channels
- -1: the current channel set previously by API_SetLine or by other functions. Usually this will mean the first channel (1)
- 0 : undefined
- 1: first channel
- 2 : second channel
- etc (you can control the max number of the channels with the maxlines parameter which is set to 4 by default)

Most commonly you will have to always pass -1 as the channel number. You will have to use other values only if you will present a GUI were the user can select different lines. Otherwise MVoIPSDK can do this automatically allocating new channels when needed.
If you need to implement a multi-line application, then you need to treat the channels separately and have a state machine for each channel (not only for global state).

Function string parameters can be also passed in encrypted format. (Read the FAQ for more details regarding the encrypted parameters)

boolean API_SetParameter(String param, String value)

Most of the parameters can be set with this function. Some parameters can take effect only when MVoIPSDK is reinitialized.
This function should be used only in special cases. You should be able to control MVoIPSDK without to use this function by using parameters.
Example: tcp.Send("API_SetParameter,mediatimeout,0\r\n")

boolean API_SetParameters(String parameters)

You can pass a set of parameters with this function in value=key lines separated by \r\n. When sending over TCP this will have to be translated to "CRLF"(in simple mode) or BOFLINE/EOFLINE (encapsulated mode).
Example:
tcp.Send("API_SetParameters,serveraddress=YOURSIPSERVER CRLF username=SIPUSERNAME CRLF password=SIPPASSWORD CRLF loglevel=5\r\n");

boolean API_SetCredentials(String server, String username, String password, String authname, String displayname)

Will set the server address (ip:port or domain:port) the SIP username and the password. These values can also be preset by parameters. Parameters with empty strings will be omitted. For example if you would like to change only the username and the password, you can write API_SetCredentials(“”,”newusername”, “newpassord”)
If authname is empty, then the username will be used for authentications. The displayname is usually empty (no special displayname will be presented for peers). If other parameters are empty, then they can be specified by user input (If the VoIP SDK has a visible user interface).

boolean API_SetCredentialsMD5(String server, String username, String md5, String realm)

Instead of passing the password directly you can use MD5 checksum.
In this case the md5 parameter must be the md5 checksum for username:realm:password

The realm parameter is optional (can be set as an empty string) but it is recommended for easy error detection. If present and the server realm don’t match with this one, an error message will be displayed by MVoIPSDK.

boolean API_Start()

Will start the engine.
Example: tcp.Send("API_Start");

boolean API_StartStack()

This function call is optional to start the sip stack on demand.
If not called, then the sip stack is started anyway if the “startsipstack” parameter is set, otherwise will start at first registration or outgoing call attempt.

boolean API_Register(String server, String username, String password, String authname, String displayname)

Will connect to the SIP server. This can be also done automatically by parameter (“register”). Need to be called only once (subsequent reregistrations are done automatically. When called subsequently, than the old registrar endpoint is deleted, a new one will be created with a new call-id and MVoIPSDK will reregister). Parameters can be empty strings if you already supplied them by parameters or by the API_SetCredentials call.
If you already passed the server, username and password (or md5) parameters with the API_SetCredentials functions, then you can call this function with empty parameters: tcp.Send("API_Register");
This function have to be called only once at the startup. Further re-registrations are done automatically based on the “registerinterval” parameter. Even if you wish to force re-registration, you should not call this more frequently than 40 seconds (because up to 40 seconds might be needed for a slow registration attempt especially if tunneling is used)

boolean API_RegisterEx(String accounts)

You can use this function for multiple secondary accounts (up to 99) on the same or other servers.
The accounts parameter must have the following format: server,usr,pwd,ival;server2,usr2,pwd2, ival;
(Or just set the “extraregisteraccounts” for this string)

boolean API_Unregister()

Will stop all endpoints (hangup current calls if any and unregister)

boolean API_CheckVoicemail(int line)

Will (re)subscribe for voicemail notifications. No need to call this function if the “voicemail” parameter is set to 2.
The line parameter should be set to -1.

boolean API_SetLine(int line)

Will set the current channel. (Use only if you present line selection for the users. Otherwise you don’t have to take care about the lines).
Note: Instead of using each API call with the line parameter, you can just use this function when you wish to change the active line and use all the other API calls with -1 for the line parameter.

int API_GetLine()

Will return the current active line. This should be the line which you have set previously except after incoming and outgoing calls (MVoIPSDK will automatically switch the active line to a new free line for these if the current active line is already occupied by a call)

int API_GetLineStatus(int line) or string API_GetLineStatusText(int line)

Query the status of the line.
Note: this is rarely needed since you receive the status also by event notifications

boolean API_Call(int line, String peer)

Initiate call to a number or sip username.
If the peer parameter is empty, then will redial the last number.
Example: tcp.Send("API_Call,-1, DESTINATION\r\n");

boolean API_CallEx(int line, String peer, int calltype)

Initiate call to a number or sip username.
If the peer parameter is empty, then will redial the last number.
The calltype can have the following values:

- 0: initiate voice call
- 1: initiate video call
- 2: initiate screensharing session

boolean API_Hangup(int line, String reasontext)

Disconnect current call(s). If you set -2 for the line parameter, then all calls will be disconnected (in case if there are multiple calls in progress). The "reasontext" parameter is optional.
Example: tcp.Send("API_Hangup\r\n");

boolean API_Accept(int line)

Connect incoming call.

boolean API_Reject(int line)

Disconnect incoming call. (API_Hangup will also work)

boolean API_Forward(int line, String peer)

Forward incoming call to peer (with 302 Moved Temporarily disconnect code)

boolean API_Transfer(int line, String peer)

Transfer current call to peer which is usually a phone number or a SIP username. (Will use the REFER method after SIP standards).
You can set the mode of the transfer with the "transfertype" parameter.
If the peer parameter is empty than will interconnect the currently running calls (should be used only if you have 2 simultaneous calls)

boolean API_TransferDialog()

Instead of calling the API_Transfer function and pass a number, with this function you can let MVoIPSDK to ask the C number from the user.

boolean API_AddVideo(int line, int calltype)

Add video media for an existing voice call.
The calltype can have the following values:

- 1: add video
- 2: add screensharing

boolean API_StopVideo(int line)

Will stop the video stream at the specified line.

boolean API_MuteEx(int line, boolean mute, int direction)

Mute current call.
The line parameter is the endpoint. -2 for all or -1 for current line.
Set the mute parameter to true for mute or false to un-mute.
The direction can have the following values:

- 0: mute in and out
- 1: mute out (speakers)
- 2: mute in (microphone)
- 3: mute in and out (same as 0)

- 4: mute default (set by the “defmute” parameter, which is “mute microphone only” by default)

int API_IsMuted(int line)

Return if the selected line is muted or not.
Return values:

- -1: unknown
- 0: not muted
- 1: both muted (in/out)
- 2: out muted (speaker)
- 3: in muted (microphone)
- 4: both muted (in/out; same as 1)

int API_IsOnHold(int line)

Query if the selected line is on hold or not
Return values:

- -1: unknown
- 0: no
- 1: not used
- 2: on hold
- 3: other party held
- 4: both in hold

Note: pass -2 for the line to find if any endpoint is in hold

boolean API_Hold(int line, boolean hold)

Hold current call. This will issue an UPDATE or a reINVITE.
Set the second parameter to true for hold and false to reload.

boolean API_HoldChange(int line)

Same as API_Hold, but without the second parameter. This call will always invert the hold status for an endpoint (If the call was active, then it will switch to held status and if the call was in hold, then it will reactivate it).

boolean API_Conf(String peer)

Add people to conference.
If peer is empty than will mix the currently running calls (if there is more than one call)
Otherwise it will call the new peer (usually a phone number or a SIP user name) and once connected will join with the current session.

boolean API_Dtmf(int line, String dtmf)

Send DTMF message by SIP INFO or RFC2833 method (depending on the “dtmfmode” parameter). Please note that the dtmf parameter is a string. This means that multiple dtmf characters can be passed at once and the VoIP SDK will streamline them properly. Use the space char to insert delays between the digits. The dtmf messages are sent with the protocol specified with the “dtmfmode” parameter.
Example: API_Dtmf(-2,” 12 345 #”);

boolean API_SendChat(int line, String peer, String message)

Send a chat message. (SIP MESSAGE method after RFC 3428)
Peer can be a phone number or SIP username/extension number.

boolean API_Chat(String peer)

Instead of calling the API_SendChat function and pass a message, with this function you can let MVoIPSDK to open its chat form.
Will open the chat form (the “number” parameter can be empty)
Peer can be a SIP username or extension number.
On successful delivery the following event is sent: EVENT, chat sent successfully
On failed delivery the following event is sent: WARNING, chat message not delivered

boolean API_VoiceRecord(int startstop, int now, String filename)

Will start/stop a voice recording session.

- Startstop: 0 to stop, 1 to start locally, 2 to start remote ftp, 3 start to record both locally and to remote ftp, 4 start to record as it is set by the "voicerecording" parameter
- Now: used if the startstop is set to 0. 0 means that the recorded file will be saved and/or uploaded at the end of the conversation only. 2 means that the file will be saved immediately
- Filename: file name used for storing the recorded voice (if empty string, than will use a default file name)

This function should be used only if you would like to control the recording duration.
If all conversations have to be recorded, then just set the "voicerecording" parameter after your needs.

The last recorded call can be played by calling the API_PlaySound with the file set to "lastvoicerecord".

boolean API_PlaySound(int start, String file, int looping, boolean async, boolean islocal, boolean toremotepeer, int line, String audiodevice, boolean isring)

Play any sound file.
At least wave files are supported in the following format:
PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits)
Playback to remote peer is supported only with narrowband codec's (disable speex wideband and ultrawideband if file playback is needed)

- start: 1 for start or 0 to stop the playback, -1 to pre-cache
- file: file name
- looping: 1 to repeat, 0 to play once
- async: false if no, true if playback should be done in a separate thread
- islocal: true if the file have to be read from the client PC file system. False if remote file
- toremotepeer: stream the playback to the connected peer
- line: used with toremotepeer if there are multiple calls in progress to specify the call (usually set to -1 for the current call if any)
- audiodevice: you can specify an exact device for playback. Otherwise set it to empty string
- isring: wheter this sound is a ringtone/ringback

Examples:

-playback a file locally
-API_PlaySound(1, "mysound.wav", 0, false, false,false, -1)
-playback a file to the connected remote peer
-API_PlaySound(1, "mysound.wav", 0, false, false,true, -1)
-stop the playback:
API_PlaySound(0, "", 0, false, false,false, -1)

boolean API_StreamSoundBuff(int start, int line, byte[] buff, int len)

Stream from raw wave audio PCM (Linear PCM) buffer or from RTP packets.
This function will stream the supplied audio buffer to the peer endpoint, transcoding if necessary.

Parameters:

- start: 1 for start or 0 to stop the playback
- line: specify the channel in case if there are multiple calls in progress (usually set to -1 for the current call if any)
- buff: audio buffer (raw PCM data or RTP packet)
- len: length of the buff (ideally it should be multiple of 360 if raw pcm or the size of the RTP packets which are to be passed separately). If len is 0 or negative and start is 1, then len will be set from buff.length.

The buff byte buffer can be passed as Base64 encoded UTF-8 string with the following characters replaced:

'=' -> '-'
'+' -> '_'
'/' -> '.'

Notes:

- *The buff should not contain any file header if you are using raw PCM (only raw linear PCM audio data) and it should contain a full RTP packet (with the RTP header) if you are passing RTP data.*
- *You might set the useaudiodevicerecord parameter to false if you need streaming but don't have an audio recorder device installed.*
- *If previously supplied buffer(s) are not completed yet, then the new one will be queued.*
- *If you are streaming in real-time (feeding the API with short RTP/audio packets) then you might set the realtimeplayback parameter to 1.*
- *Alternatively you can use the API_StreamSoundStream function to pass an InputStream instead of byte buffers.*
- *In case if you wish to send (also) video, then use the API_SendVideoRTP function.* See the video guide and the video streaming guide for more details.
- *In case if you wish to receive the audio stream from the remote peer endpoint, see this FAQ point instead.*

Buffer/stream format:

At least raw linear PCM is supported in 8kHz 16 bit mono format: PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits). In case if your buffer is narrowband (8 kHz 16 bit mono), then the stream will be automatically converted to wideband if the VoIP audio codec is wideband such as opus or speex. In case if your buffer is wideband (16 kHz 16 bit mono) then you should set the playbacksteamformat parameter to 1 to allow auto conversion to narrowband if the VoIP audio codec is narrowband such as G.711 (PCMU, PCMA), G.729 or GSM. With raw PCM the supplied buffer length should be a multiply of 160 or 320, otherwise extra bytes might be dropped. The function can accept also RTP packets and will transcode them if necessary. Set the streamsoundisrtp parameter to -1 to auto-guess raw PCM vs RTP (this is the default value), set to 0 if you supply raw PCM buffer or set to 1 if you supply RTP packets.

For more details see the "How to send a media stream" FAQ point in the JVoIP documentation.

byte[] API_GetMedia(int timeout)

Retrieve the next media (audio or video) packet as a byte buffer from the JVoIP internal queue.
The buffer format (prefix bytes) can be specified with the related sendmedia_... parameters as described in the streaming FAQ.

The byte buffer will be returned as Base64 encoded UTF-8 string. Replace the following characters before to decode:

'-' -> '='
'_' -> '+'
'.' -> '/'

This is a blocking function call, so you should call it from a separate thread.
The timeout can be specified in milliseconds. If it is higher then 0, then JVoIP will use an additional waiting lock. If 0 then will issue a simple/fast blocking read.
The internal queue size is 900 and on overflow it will remove all queued packets (if you are not calling this function or not fast enough).

The internal media packets queueing will start only if the sendmedia_mode is set to 2 or 3 or at first call to this function.
(The first call for this function will also set the sendmedia_mode parameter to 2 or 3 if it was not set before)
If you don't need media streaming anymore, then you might call the API_GetMediaEnd() function to stop queuing any more media packets, otherwise it will stop automatically after some time of no usage or overflow.

The receive byte buffer might contain some header bytes as specified by the sendmedia_ parameters.

For more details see the "How to get the media stream" FAQ point in the JVoIP documentation.

string API_GetAudioDeviceList(int dev)

Will return the list of audio devices separated by \r\n.
Set the dev parameter to 0 to list the recording device names. Set to 1 for to get the playback or ringer devices.
Note: usually you don't need to use this function. Just call the "API_AudioDevice" to let the users to change their audio settings.

string API_GetAudioDevice(int dev)

Will return the currently selected audio device for the dev line (dev values are 0 for recording, 1 for playback, 2 for ringer).

boolean API_SetAudioDevice(int dev, string devicename,int immediate)

Select an audio device. The devicename should be a valid audio device name (you can list them with the API_GetAudioDeviceList call)
The "dev" parameter can have the following values:

0: recording device (microphone)
1: playback device (speaker, headset)
2: ringer device (speaker, headset)

The "immediate" parameter can have the following values:

0: default (after the "changeaudiodevimmediate" parameter)
1: next call only
2: immediately for active calls

Note:

For the ringer device you can also pass the "All" string as the devicename to make MVoIPSDK to ring on all devices for incoming calls.
All devices can also accept the "Default" devicename which will select the system default audio device.
For the device you can also pass a number which is the order of the audio device as returned by API_GetAudioDeviceList staring from 1. Valid values are between 1 and 9 or 0 for the system default device.
Usually you don't need to use this function. Just call the "API_AudioDevice" to let the users to change their audio settings.

boolean API_SetVolume(int dev,int volume)

Set volume (0-100%) for the selected device.
The dev parameter can have the following values:

0 for the recording (microphone) audio device
1 for the playback (speaker) audio device
2 for the ringback (speaker) audio device

int API_GetVolume(int dev)

Return the volume (0-100%) for the selected device.
The dev parameter can have the following values:
0 for the recording (microphone) audio device
1 for the playback (speaker) audio device
2 for the ringback (speaker) audio device

String API_VAD()

Returns voice activity statistics. See the VAD notification for more details.
Note: if you call this function, VAD will not be sent automatically anymore. (So you will need to continue to poll for the details).

string API_GetVersion()

Return the program version number.

string API_GetStatus(int line, int strict)

Returns line status or global status if you pass -2 as line parameter. The possible returned texts are the same like for notifications (listed below).
If the strict variable is set to 1, then it will return "Unknown" if no such line is activated. If the strict variable is set to 0, then it will return the default active line if the line doesn't exists.
You should use the notifications described below to get the actual status of MVoIPSDK *instead of continuously polling it with this function call.*

string API_Poll()

You might call this function periodically (from a timer at around 1 second) instead of using Webphonetojs events.
It will return exactly the same strings like the notification events. Accumulated events since the last call will be separated by \r\n.
The "jsscriptpoll" parameter must be set to 1,2 or 3 for this to work. This can also be used as a workaround if notifications are not working in the target environment.

Contacts

Contacts are normally handled by the caller process (your app) because they have less to do with the VoIP stack (except presence status). This should be done easily from your application and better adopted to your needs. You can use the API to add VoIP functionality to your address book or contact list. The status (Online/Offline/Busy) of the users can be loaded from your VoIP server database. Based on the user presence you can display the different buttons with your design. Near each contact you can display a call/chat button which will launch a MVoIPSDK instanced preconfigured with the actual contact ("callto" parameter).

However for your convenience, the VoIP SDK also provides a simple contact management API.

Contacts parameters are stored as comma delimited strings with the following parameters:
imstatus,name,sip,phone,phone2,phone3,othernumbers,sipcontacturi,email,web,address,speeddial,extra,internalextra

boolean API_SetContacts(String contacts)
Set all contacts (contact parameters separated by new line)

String API_GetContacts()
Will return all contacts (in separated lines the parameters described above)

boolean API_DelContact(String name)
Delete contact.

boolean API_AddContact(String params)
Add a contact. Example: Add_Contact(',John Smith,jsmith,'); //here we set only the name and the SIP fields

boolean API_SetContact(String name, String params)
Change contact.

String API_GetContact(String name)
Will return a single contact in the format described above.

Helper functions to set/get individual fields:

boolean API_SetContactName(String name, String param)
Set contact name.

boolean API_SetContactSIP(String name, String param)
Set contact sip uri.

boolean API_SetContactPhone(String name, String param)
Set contact phone number.

boolean API_SetContactSpeedDial(String name, String param)
Set contact speed dial number (short number).

String API_GetContactName(String name)
Get contact name.

String API_GetContactSIP(String name)
Get contact sip uri.

String API_GetContactPhone(String name)
Get contact phone number.

String API_GetContactSpeedDial(String name)
Get contact speed dial number (short number).

Presence

Presence is based on SIP SIMPLE SUBSCRIBE/NOTIFY mechanism and it is used to detect the online status of the contacts.
There is no need to manage the contacts within MVoIPSDK (as described above) to have presence functionality (so you can manage the contacts externally in your application).
The following steps are required:

1. Related settings:
   - enablepresence: 0/1
   - email = email address sent with contact info
   - presenceexpiresec = 3600
   - autoacceptpresencrequests = -1; //-1: not set (1), 0=auto reject all,1=ask for new users,2=yes, autoaccept new unknown users
2. First you should call API_PushContactlist to pass all the usernames and phonenumbers from your external contact list if any. This is necessary, because for existing contacts MVoIPSDK can accept the requests automatically, while for other it might ask for user permission
3. On first start you might call API_NumExists. If using the Mizu VoIP server, then it will return all existing contacts with SERVERCONTACTS,userlist notification where userlist are populated with the valid users and their online status.
4. Call API_CheckPresence(userlist). To save server resources, you should carefully select the contacts. (Send only the contacts which are actually used and called numbers. We recommend up to 50 contacts. If the user select a contact, then you can call this function later with that single contact to request its status)
5. Use the API_SetPresenceStatus(statustring) function call to change the user online status with one of the followings strings: Online, Away, DND, Invisible , Offline (case sensitive)
6. Once these are done, the following notifications can be received from MVoIPSDK:
   NEWUSER,username,displayname,email
   PRESENCE, status,username,displayname,email
   (displayname and email can be empty)

   On newuser, you should ask the user if wish to accept it. If accepted, call the API_NewUser function. The same function should be called when the user adds a new contact to its contactlist.
   For presence the following status strings are defined (be prepared to receive any of these and handle it with case insensitive by displaying red/green/gray/other icons):
   - Open/Online/Reachable/Available/Call Me/Registered [GREEN]
   - DND (Do not disturb; halt popups and sounds) [RED]
   - Busy/Speaking (can be auto set) [ORANGE/YELLOW]
   - Pending/Forwarding [ORANGE/YELLOW]
   - Away/Idle [ORANGE/YELLOW]
   - Close/Unreachable/Offline/Unregistered [GREY/WHITE]
   - Unknown/Not Set [GREY/WHITE/NOCOLOR]
   - Invisible (no status notifications will be sent) [GREY/WHITE/NOCOLOR]

   Other suggestion for colors:
   - red: busy/dnd
   - bright green: online
   - pale green: away/forwarding/pending
   - white: user exists but unknown status or invisible
   - no color: user doesn't exists / no presence feature

Miscellaneous

Some other not so important API calls are listed below:

boolean API_Test()
You might use this function to check the API availability. Should return true.

int API_TestEx()
You might use this function to check the API availability. Should return 42.

boolean API_Test()
You might use this function to check the API availability (should return true).

boolean API_HTTPKeepAlive()
You should call this function periodically more frequently than the timeout specified by the "httpsessiontimeout" parameter. (For example call this in every 5 minute). This is to prevent orphaned MVoIPSDK instances (when your html page was closed or crashed but MVoIPSDK is still running in the background).
If you don't wish to call this function periodically, then you should set the "httpsessiontimeout" parameter to 0.

boolean API_ServerInit(String address) -deprecated
Call this function before to start any communication with this address (usually an IP number). This is required to release the security restrictions. Wait 1-2 second before calling the next function like API_Register or API_Call. This function is deprecated since v.3.8 (no need to call this, just call API_Register or others directly)

boolean API_Stop()
Will stop all endpoints. This function call is optional when you unload MVoIPSDK from external app.

boolean API_Exit()
Will stop all endpoints and terminates MVoIPSDK. This function call is optional when you unload MVoIPSDK or wish to issue a forced termination. Its behavior can be controlled by the "exitmethod" parameter.

boolean API_CapabilityRequest(String server, String username)
Will send an OPTION request to the server. Usually you should not use this function.
The server parameter can be empty if you already set it with other API calls or by parameter.
The username parameter can be empty (in this case the "From" address will be set to "unknown")

boolean API_SetSIPHeader(int line, String hdr)
Set a custom sip header (a line in the SIP signaling) that will be sent with all messages. Can be used for various integration purposes (for example for sending the http session id). Multiple headers can be separated by CRLF (\r\n). You can also set this with parameter (customsipheader).

String API_GetSIPHeader(int line, String hdr)
Return a sip header value received by MVoIPSDK. If not found it will return a string beginning with "ERROR:" such as "ERROR: no such line".

boolean API_SendSIP(String msg)
Will send a custom SIP signaling message (for example OPTIONS, NOTIFY, etc). The message will be sent within 1-3 seconds after the function call is completed.

String API_GetLastRecSIPMessage(String line)
Get the last received SIP message as clear text. Line is the line number or the SIP call id.

boolean API_IsOnline()
Return true if network is present

boolean API_IsRegistered()
Return true if MVoIPSDK is registered ("connected") to the SIP server.

int API_IsInCall()
Return whether the sip stack is in call: 0=no,1=ringing,2=speaking

int API_GetCurrentConnectedCallCount()
Get number of current connected calls

String API_GetRegFailReason(boolean extended)
Will return a text about the reason of the last failed registration. Set the extended parameter to true to get more details

String API_GetCallerID(int line)
Will return the remote party name

String API_GetIncomingDisplay(int line)
Get incoming caller id (might return two lines: caller id \n caller name)

String API_GetLastCallDetails()
Get details about the last finished call.

boolean API_ShowLog()
Show a new window with logs.

void API_AddLog(String msg)
Add a message to MVoIPSDK log.

String API_HTTPGet(String url)
Send a HTTP GET request. Check if return string begins with “ERROR”.

boolean API_HTTPPost(String url, String data)
Send a HTTP POST request.

String API_HTTPReq(String url, String data)
Send a HTTP POST or GET request. (If data is empty, then GET will be sent). Can be tunneled. Can block for up to 20 seconds.

boolean API_HTTPReqAsync(String url, String data)
Send a HTTP POST or GET request. (If data is empty, then GET will be sent). Can be tunneled. The result will be returned in notifications with “ANSWER” header.

boolean API_SaveFile(String filename, String content)
Will save the text file to local disk MVoIPSDK working directory in encrypted format (use API_SaveFileRaw to save as-is)

String API_LoadFile(String filename)
Load file from local disk.

boolean API_SaveFileRemote(String filename, String content)
Save file to remote storage (preconfigured ftp or http server)

boolean API_LoadFileRemote(String filename)
Load file from remote storage. The download process is performed asynchronously. You need to call this function only once and then a few seconds later call the API_LoadFile function with the same file name. It will contain “ERROR: reason” text if the download failed.

String API_LoadFileRemoteSync(String filename)
Will download the specified file from remote storage synchronously (will block until done or fails).

String API_GetBlacklist()
Get blacklist

boolean API_SetBlacklist(String str)
Set whole blacklist (users/numbers separated by comma)

boolean API_AddToBlacklist(String str)
Add to blacklist

Notifications

Notifications are received on the same TCP socket where you are sending commands.
The most important are the “STATUS” notifications which will tell you about the sipstack state-machine so you can adjust your app logic and user interface accordingly.
Usually you will have to parse the received strings from your code. The parameters are separated by comma ‘,’. You can get more than one event at once, separated by newline (or ,NEOL \r\n) so you should check for new lines first and then parse all events.

Notifications might be prefixed with the “WPNOTIFICATION,” string and suffixed by the “,NEOL” string (you should handle and remove these before parsing the rest of the line).

First you have to check the first parameter (until the first comma) to determine the event type. Then you have to check for the other parameters according to the specification below.
The best is to try to log out all messages at first and from there the usage should be more obvious.

The following messages are defined:

STATUS,line,statustext,peername,localname,endpointtype

Where line can be -1 for general status or a positive value for the different lines.

STATUS notifications will tell you about the sipstack state-machine so you can adjust your app logic and/or user interface accordingly.
General status (with line parameter set to -1) means the status for the "best" endpoint.
This means that you will usually see the same status twice (or more). Once for general MVoIPSDK status and once for line status.
For example you can receive the following two messages consecutively:

STATUS,1,Connected,peername,localname,endpointtype,peerdisplayname,[callid]
STATUS,-1,Connected

You might decide to parse only general status messages (where the line is -1).
The following **statustext** values are defined **for general status (line set to -1):**

- Ready
- Register…
- Registering…
- Register Failed
- Registered
- Accept
- Starting Call
- Call
- Call Initiated
- Calling…
- Ringing…
- Incoming…
- In Call (xxx sec)
- Hangup
- Call Finished
- Chat

Note: general status means the "best" status among all lines. For example if one line is speaking, then the general status will be "In Call".

The following **statustext** values are defined **for individual lines** (line set to a positive value representing the channel number starting with 1):

- Unknown (you should not receive this)
- Init (MVoIPSDK started)
- Ready (sip stack started)
- Outband (notify/options/etc. you should skip this)
- **Register** (from register endpoints)
- Subscribe (presence)
- Chat (IM)
- **CallSetup** (one time event: call begin)
- Setup (call init)
- InProgress (call init)
- Routed (call init)
- Ringing (SIP 180 received or similar)
- **CallConnect** (one time event: call was just connected)
- InCall (call is connected)
- Muted (connected call in muted status)
- Hold (connected call in hold status)
- Speaking (call is connected)
- Midcall (might be received for transfer, conference, etc. you should treat it like the Speaking status)
- **CallDisconnect** (one time event: call was just disconnected)
- Finishing (call is about to be finished. Disconnect message sent: BYE, CANCEL or 400-600 code)
- Finished (call is finished. ACK or 200 OK was received or timeout)
- Deletable (endpoint is about to be destroyed. You should skip this)
- Error (you should not receive this)

You will usually have to display the call status for the user, and when a call arrives you might have to display an accept/reject button.
For simplified call management, you can just check for the one-time events (CallSetup, CallConnect, CallDisconnect)

Peername is the other party username (if any)
Localname is the local user name (or username).
Endpointtype is 1 from client endpoints and 2 from server endpoints.
Peerdisplayname is the other party display name if any
CallID: SIP session id

For example the following status means that there is an incoming call ringing from 2222 on the first line:

STATUS,1,Ringing,2222,1111,**2**,Katie,[callid]

The following status means an outgoing call in progress to 2222 on the second line:

STATUS,2,Speaking,2222,1111,**1**,[callid]

To display the "global" phone status, you will have to do the followings:

1. Parse the received string (parameters separated by comma)
2. If the first parameter is "STATUS" then continue

3. Check the second parameter. It "-1" continue otherwise nothing to do
4. Display the third parameter (Set the caption of a custom control)
5. Depending on the status, you might need to do some other action. For example display your "Hangup" button if the status is between "Setup" and "Finishing" or popup a new window on "Ringing" status if the endpointtype is "2" (for incoming calls only; not for outgoing)

If the "jsscripstats" is on (set to a value higher than 0) then you will receive extended status messages containing also media parameters at the end of each call:
STATUS,1,Connected,peername,localname,endpointtype,peerdisplayname,**rtpsent,rtprec,rtploss,rtplosspercet,serverstats_if_received**,[callid]

PRESENCE,peername,state,details,displayname,email

This notification is received for presence changes (peers online status).
Peername: username of the peer
State and details: presence status string; one of the followings:
CallMe,Available,Open,Pending,Other,CallForward,CallSetup,Speaking,Busy,Idle,DoNotDisturb,DND,Unknown,Away,Offline,Closed,Close,Unreacheable, Unregistered,Invisible,Exists,NotExists,Unknown,Not Set or as reported by the peer
Displayname: peer full name (it can be empty)
Email: peer email address (it can be empty)

Notes for the state and details fields:
One of these fields might be empty in some circumstances and might not be a string in the above list (especially the details).
The details field will provide a more exact description (for example "Unreachable") while the state field will provide a more exact one (for example "Close"). For this reason if you have a presence control to be changed, check the details string first and if you can't recognize its content, then check the state string. For displaying the state as text, you should display the details field (and display the state field only if the details string is empty).

CHAT,line,peername,text

This notification is received for incoming chat messages.
Line: used phone line
Peername: username of the peer
Text: the chat message body

CHATREPORT,line,peername,status,statustext

Chat transmission status report so you can process if outgoing message deliver was successfully or failed.
Line: used phone line
Peername: username of the peer
Status: 1 means sending in progress, 2 means successfully sent, 3 means failed
Statustext: error text if any

CHATCOMPOSING,line,peername,status

Chat transmission status report so you can process if outgoing message deliver was successfully or failed.
Line: used phone line
Peername: username of the peer
Status: 1 means other party is typing, 2 means other party is idle or stopped typing

CDR,line, peername,caller, called,peeraddress,connecttime,duration,discparty

After each call, you will receive a CDR (call detail record) with the following parameters:
Line: used phone line
Peername: other party username, phone number or SIP URI
Caller: the caller party name (our username in case when we are initiated the call, otherwise the remote username, displayname, phone number or URI)
Called: called party name (our username in case when we are receiving the call, otherwise the remote username, phone number or URI)
Peeraddress: other endpoint address (usually the VoIP server IP or domain name)
Connecttime: milliseconds elapsed between call initiation and call connect
Duration: milliseconds elapsed between call connect and hangup (0 for not connected calls. Divide by 1000 to obtain seconds.)
Discparty: the party which was initiated the disconnect: 0=not set, 1= MVoIPSDK, 2=peer, 3=undefined
Disconnect reason: a text about the reason of the call disconnect (SIP disconnect code, CANCEL, BYE or some other error text)

START,what

This message is sent immediately after startup (so from here you can also know that the SIP engine was started successfully).
The what parameter can have the following values:
"api" -api is ready to use
"sip" –sipstack was started

EVENT,TYPE,txt

Important events which should be displayed for the user.
The following TYPE are defined: EVENT, WARNING, ERROR
This means that you might receive messages like this:
WPNOTIFICATION,EVENT,EVENT,any text NEOL \r\n

These messages will be received only if you set the "jsscriptevent" parameter to 2. (by default is set to 2)

POPUP,txt

Should be displayed for the users in some way.

ACTION,txt

Various custom messages. Ignore.

LOG,TYPE,txt

Detailed logs (may include SIP signaling).
The following TYPE are defined: EVENT, WARNING, ERROR

These messages will be received only if you set the "jsscriptevent" parameter to 3. (by default is set to 2)

LOG,RTP,txt

RTP statistics.
Example: RTP: sent 15695 lasts: 0 (p2p: 0 sdp: 0 rrp: 15695 tnl: 15701), rec: 17281 lastr: 0, loss: 201 1%, cpu: 0.0%, cpurel: 0.0% (0.0), srvsent: 10326 srvrec: 10302 srvloss: 10 0%

VAD,parameters

Voice activity.
This is sent in around every 2000 milliseconds (2 seconds) by default (configurable with the vadstat_ival parameter in milliseconds) if you set the "vadstat" parameter to 3 or it can be requested by API_VAD. Also make sure that the "vad" parameter is set to at least "2".
This notification can be used to detect speaking/silence or to display a visual voice activity indicator.

Format:
VAD,local_vad: ON local_avg: 0 local_max: 0 local_speaking: no remote_vad: ON remote_avg: 0 remote_max: 0 remote_speaking: no

Parameters:
local_vad: whether VAD is measured for microphone: ON or OFF
local_avg: average signal level from microphone
local_max: maximum signal level from microphone
local_speaking: local user speak detected: yes or no

remote_vad: whether VAD is measured from peer to speaker out: ON or OFF
remote_avg: average signal level from peer to speaker out
remote_max: maximum signal level from peer to speaker out
remote_speaking: peer user speak detected: yes or no

Other notifications

Format: messageheader, messagetext. The followings are defined:

- "**START**" [api/sip]. You should start to call the API only after you receive the START,api notification.
- "**CREDIT**" messages are received with the user balance status if the server is sending such messages.
- "**RATING**" messages are received on call setup with the current call cost (tariff) or maximum call duration if the server is sending such messages.
- "**MWI**" messages are received on new voicemail notifications if you have enabled voicemail and there are pending new messages
- "**PRESENCE**" peer online status
- "**SERVERCONTACTS**" contact found at local VoIP server
- "**NEWUSER**" new user request
- "**ANSWER**" answer for previous request (usually http requests)

Parameters

Parameters can be specified in the following ways:

- using the API_SetParameters function to pass all of them at once
- using the API_SetParameter function
- config file: wpcfg.ini file in ini file format
- SIP signaling (sent from server) with the x-mparam header (or x-mparamp if need to persist). Example: x-mparam=loglevel=5;aec=0

Any of these methods can be used or they can be even mixed.
All parameters can be passed as strings and will be converted to the proper type internally by the VoIP SDK.
Parameters can be also encrypted. See the "Parameter security" section for the details.

For a basic usage you will have to set only your VoIP server ip or domain name ("serveraddress" parameter)
The rest of the parameters are optional and should be changed only if you have a good reason for it.

Note: once a parameter is set, it might be cached by the SDK and used even if you remove it later. To prevent this, set the parameter to "DEF" or "NULL". So instead of just deleting, set its value to "DEF" or "NULL". "DEF" means that it will use the parameter default value if any. "NULL" means empty for strings, otherwise the parameter default value. Example: transport=DEF

Main Parameters

The parameters can be used to control the most important settings and behavior like server domain, SIP authentication parameters, called party number and whether a call have to be started immediately as the SDK starts or you let the user to enter these parameters manually.

serveraddress

(string)
The domain name or IP address of your SIP server. By default it uses the standard SIP port (5060). If you need to connect to other port, you can append the port after the address separated by colon.
Examples:
"mydomain.com" -this will use the default SIP port: 5060
"sip.mydomain.com:5062"
"10.20.30.40:5065"
Default value is empty.

If not set, then you (or the users) will be able to call only using full SIP URI and it is more difficult to accept incoming calls. If set, then any username/phone number can be called what is accepted by the server.

username

(string)
This is the SIP username (A number/Caller-ID for the outgoing calls). MVoIPSDK will authenticate with this username on your server.
When compact is true, then this parameter must be filled properly. Otherwise it can be empty or omitted so the users will have to type it.
Default value is empty.

Note: If you need a different name for SIP user name and Auth name (authorization name) then you might have to also use the "sipusername" parameter.

password

(string)
SIP authentication password. When compact is true, then this parameter (and also the username) must be filled properly. Otherwise it can be empty or omitted (the user will have to enter its password). This parameter can be set also in encrypted format or you can use the md5 parameter instead of the password. More details about the parameters encryption can be found in the "Parameter security" section.
Default value is empty.

Other Parameters

These parameters are more rarely used or should be used only if you have at least a minimal technical knowledge about VoIP. *You should modify only those parameters which you fully understand otherwise better if you leave all with the default values (the default values are already optimized for production).*

register

(number)
With this parameter you can set whether MVoIPSDK should register (connect) to the sip server.
0: no
1: auto guess (yes if username/password is set, otherwise no)
2: yes

MVoIPSDK will also reregister automatically based on the “registerinterval” parameter.
Default value is 1.

autocall

(boolean)
If set to true then the VoIP SDK will immediately starts the call with the given parameters (for example with your page load). The serveraddress, username, password, callto must be also set for this to work.
Default value is false.

callto

(string)
Can be any phone number/username that can be accepted by your server or a SIP URI. When “autocall” or “compact” is true, then this parameter should be filled properly. Otherwise it can be empty or omitted (the user will enter the number to call)
Default value is empty.

use_rport

(number)
Check rport in SIP signaling (requested and received from the SIP server by the VIA header)
0=don’t ask (rtport request will not be added to the VIA header)
1=use only for symmetric NAT (only when it is sure that the public address will be correct)
2=always (always request and use the returned value except if already on public ip)
3=request even on public IP (meaningless in most cases)
9=request with the signaling, but don’t use the returned value (good If you want to keep the local IP and for peer to peer calls)
Change to 0 or 2 only if you have NAT issues (depending on your server type and settings)
(Usually use_rport and use_fast_stun should have the same value set)
Default is 1

use_fast_ice

(number)
Fast ICE negotiations (for p2p rtp routing):
0=no (set to 0 only if your server needs to always route the media)
1=auto
2=yes
3=always (not recommended)
Default is 1
Note: if set to 1 or 2 then the stun should not be disabled

use_fast_stun

(number)
Fast stun request on startup.
-1=force private address (if the client has both private and public IP, than the private IP will be sent in the signaling)
0=no
1=use only for stable symmetric NAT (recommended)
2=use only for symmetric NAT
3=always
4=use even on public IP
Change if you have NAT issues (depending on your server type and settings)
(Usually use_fast_stun and use_rport should have the same value set)
Default is 1

udpconnect

(number)
Specify whether the UDP have to be connected before sending on the socket. (Some server might require udp connect and in this way the VoIP SDK can always detect its local address correctly. However this should not be used whit multiple servers or separate domain and outbound proxy)
0=no
1=on init
2=on first send (not recommended. can block)
3=on both or any (not recommended)
Default value: 0

keepalivetype

(number)
NAT keep-alive packet type.
0=no keep-alive
1=space + CRLF (_\r\n) (very efficient because low bandwidth and low server usage)
2=NOTIFY (standard method)
3=CRLF (\r\n) (very efficient because low bandwidth and low server usage. Not recommended)
4=CRLFCRLF (\r\n\r\n) (very efficient because low bandwidth and low server usage. After RFC draft)

Default is 4.

keepaliveival

(number)
NAT keep-alive packet send interval in milliseconds.
Set to 0 to disable.
Default value is 28000. (28 sec)

registerinterval

(number)
Registration interval in **seconds** (used by the re-registration expires timer).
If your server supports keep-alive messages (to prevent NAT binding timeouts), then you might set to a longer interval (~3600 sec) to prevent high CPU usage on your server especially if you have many hundreds of SIP UA running at the same time. If your server doesn't support keep-alive, then you might set this to a lower value (between 30 and 90 sec. 60 sec is a good choice for most NAT devices and routers). Note that usually this is not necessary because server side support is not needed to keep the NAT bindings.
The actual resend of the REGISTER messages will be sent at a shorter interval the cover the UDP packet loss and network/server delays.
Hide the connect/register button or don't call the API_Register function to disable registrations.
Valid range is between 1 and 30000. (If below 10, then 120 sec will be used).
Default value is 120.

needunregister

(boolean)
Set to false to prevent unregister messages to be sent by MVoIPSDK.
Default is true

acceptsrvexpire

(number)
Accept the expires interval sent by the server.
0: no
1: yes (prioritize the contact expire)
2: yes (prioritize the global expire)
Default value is 1.

changesptoring

(number)
If to treat session progress (183) responses as ringing (180). This is useful because some servers never sends the ringing message, only a session progress and might start to send in-band ringing (or some announcement)
The following values are defined:
0: do nothing,
1: change status to ring

2: start to ring, start local ring and be ready to accept media (which is usually a ringtone or announcement)
3: start media receive and playback (and media recording if the “earlymedia” parameter is set)
4: change status to ringing and start media receive and playback (and media recording if the “earlymedia” parameter is set to true)
Default value is 2.

*Note: on ringing status of the windows SDK is able to generate local ringtone. However this locally generated ringtone playback is stopped immediately when media is started to be received from the server (allowing the user to hear the server ringback tone or announcements)

natopenpackets

(number)
Change this option only if you have RTP setup issues with your server(s).
UDP packets to send to open the NAT device and initiate the RTP. Some servers will require at least 5 packets before starting to send the media after the 183 “session in progress” response. In this case set this value to 10 (In this way the server will receive at least 5 packets even on high packet loss networks)
0: no
1: write only an empty udp packet
2: write a normal RTP packet
3 or more: write this number of RTP packets

Default is 2

*Note: instead of sending more “fake” packets, you can set the “earlymedia” to 1 or more to begin the rtp stream immediately.
*Note: you can use the “natopenpackettype” to specify the format. 1 means short CRLFCRLF packet (default). 2 means full RTP packet with zeroed content.

earlymedia

(number)
Start to send media when session progress is received.
0: no
1: reserved
2: auto (will early open audio if wideband is enabled to check if supported)
3: just early open the audio
4: null packets only when sdp received
5: yes when sdp received
6: always forced yes
Default is 2.

*Note: For the early media to work, MVoIPSDK has to open the NAT when SDP is received. This can be done by sending a few fake rtp packets or by starting to send the media immediately when session in progress is received. The first method consume less bandwidth, but it is not supported by some servers.

setfinalcodec

(number)
Some server cannot handle the final codec offer in the ACK message correctly.
In this case you will have to set this setting to 0, otherwise you will have one way audio.
0=never (RFC compliant)
1=auto guess (not send in case of certain servers and autocorrect in subsequent calls)
2=when multiple codecs are received
3=always reply with the final codec in the ACK message
Default value is 1.

proxyaddress

(string)
Outbound proxy address (Examples: mydomain.com, mydomain.com:5065, 10.20.30.40:5065)
Leave it empty if you don’t have a stateless proxy. (Use only the serveraddress parameter)
Default value is empty.

usehttpproxy

(number)
Used only for HTTP tunneling with Mizu VoIP servers.
0: no
1: same as sip proxy (proxyaddress)

2: system default
3: manual (must be set by the httpproxyurl parameter –deprecated after version 3.5)
4: auto
Default value is 4.

httpserveraddress

(string)
Useful when the transport parameter is set to 4 (auto) to specify the http tunneling gateway address.
Default value is null (address loaded from the “serveraddress” parameter)

transport

(number)
Transport protocol.
-1: auto guess
0: UDP (User Datagram Protocol. The most commonly used transport for SIP)
1: TCP (signaling via TCP. RTP will remain on UDP)
2: TLS (encrypted signaling)
3: HTTP tunneling (both signaling and media. Supported only by mizu server or mizu tunnel)
4: HTTP proxy connect (requires tunnel server)
5: Auto (automatic failover from UDP to HTTP if needed)
Default is -1.

mediaencryption

(number)
Media encryption method
0: not encrypted (default)
1: auto (will encrypt if initiated by other party)
2: SRTP
3: ZRTP (optional module)
Default is 0.

strictsrtp

(number)
Media encryption method
0: most compatible
1: default auto
2: strict (might disconnect if peer is not respect the standard or on protocol error)
3: allow only strict SRTP call, otherwise disconnect
Default: 1

has_video

(boolean)
Enable/disable video features.
You must download the phone video package for this to work and read its documentation for more details. The video module is provided “as is”. Mizutech currently doesn’t provide direct technical support for this functionality.

dtmfmode

(number)
DTMF send method
0=disabled
1=sip INFO method
2=RFC2833 in the RTP (if RTP stream is working, otherwise it will send also SIP INFO)
3=both INFO and RFC2833
4=RFC2833 (will not send SIP INFO even if there is no RTP stream negotiated)

Default is 2.

voicemail

(number)
Subscribe to voicemail notifications (MWI). Accepted values:
0=disabled
1=display voicemail only if NOTIFY is received automatically without subscription
2=auto-detect if voicemail SUBSCRIBE is needed
3=subscribe for voicemail messages after successful registration
4=subscribe for voicemail messages on startup
Default value is 2. Set to 3 if your server has support for MWI to be sure that MVoIPSDK will check the voicemail.

voicemailnum

(String)
Specify the voicemail address. Most servers will automatically send the voicemail access number so you don't need to set this parameter.

transfertype

(number)
-1=default transfer type (same as 6)
0=call transfer is disabled
1=transfer immediately and disconnect with the A user when the Transf button is pressed and the number entered (unattended transfer)
2=transfer the call only when the second party is disconnected (attended transfer)
3=transfer the call when MVoIPSDK is disconnected from the second party (attended transfer)
4=transfer the call when any party is disconnected except when the original caller was initiated the disconnect (attended transfer)
5=transfer the call when MVoIPSDK is disconnected from the second party. Put the caller on hold during the call transfer (standard attended transfer)
6=transfer the call immediately with hold and watch for notifications (unattended transfer)

Default is -1 (which is the same as 6)
If you have any incompatibility issue, then set to 1 (unattended is the simplest way to transfer a call and all sip server and device should support it correctly)

transfwithreplace

(number)
Specify if replace should be used with transfer so the old call (dialog) is not disconnected but just replaced.
This way the A party is never disconnected, just the called party is changed. The A party must be able to handle the replace header for this.
-1=auto
0=no
1=yes
Default is -1

allowreplace

(int)
Allow incoming replace requests.
0=no
1=yes and always disconnect old ep
2=yes and don't disconnect if in transfer
3=yes but never disconnect old ep
Default is 2.

discontransfer

(number)
Specify if line should disconnect after transfer
-1=auto
0=never
1= on C party connected status
2= on timeout
3= on connected or timeout
4= on ok for refer
Default is -1

disconincomingrefer

(number)
Specify if line should disconnect after transfer
-1=auto
0=no
1= yes

Default is -1

transferdelay

(number)
Milliseconds to wait before sending REFER/INVITE while in transfer.
Default value is 400.

checksubscriptionstate

(number)
Specify if line should disconnect after transfer
-1=auto
0=no
1= disconnect
2= reload

Default is -1

subscribefortransfer

(number)
Specify if line should disconnect after transfer
-1=auto
0=never
1= if no notify received
2= always

Default is -1

checksrvrecords

(number)
SRV DNS record lookup setting:
-1: auto (will check SRV record for the VoIP server, but remembers if fails and will not check again next time)
0: don't lookup (will use only A record)
1: lookup A record first. If fails then lookup srv record (because mostly the srv record is not set anyway)
2: lookup SRV record first for VoIP server address. If fails then lookup A record (RFC compliant)
3: always lookup SRV record first. If fails then lookup A record
4: check also without the _sip._udp. prefix
If the SRV lookup returns multiple records, than MVoIPSDK will failover to the next server on connection failures.
Default value is -1.

dnslookup

(number)
Domain record lookup mode
0=auto (same as 2)
1=yes always re-query
2=use cache if needed (default)
3=use cache whenever possible
4=from cache only
5=disable
Default value is 0.

audiodevicein

(string)
Audio device name for recording (microphone). Set to a valid device name or "Default" which would select the system default audio device.

audiodeviceout

(string)
Audio device name for playback (speaker). Set to a valid device name or "Default" which would select the system default audio device.

audiodevicering

(string)
Audio device name for ringtone. Set to a valid device name or “Default” which would select the system default audio device. You can also set it to “All” to have the ringtone played on all devices.

playring

(number)
Generate ringtone for incoming and outgoing calls.
0=no (you can generate ringtone also by using the api to playback a sound file when you receive ringing notifications)
1=play ringtone for incoming calls
2=play ringtone for incoming and outgoing calls. (ringtone for outgoing calls can be generated also by your VoIP sever. When remote ringtone is received, MVoIPSDK will stop the local ringtone playback immediately and starts to play the received ringtone or announcement)
Default is 2.

ringtone

(string)
Specify a ringtone sound file to be used. If not specified, then MVoIPSDK will use its own built-in ringtone for call alert.
The file should be in the following format: PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits).
You can use any sound editor to convert your file to this format (usually from File menu -> Save as).
Default value is empty.

ringincall

(number)
Ring while in call if incoming or outgoing call
0=No
1=Only a beep for incoming call
2=Yes, normal ring
Default value is 1

volumein

(number)
Default microphone volume in percent from 0 to 100. 0 means muted. 100 means maximum volume, 0 is muted.
Default is 50% (not changed)

Note: The result volume level might be affected by the AGC if it is enabled.

volumeout

(number)
Default speaker volume in percent from 0 to 100. 0 means muted. 100 means maximum volume, 0 is muted.
Default is 50% (not changed)

Note: The result volume level might be affected by the AGC if it is enabled.

volumering

(number)
Default ringback volume in percent from 0 to 100. 0 means muted. 100 means maximum volume, 0 is muted.
Default is 50% (not changed)

checkvolumesettings

(number)
Check if system volume settings are too low or muted and increase if yes.
0: no
1: at first run
2: always
Default: 2

beeponconnect

(number)
Will play a short sound when calls are connected
0=Disabled
1=For auto accepted incoming calls
2=For incoming calls
3=For outgoing calls
4=For all calls
Default value is 0

agc

(number)
Automatic gain control.
0=Disabled
1=For recording only
2=Both for playback and recording
3=Guess
Default value is 3

This will also change the effect of the volumein and volumeout settings.
For the AGC to work the mediaench module must be also deployed. See the related FAQ section for more details.
Download: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip

stereomode

(boolean)
Set to true for 2 audio channel or false for 1 (mono).
When stereo is set, the VoIP SDK will convert also mono sources to stereo output.
Default is false.

plc

(boolean)
Enable/disable packet loss concealment
Default is true (enabled)

vad

(number)
Enable/disable voice activity detection.
0: auto
1: no
2: yes for player (will help the jitter)
3: yes for recorder
4: yes for both
Default is 2.

Note: this is automatically set to 4 if the aec2 algorithm is used.

If you wish to use VAD related statistics in your application, you might have to also set the “vadstat” parameter after your needs. Possible values: 0=no,1=auto (default),2=detect no mic audio,3=send statistics. See the VAD notification and API_VAD for more details.

aec

(number)
Enable/disable acoustic echo cancellation
0=no
1=yes except if headset is guessed
2=yes if supported
3=forced yes even if not supported (might result in unexpected errors)
Default is 1.

For this AEC to work the mediaench module must be also deployed. See the related FAQ section for more details.

Download: http://www.mizu-voip.com/Portals/0/Files/mediaench.zip

aec2

(number)
Secondary AEC algorithm.
0=no
1=auto
2=yes
3: yes with extra (this might be too much under normal circumstances)
Default is 1

Note: for full echo cancellation you can set the aec to 1,2 or 3 and aec2 to 2 or 3.

denoise

(number)
Noise suppression.
0=Disabled
1=For recording only
2=Both for playback and recording
3=Auto guess
Default value is 3

For this to work the mediaench module must be also deployed. See the related FAQ section for more details.
Download: http://www.mizu-voip.com/Portals/0/Files/mediaench.zip

silencesupress

(number)
Enable/disable silence suppression
Usually not recommended unless your bandwidth is really bad and expensive.
-1=auto
0=no (disabled)
1=yes
Default is -1 (which means no, except mobile devices with low bandwidth)

rtcp

(boolean)
Enable/disable rtcp. (RFC 3550. Partial support)

use_gsm

(number)
GSM codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1.

use_ilbc

(number)
iLBC codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1.

use_speex

(number)
Narrowband speex codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

use_speexwb

(number)
Wideband speex codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

Note: to enable wideband in all circumstances, set the "disablewbforpstn" and "disablewbonmac" parameters to false.

use_speexuwb

(number)
Ultra wideband speex codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

Note: to enable wideband in all circumstances, set the "disablewbforpstn" and "disablewbonmac" parameters to false.

use_opus

(number)
Narrowband (8000 Hz) opus codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

use_opuswb

(number)
Wideband (16000 Hz) opus codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

use_opusswb

(number)
Super wideband (24000 Hz) opus codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

use_opusuwb

(number)
Ultra wideband (fullband at 48000 Hz) opus codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 1

disablewbonmac

(boolean)
Whether to disable wideband codec on mac devices.
Mac OS X has a JVM bug which prevents to reopen the audio devices with different sample rate.
Set this to false only if you are using wideband codec for each calls (so there is no chance that a call have to be handled in narrowband)
Default value is true

disablewbforpstn

(int)
This setting will disable speex and opus wideband and ultrawideband for outgoing calls to regular phone numbers since these are usually not supported for pstn calls and they might requires longer initialization.
0: no
1: check at first call
2: check all calls
Default is 1. Set to 0 to never disable wideband.

use_g729

(number)
G.729 codec setting. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 2
**In some countries a license/patent is required if you use G.729 so enable only if you have licenses or licenses are not required in your case (consult your lawyer if you are not sure)*

use_pcma

(number)

G711alaw codec. 0=never,1=don't offer,2=yes with low priority,3=yes with high priority
Default is 2

use_pcmu

(number)
G711ulaw codec. 0=never, 1=don't offer, 2=yes with low priority, 3=yes with high priority
Default is 1

alwaysallowlowcodec

(number)
Set to 2 to always put low computational and low bandwidth codec in the offer list, specifically GSM and PCMU. Low CPU or bandwidth devices might choose these codecs (such as a mobile phone on 3G).
Set to 0 to disable.
Default is 1 (auto)

codecframecount

(number)
Number of payloads in one UDP packet.
By default it is set to 0 which means 2 frames for G729 and 1 frame for all other codec.

udptos

(number)

Sets traffic class or type-of-service octet in the IP header for packets sent from UDP socket which can be used to fine-tune the QoS in your network. As the underlying network implementation may ignore this value applications should consider it a hint.

The value must be between 0 and 255.
Valid values:

- 0: disabled
- 1: automatic (set to 10 under normal conditions and disabled when in tunneling)
- 2: low-cost routing
- 4: reliable routing
- 8: throughput optimized routing
- 10: low-delay routing
- or'ing the above values (from above 2)

Default value is 1.

Notes:

for Internet Protocol v4 the value consists of an number with precedence and TOS fields as detailed in RFC 1349. The TOS field is bitset created by bitwise-or'ing values such the following :

IPTOS_LOWCOST: 2
IPTOS_RELIABILITY: 4
IPTOS_THROUGHPUT: 8
IPTOS_LOWDELAY: 16

The last low order bit is always ignored as this corresponds to the MBZ (must be zero) bit.
For Internet Protocol v6 tc is the value that would be placed into the sin6_flowinfo field of the IP header.

Under Windows OS this has to be enabled by setting the HKEY_LOCAL_MACHINE\SYSTEM\CurrentControlSet\Services\TcpIp\Parameters\DisableUserTOSSetting registry value to 0.

automute

(number)
Specify if other lines will be muted on new call
0=no (default)
1=on incoming call
2=on outgoing call
3=on incoming and outgoing calls
4=on other line button click
Default is 0

autohold

(number)
Specify if other lines will be muted on new call
0=no (default)
1=on incoming call
2=on outgoing call
3=on incoming and outgoing calls
4=on other line button click
Default is 0

holdontransfer

(number)
Specify if line should disconnect after transfer
-1=auto
0=no
1= yes
2= hold and reload if needed
Default is -1

holdtypeonhold

(number)
Specify how to hold
-2=no
-1=auto (defaults to 2)
0=no
1=not used
2=hold
3=other party hold,
4=both in hold

Default is -1

defmute

(number)
Default mute direction
0: both
1: mute out (speakers)
2: mute in (microphone)
3: both
4: both
5: disable mute

ackforauthrequest

(number)
If to send ACK for authentication requests (401,407).
0=no
1=yes (default)
Should be changed only if you have compatibility issues with the server used.

favorizecontactaddr

(number)
You may change it if you have compatibility issues with stateless proxies
0=never
1=no
2= conform RFC
3= yes. Sending for both server and contact URI (default)
4=always

prack

(boolean)
Enable 100rel (PRACK)
Set to false if you have incompatibility issues.
Default is false.

sendmac

(boolean)
Will send the client MAC address with all signaling message in the X-MAC header parameter.
Default value is false.

customsipheader

(string)
Set a custom sip header (a line in the SIP signaling) that will be sent with all messages. Can be used for various integration purposes (for example for sending the http session id). You can also change this parameter runtime with the API_SetSIPHeader function.
Default value is empty.

techprefix

(string)
Add any prefix for the called numbers.
Default is empty.

mustconnect

(boolean)
If set to true, than users must register before to make any calls.
Default value is false.

rejectonbusy

(boolean)
Set to true to reject all incoming call if there is already a call in progress.
Default value is false.

callforwardonbusy

(String)
Specify a number where calls should be forwarded when the user is already in a call. (Otherwise the new call alert will be displayed for the user or a message will be sent on the API)
Default is empty.

callforwardalways

(String)
Specify a number where ALL calls should be forwarded.
Default is empty.

calltransferalways

(String)
Specify a number where ALL calls should be transferred.
This might be used if your server doesn't support call forward (302 answers).
Default is empty.

autoignore

(int)
Set to ignore all incoming calls.
0=don't ignore
1=silently ignore
2=reject
Default value is 0.

autoaccept

(boolean)
Set to true to automatically accept all incoming calls (auto answer).
Default value is false.

Note: Autoanswer can be also forced from the server by the "P-Auto-Answer: normal" SIP header.

blacklist

(string)
Block incoming communication from these users. (users/numbers separated by comma).
Default value is empty.

rejectcallto

(int)
Set to ignore calls if target doesn't match
0=accept all incoming calls
1=check if target user match
2=check rinstance
3=check rinstance strict
4=check all strict
Default value is 0.

hideautocall

(int)
Set to 1 to suppress notifications (STATUS, CDR) from automatically handled calls (ignored, forwarded, rejected and similar).
0=send status notifications also about auto handled calls
1=do not send status notifications from auto handled calls
Default is 0.

ringtimeout

(number)
Maximum ring time allowed in millisecond.
Default is 90000 (90 second)

calltimeout

(number)
Maximum speech time allowed in millisecond.
Default is 10800000 (3 hours)

startsipstack

(number)
Specify whether to automatically start the sip stack.
0=no (the sipstack will be started on the first register or call event)
1=on startup if not tunneling or serveraddress/username/password are set (the sipstack will be started at app init)
2=on startup always (the sipstack will be started at app init)
Other=seconds (the sipstack will be started after the specified seconds)
Default value is 1

timer

(number)
You can slow down or speed up the SIP protocol timers with this setting. You may set it to 15 if you have a slow server or slow network.
Default value is 10.

timer2

(number)

Same as “timer” but it affects idle, connect and ring timeout and maximum call durations.
Default value is 10.

mediatimeout

(number)
RTP timeout in seconds to protect again dead sessions.
Calls will be disconnected if no media packet is sent and received for this interval.
You might increase the value if you expect long call hold or one way audio periods.
Set to 0 to disable call cut off on no media.
Default value is 300 (5 minute)

mediatimeout_notify

(number)
RTP timeout in seconds for API notify.
After this timeout a warning message is sent via notifications without any further action.
The following log will be generated: “WARNING,media timeout (notify)”
Default value is 0 (disabled)

rtpkeepaliveival

(number)
RTP stream keep-alive packet send interval in milliseconds.
This is useful if your server has an RTP timeout setting to prevent disconnects when MVoIPSDK is hold or muted.
Default value is 0. (You might set it to 25000 for example)

sendrtponmuted

(boolean)
Send rtp even if muted (zeroed packets)
Set to true only if your server is malfunctioning when no RTP is received.
Default value is false.

discmode

(number)
For call disconnect compatibility improvements. Some VoIP devices might have bugs with CANCEL forking, so it is better to always send a BYE after the CANCEL message on call disconnect. In this case set the discmode parameter to 3.
1: quick
2: conform the RFC
3: send BYE after CANCEL when needed
4: double: always repeat the CANCEL and the BYE messages
Default value is 2.

waitforunregister

(number)
Maximum time in milliseconds to wait for unregistration when the API_Unregister is called or MVoIPSDK is closed.
If set to 0 that an unregister message is sent (REGISTER with Expires set to 0) but MVoIPSDK is not waiting for the response, which means that it will not repeat the un-register in case if the UDP packet was lost.
Default value is 2000.

md5

(string)
Instead of using the password parameter you can pass an MD5 checksum for better protection: MD5(username:realm:password)
(The parameters are separated with the ‘:’ character)
The realm is usually your server domain name or IP address (otherwise it is set on your server)
If you are not sure, you can find out the realm in the “Authenticate” headers sent by your server with the” 401 Unauthorized” messages. Example:
WWW-Authenticate: Digest realm="YOURREALM", nonce="xxx", stale=FALSE, algorithm=MD5
Default is empty.

realm

(string)
Set if your server realm (SIP domain) is not the same with the “serveraddress” parameter.
If the “md5” parameter was set, then this must match with the realm used to calculate the md5 checksum.
Default is empty, which means that the “serveraddress” will be used.

encrypted

(boolean)
Specify if the transport will be encrypted (both media and the signaling)
Compatible only with Mizu VoIP servers.
Automatically turned on when using http tunneling.
Default is false.

authtype

(number)
Some server doesn’t allow “web” or “proxy” authentication.
0=normal
1=only proxy auth
2=only simple auth

sipusername

(string)
Specify default SIP username. Otherwise the “username” parameter will be used for both the username and the authentication name.

If this is not specified, then the “username” will be used for the From field and also for the authentication.
If both username and sipusername is set then
-the username will be used in the From and Contact fields (CLI/caller-id)
-the sipusername will be used for authentication only

Default is empty.

displayname

(string)
Specify default display name used in “from” and “contact” headers.
Default is empty (the “username” field will be displayed for the peers)

pwdencrypted

(number)
Specify if you will supply encrypted passwords via parameters or via the api.
0=no (default)
1=xor
2=des+base64
3=xor+base64 (this is the preferred method; easiest but still secure enough)
4= base64
This method is deprecated from version 3.4. All parameters can be passed encrypted now by just prefixing them with the “encrypted__X__” string where X means the id of the encryption method used.

From version 4.8 there is no need to specify this parameter anymore. Just prefix any parameter with encrypted_X as described in the “Parameter security” section

voicerecording

(number)
0=no (default)
1=local (in the user home directory)
2=remote ftp
3=both

voicerecfilename

(number)
The format of the recorded filenames.
0=date-time + peer name (default)
1=date-time + sip call-id
2=sip call-id
3=date-time + username
4=date-time + username + peer name

The date-time will be formatted in the following way: yyyyMMddhhmmss

Note: You can also use the "voicerecfilenameprefix" parameter to add a prefix for the file name.

voicerecftp_addr

(string)
FTP location for the recorded voice files if the "voicerecording" parameter is set to 2 or 3.
Format: ftp://USER:PASS@HOST:PORT/PATH/TO/THEFILE
Example: ftp://user01:pass1234@ftp.foo.com/FILENAME

The FILENAME part of the string will be replaced with the file name according to the "voicerecfilename" parameter.

voicerecformat

(number)
Recorded file compression.
0: PCM wave stereo files with separate channels for in/our (default)
1: raw gsm. 2 files will be generated for each call. One for the recorder file and another for the playback. These files can be played with players supporting gsm codecs for example quicktime (which works also as a browser plugin) or a winamp plugin is downloadable from here.
2: ogg/vorbis (optional, on request; module not included by default)

voicerecordingbuff

(number)
The maximum recorded file length.
-1: dynamic, no limit
1: max around 1 minute
2: max around 2 minute
…

Default is -1.

syncvoicerec

(number)
How to synchronize the recording/playback side:
-1: Auto
0: No (don't synchronize)
1: Yes (fill with noise the other channel)
2: Yes (wait for both side)
Default: 2

ftp_addr

(string)
FTP location for general storage (for example for settings, contactlists)
Format: ftp://USER:PASS@HOST:PORT/PATH/TO/THEFILE
Example: ftp://user01:pass1234@ftp.foo.com/FILENAME

The FILENAME part of the string will be replaced with the actual file name.

http_addr

(string)
HTTP location for general storage (for example for settings, contactlists)

Format: http://www.yourdomain.com/storage/
(this is just an example URL format. This URL will not work. You need to change this to your own web address)

autocfgsave

(number)
Configurations and statistics are stored in a local file to be reused in next sessions.
This is not critical and the MVoIPSDK will work just fine if this file is lost or deleted by the user.
Sometime is useful to not allow configuration/settings storage on the user device.
The autocfgsave option can be set to the following values:

- -2: disable all file write forced
- -1: disable file write
- 0: disable config storage
- 1: save only
- 2: load only
- 3: save and load

Default is 3.

canlogtofile

(boolean)
With loglevel set to 2 or more, the logs are written also to a local file. To disable the log files, set the canlogtofile parameter to “false” (or set the “loglevel” to 1).
Default is true.

resetsettings

(boolean)
Set to true to clear all previously stored or cached settings.
Default is false.

signalingport

(number)
Specify local SIP signaling port to use.
Default is 0 (a stable port which is selected randomly at the first usage)

Note: this is not the port of your server where the messages should be sent. This is the local port for MVoIPSDK.

rtpport

(number)
Specify local RTP port base.
Default is 0 (random between 20000 and 21000)

Note: If not specified, then VoIP SDK will choose a random port between 20001 and 21000 which is then remembered at the first successful call and reused next time (stable rtp port).

localip

(String)
Specify local IP address to be used.
This should be used only on devices with multiple ethernet interface to force the specified IP.
Default is empty (autodetect)

jittersize

(number)
Although the jitter size is calculated dynamically, you can modify its behavior with this setting.
0=no jitter,1=extra small,2=small,3=normal,4=big,5=extra big,6=max
Default is 3

maxjitterpackets

(number)

You can limit the jitter buffer size with this setting.
With the jittersize left as default (3) the maximum buffered packet count is limited to 8, so you might set this parameter to a lower value.
One packet means a received udp packet which might contain one or more audio frame.
For example when using G.729 the typical media stream are with 2 frames/packet. Each frame is 10 msec length.
A jitter limitation of 5 would mean maximum 100 msec to be cached. (while the default setting would allow 8 packet which means 160 msec)
Default value is 99 (no limitation)

increasepriority

(boolean)
This will increase the priority for the whole thread-group which might help on slow CPU's or when other applications are generating high CPU load.
Note: the priority of the threads handling media are increased regardless of this setting.
Default is false.

aqtest

(int)
Audio quality test.
Set to 1 for server/voice quality tests. More details in the FAQ.

loglevel

(number)
Tracing level. Values from 0 to 6.
If you set it to more than 3, then a log window will appear and also will write the logs to a file (if file write permissions are enabled on the client side).
With level 0, the VoIP SDK will not even display important even notifications for the user. Don't use this level if possible.
Loglevel 4 means a full log including SIP signaling. Loglevel 5 or 6 should be avoided (this can slow down MVoIPSDK)
Increased log levels has big impact on performance and usability. Use it only for short tests.
Text logs are sent to the following outputs:

-status display (only level 1 –these are the most important events that needs to be displayed also for the user)
-log window (if loglevel is higher than 3 then a log window will appear automatically. Copy the logs with Ctrl+A,Ctrl+C,Ctrl+V)
-file if loglevel is higher than 3 (*log.dat in the native directory)

Default is 1.

logtoconsole

(boolean)
Whether to send tracing to the console.
Default is false.

capabilityrequest

(boolean)
If set to true then will send a capability request (OPTIONS) message to the SIP server on startup. The serveraddress parameter must be set correctly for this to work. This method is useful to release the security restrictions when using the VoIP SDK with the API and also to open the NAT devices.
Default value is false.

natkeepalive

(boolean)
If set to true then will send a short message (\r\n) to the SIP server on startup. The serveraddress parameter must be set correctly for this to work. This method is useful to release the security restrictions and also to open the NAT devices.
Deprecated since v.3.8.2
Default value is false.

keepaliveival

(number)
NAT keep-alive interval in milliseconds which is usually sent from register endpoints.
Default value is 28000. (28 seconds)

recaudiobuffers

(number)
Number of buffers used for audio recording.

Default is 7.

recaudiomode

(number)
Audio recording mode. 0 means default; 1 means event based; 2 means device poll.
Default is 0.

useencryption

(boolean)
Set to true for encrypted communication (both media and signaling)
Works only with mizu servers.

httpsessiontimeout

(number)
Maximum session time in minutes.
You must call the API_HTTPKeepAlive() periodically (for example in every 1 minute) to avoid the timer expiry.
Default value is 3 minute.

jsscriptevent

(number)
Defines the level of notifications:
0=no notifications,1=status and cdr,2=important events,3= all logs including SIP signaling messages (depending also on loglevel).
Default is 2.

jsscripstats

(number)
Set to a value in seconds if you wish to receive extended periodic statistics for each line (STATUS notifications).
Default is 0 (no periodic statistics)

jsscriptpoll

(number)
Instead of auto-receiving notifications, you can also use API_Poll to ask for the events.
-1=auto/on demand (will turn to 2 on first API_Poll)
0=don't use polling
1=use polling
2=use polling or webhonetojs
3=use polling only
Default is -1

language

(string)

The following languages are built-in:
-en: english (default)
-ru: russian
-hu: hungarian
-ro: romanian
-de: deutsch
-it: italian
-es: spanish
-tr: turkish
-pr: portugheze
-jp: japanese
-fr: french

Both the 2 char and the full name variant can be used. For example: language = "it"

If you are using a html skin, then you will also have to translate the strings in your html.
If you wish the status messages to be also translated, set the “translatemode” parameter to 0.
Other languages can be added on your request or just translate your user interface.

In some countries you might also have to set the **locale** parameter to one of the following values: CANADA, CHINA, CHINESE, ENGLISH, FRANCE, GERMAN, GERMANY, ITALIAN, ITALY, JAPAN, JAPANESE, KOREA, KOREAN, ROOT (root locale), TAIWAN, UK, US. The default value is ENGLISH.

charset

(string)

Set the character decoding for SIP signaling.
Default is empty (will load the local system default).
More details:
http://docs.oracle.com/javase/7/docs/api/java/nio/charset/Charset.html
http://www.iana.org/assignments/character-sets/character-sets.xhtml

wpapilistenport

(number)
Internal listen port for TCP/HTTP.
Default is: 18522
Do not change.

wpapiudplistenport

(number)
Internal listen port for UDP.
Default is: 18522

wpapiconnectport

(number)
Internal messages will be sent to this port if UDP API is used.
Default is: 19521
This will be automatically set to the port from where the webphone received API requests.

Note: these are the internal ports used by the SDK. The default external port is 8952 so you always have to connect to 127.0.0.1: 18620 from your application.

FAQ

How to get my own VoIP SDK?

1. Have a look at the description at: https://www.mizu-voip.com/Software/Softphones/WindowsSIPSDK.aspx
2. Download and try from http://www.mizu-voip.com/Portals/0/Files/MVoIPSDK.zip
3. Contact Mizutech at info@mizu-voip.com with the following details
   -your VoIP server(s) address (ip or domain name). This will be hardcoded in your release; otherwise anybody could just download it from your and use as it owns)
   -your company details for the invoice (if you are representing a company)
4. Mizutech support will send your own MVoIPSDK build within one workday on your payment.
   The payment can be made from the pricing grid or other options (paypal, credit card, wire transfer) can be found here: http://www.mizu-voip.com/Company/Payments.aspx

What about support?

All licenses include also a support plan in the cost.
The support is done mostly by email. For “gold license” we offer 24/7 phone emergency support.
Maintenance upgrades are also free as included with your license plan.
Email to info@mizu-voip.com with any issue you might have.

Guaranteed supports hours depend on the purchased license plan and are included in the price.
Once your initial 1-4 years support period expires, it can be increased by 2 years for around $600 (This is optional. There is no need for any support plan to operate your MVoIPSDK).

What I will receive once I have made the payment for VoIP SDK?

You will receive the followings:

- The installer for the MVoIPSDK itself
- latest documentations
- invoice (on request or if you haven't received it before the payment)
- support on your request according to the license plan

Can Mizutech do custom development if required?

Yes, please contact us at info@mizu-voip.com. Please contact us only with MVoIPSDK related or VoIP specific projects.

Is it working with any VoIP servers?

Yes. The VoIP SKD for Windows uses the SIP protocol standard to communicate with VoIP servers and sofswitches. Since most of the VoIP servers are based on the SIP protocol today, MVoIPSDK should work without any issue.
If you have any incompatibility problem, please contact info@mizu-voip.com with a problem description and a detailed log (loglevel set to 4). For more tests please send us your VoIP server address with 3 test accounts.

Is MVoIPSDK using any Mizutech service or will contact Mizutech servers?

The VoIP SDK will connect to your voip server directly. No any intermediary app server is involved.
MVoIPSDK will not "call to home" and will not send any sensitive information to Mizutech servers.
MVoIPSDK might contact Mizutech servers for the following reasons:
-demo versions might contact Mizutech licensing servers time to time. This is completely removed in final builds (after your order)
- MVoIPSDK will use a random stun server by default hosted by Mizutech. This "fast stun" protocol is usually not required for normal functionality so you might just disable it (set the "use_fast_stun" parameter to 0) or set the "stunserver" parameter to your stun server. (However these can improve the connectivity if your VoIP server is not NAT friendly so better to leave it as is. Mizu services (non-) availability can't alter the usability of your product)
-In some circumstances MVoIPSDK might try to load some resources from Mizutech web servers if not found in your deployed package. For example the ring.wav or the mediaench dll's. These can be disabled too.
-In some circumstances the MVoIPSDK might download and install the Java SDK from the Mizutech website (if not already installed)

Is there any way to get the source code?

The VoIP SDK is a close-source application. The source code is available only for internal usage and for a higher price.

How can I make a call?

- Make sure that the "serveraddress" parameter is set correctly (otherwise you will be able to make calls only to direct SIP URI).
- Optionally: Register to the server. This can be done automatically if the "username" and "password" parameters are preset. Alternatively you can register from app (API_Register) or just let the user to fill in the username/password fields and click on the "Connect" button. If MVoIPSDK is registered, then it can already accept incoming calls (it will do it automatically, or you can handle incoming calls from your app, or you can entirely disable incoming calls)
- Now you can make outgoing calls in the following ways:
    - Automatically with MVoIPSDK start. For this you will have to preset the username/password/autocall and callto parameter. Then MVoIPSDK will immediately launch the outgoing call when starts (usually with your page load)
    - Just let the users to enter a called number and hit the "Call" button
    - or just call the API_Call function (from user button click or from your business logic)

Read through the parameters to find out more call divert settings, such as auto-answer or forward.

Multiple account registration

The VoIP SDK is capable to register multiple accounts at the same time. This can be useful to be able to receive calls from multiple accounts or multiple servers. The SIP accounts can be configured by parameters in the following way:

serveraddress, username, password, registerinterval: primary account
serveraddress2, username2, password2, registerinterval2: second account
serveraddress3, username3, password3, registerinterval3: third account
...

serveraddressN, usernameN, password, registerintervalN: N account

Or via the API_RegisterEx(String accounts) API call where the accounts are passed as string in the following format:

server,usr,pwd,ival;server2,usr2,pwd2, ival;

(accounts separated by ; and parameters separated by ,)

Or via the "extraregisteraccounts" parameter which have to be set like this:

server,usr,pwd,ival;server2,usr2,pwd2, ival;

Notes:

-Up to 99 secondary accounts can be used this way

-The ival/registerinterval parameter is optional (default is 3600 which means one hour)

-All other parameters are applied globally for all account (there is no per account profile). The "proxyaddress", if set, will be applied for primary account only

-STATUS is not reported from secondary accounts

ERROR and WARNING messages in the log

If you set MVoIPSDK loglevel higher than 1 than you will receive messages that are useful only for debug. A lot of ERROR and WARNING message cannot be considered as a fault. Some of them will appear also under normal circumstances and you should not take special attention for these messages. If there are any issue affecting the normal usage, please send the detailed logs to Mizutech support (info@mizu-voip.com) in a text file attachment.

TCP client cannot connect to API

-Make sure that the service is running: check the MVoIPSDKService NT service is running (search for "Services" in the Start menu to launch the Services console).

-Check the logs to make sure that the service was capable to listen on 18620. (Check if other some app is somehow using the same port. This can be seen from the Task Manager -> Resource monitor or by using the netstat command line)

-Disable your firewall if any (some exotic firewalls might block also localhost communications)

No NOTIFICATION messages are revived

This usually means that the internal SIP engine was not started properly.

-Check the logs to see if the engine was capable to launch the internal sip stack at \MVoIPSDK\content\native\ wphoneapp.jar

-Check the internal SIP stack logs at \MVoIPSDK\content\native*log.dat

-Reinstall the MVoIPSDK and restart your PC

Failed outgoing calls

By default only the PCMU,PCMA, G.729 and the speex ultra-wideband codec's are offered on call setup which might not be enabled on your server or peer UA. You can enable all other codec's (PCMA, GSM, speex narrowband, iLBC and G.729) with the use_xxx parameters set to 2 or 3 (where xxx is the name of the codec: use_pcma=2, usgsm=2, use_speex=2,use_g729=2,use_ilbc=2).

Some servers has problems with codec negotiation (requiring re-invite which is not support by some devices). In these situations you might disable all codec's and enable only one codec which is supported by your server (try to use G.729 if possible. Otherwise PCMU or PCMA is should be supported by all servers)

Calls are disconnecting

If the calls are disconnecting after a few second, then try to set the "invrecordroute" parameter to "true" and the "setfinalcodec" to 0.

If the calls are disconnecting at around 100 second, then most probably you are using the demo version which has a 100 second call limit.

If the calls are disconnecting at around 3 minutes check the httpsessiontimeout parameter (you need to call the API_HTTPKeepAlive function periodically from a timer)

Not working with Avaya systems

Select UDP (and not TCP or TLS) for the link protocol on your Avaya configuration.

Known limitations

- Some Linux distributions doesn't have full duplex audio driver. In these circumstances only one audio stream can be used (MVoIPSDK should be able to handle this automatically with default settings)
- Some OS/driver/hardware configurations might not support wideband audio (in these circumstances MVoIPSDK will automatically use narrowband only)

- No technical support for video related functionality (we provide it “as is”)
- The full STUN specification is not implemented due to file size considerations (a light STUN version is used with a built-in list of available servers if needed)
- Processing of In-Band DTMF is not supported (INFO or RFC2833 are both supported)
- The VoIP SDK is targeting PC platforms only. For smartphones you should check our mobile softphone’s.

Why I see RTP warning in my server log

MVoIPSDK will send a few (maximum 10) short UDP packets (\r\n) to open the media path (also the NAT if any).
For this reason you might see the following or similar Asterisk log entries: “WARNING[8860]: res_rtp_asterisk.c:2019 ast_rtp_read: RTP Read too short” or “Unknown RTP Version 1”.
These packets are simply dropped by Asterisk which is the expected behavior. This is not a MVoIPSDK or Asterisk error and will not have any negative impact for the calls. You can safely skip this issue.

You might turn this off by the “natopenpackets” parameter (set to 0). You might also set the “keepaliveival” to 0 and modify the “keepaliveival” (all these might have an impact on MVoIPSDK NAT traversal capability)

RTP statistics

For RTP statistics increase the log level to at least 3 and then after each call longer than 7 seconds you should see the following line in the log:
EVENT, rtp stat: sent X rec X loss X X%.
If you set the “loglevel” parameter to at least “5” than the important rtp and media related events are also stored in the logs.

NAT settings

In the SIP protocol the client endpoints have to send their (correct) address in the SIP signaling, however in many situations the client is not able to detect it’s correct public IP (or even the correct private local IP). This is a common problem in the SIP protocol which occurs with clients behind NAT devices (behind routers). The clients have to set its IP address in the following SIP headers: contact, via, SDP connect (used for RTP media). A well written VoIP server should be able to easily handle this situation, but a lot of widely used VoIP server fails in correct NAT detection. RTP routing or offload should be also determined based in this factor (servers should be always route the media between 2 nat-ed endpoint and when at least one endpoint is on public IP than the server should offload the media routing). This is just a short description. The actual implementation might be more complicated.

You may have to change MVoIPSDK configuration according to your SIP server if you have any problems with devices behind NAT (router, firewall).
If your server has NAT support then set the use_fast_stun and use_rport parameters to 0 and you should not have any problem with the signaling and media for MVoIPSDK behind NAT. If your server doesn’t have NAT support then you should set these settings to 2. In this case MVoIPSDK will always try to discover its external network address.

Example configurations:
If your server can work only with public IP sent in the signaling:
-use_rport 2 or 3
-use_fast_stun: 1 or 2

If your server can work fine with private IP’s in signaling (but not when a wrong public IP is sent in signaling):
-use_rport 9
-use_ fast_stun: 0
-optionally you can also set the “udpconnect” parameter to 1

Asterisk is well known about its bad default NAT handling. Instead of detecting the client capabilities automatically it relies on pre-configurations. You should set the "nat" option to "yes" for all peers.
More details:
http://www.voip-info.org/wiki/view/NAT+and+VOIP
http://www.voip-info.org/wiki/view/Asterisk+sip+nat
http://www.asteriskguru.com/tutorials/sip_nat_oneway_or_no_audio_asterisk.html

Server failover/fallback

Use the following settings if you have 2 voip servers:

- serveraddressfirst: the IP or domain name of the first server to try
- serveraddress: the IP or domain name of the next server
- autotransportdetect: true
- enablefallback: true

In this way MVoIPSDK will always send a register to the first server first and on no answer will use the second server (the “first” server is the “serveraddressfirst” at the beginning, but it can change to “serveraddress” on subsequent failures to speed up the initialization time)

Alternatively you can also use SRV records to implement failover or load balancing.

I have call quality issues

Call quality is influenced primarily by the followings:

- Codec used to carry the media
- Network conditions (check also your upload packet loss/delay/jitter)
- Hardware: enough CPU power and quality microphone/speaker (try a headset, try on another device)
- (Missing) AEC and denoise

If you have call quality issues then the followings should be verified:

- whether you have good call quality using a third party softphone from the same location (try X-Lite for example). If not, than the problem should be with your server, termination gateway or bandwidth issues.
- make sure that the CPU load is not near 100% when you are doing the tests
- make sure that you have enough bandwidth/QoS for the codec that you are using
- change the codec (disable/enabled codec’s with the use_xxx parameters where xxx have to be replaced with the codec name)
- deploy the mediaench module (for AEC and denoise). (Or disable it if it is already deployed and you have bad call quality)
- MVoIPSDK logs (check audio and RTP related log entries)
- wireshark log (check missing or duplicated packets)

I have one way audio

1. Review your server NAT related settings
2. Set the “setfinalcodec” parameter to 0 (especially if you are using Asterisk or OpenSIPS)
3. Set use_fast_stun, use_fast_ice and use_rport to 0 (especially if you are using SIP aware routers). If these don’t help, set them to 2.
4. If you are using MizuVoIP server, set the RTP routing to “always” for the user(s)
5. Make sure that you have enabled all codec’s
6. Make a test call with only one codec enabled (this will solve codec negotiation issues if any)
7. Try the changes from the next section (Audio device cannot be opened)
8. If you still have one way audio, please make a test with any other softphone from the same PC. If that works, then contact our support with a detailed log (set the” loglevel” parameter to 5 for this)

Audio device cannot be opened

If you can’t hear audio, and you can see audio related errors in the logs (with the loglevel parameter set to 5), then make sure that your system has a suitable audio device capable for full duplex playback and recording with the following format:
PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian

If you have multiple sound drivers then make sure that the system default is workable or set the device explicitly from MVoIPSDK.

To make sure that it is a local PC related issue, please try MVoIPSDK also from some other PC.

You might also try to disable the wideband codec’s (set the use_speexwb and use_speexuwb parameters to 0 or 1).

Another source for this problem can be if your sound device doesn’t support full duplex audio (some wrong Linux drivers has this problem). In this case you might try to disable the ringtone (set the “playring” parameter to 0 and check if this will solve the problem).

If these doesn’t help, you might set the “cancloseaudioline” parameter to 3 and/or the "singleaudiostream” to 5.

Error creating UDP sockets

In some specific circumstances MVoIPSDK might not be able to create more UDP sockets.
In this case you will see one of the followings in the log:

- ERROR, catch on udp bind Unrecognized Windows Sockets error: 0: Cannot bind
- ERROR, catch on udp bind maximum number of DatagramSockets reached

Verify your firewall (especially if you are using some third party firewall) and virus scanner.

No ringback tone

Copy the ring.wav to your app folder.
(You can also use your custom ringtone: any wave file encoded as standard 8 kHz 16 bit mono PCM files 128 kbits - 15 kb/sec)

If this doesn't help:

Depending on your server configuration, you might not have ringback tone or early media on call connect.
There are a few parameters that can be used in this situation:

- set the "changesptoring" parameter to 3
- set the "natopenpackets" parameter to 10
- set the "earlymedia" parameter to 3
- change the use_fast_stun parameter (try with 0 or 2)

One of these should solve the problem.

The remote party hear itself back (echo)

To solve the aec issues you need to set the "aec" parameter to 2. (for better quality also set the "denoise" parameter to 1). The AEC should eliminate more than 90% of the echo with around 92% success rate. (This is if a speaker is used. If the user will use a headset than echo is generated only if the headset is broken).

Chat is not working

Make sure that your softswitch has support for IM and it is enabled. MVoIPSDK is using the MESSAGE protocol for this from the SIP SIMPLE protocol suite as described in RFC 3428.
Most Asterisk installations might not have support for this by default. You might use Kamailio for this purpose or any other softswitch (most of them has support for RFC 3428).

Subsequent chat messages are not sent reliably

Set the "separatechatdiag" parameter to 1.

MVoIPSDK doesn't receive incoming calls

To be able to receive calls, MVoIPSDK must be registered on your SIP server first (you can use the "register" parameter with supplied username and password). Once MVoIPSDK is registered, the server should be able to send incoming calls to it.

The other reason can be if your server doesn't handle NAT properly.
Please try to start MVoIPSDK with use_fast_stun parameter set to 0 and if still not works then try it with 2.

If the calls are still not coming, please send us a log from MVoIPSDK (set MVoIPSDK loglevel parameter to 5) and also from the caller (your server or remote SIP client)

What is the best codec?

There is no such thing as the "best codec". All commonly used codec's present in MVoIPSDK are well tested and suitable for IP calls.
This depends mainly on the circumstances.
Usually we recommend G.729 since this provides both good quality and good compression ratio.
If G.729 is not available in your license plan, than the other codecs are also fine (GSM, speex, iLBC)
Otherwise the G711 codec is the best quality narrowband codec. So if bandwidth is not an issue in your network, than you might prefer PCMU or PCMA (both have the same quality)
Between MVoIPSDK users (or other IP to IP calls) you should prefer wideband codec's (this is why you just always leave the speex wideband and ultra-wideband with the highest priority if you have calls between your VoIP users. These will be picked for IP to IP calls and simply omitted for IP to PSTN calls)
To calculate the bandwidth needed, you can use this tool. You might also check this blog entry: Codec misunderstandings

What is the default codec priority?

If you doesn't change the codec priorities with the parameters, than the default codec order will be the following (listed in priority order):

1. speex wideband (enabled low priority 2)
2. G.729 (enabled low priority 2)
3. PCMU (enabled low priority 2)
4. PCMA (disabled 1)
5. Opus all (disabled 1)
6. speex ultrawideband (disabled 1)
7. speex narrowband (disabled 1)
8. GSM (disabled 1)
9. iLBC (disabled -not activated 0)

This means that to prefer a codec you just have to add one single line for the parameter:

-use_myfaworitecodec=3

This will automatically enable and put your selected codec as the highest priority one.

If you set all codec with the same priority, then the real priority will be the following:

1. Speex ultrawideband (top priority)
2. Speex wideband
3. G729
4. G711
5. Gsm
6. Speex narrowband
7. Ilbc (lowest priority)

*Speex wideband and ultra-wideband can be automatically disabled if your sound card doesn't support the increased sample rate.
*The other endpoint usually will pick up the first codec, or MVoIPSDK will pick-up the first in this list from the list of codec's sent by the other peer
* G.729, GSM and speex narrowband and ultra-wideband codec's are disabled by default. Set use_g729,use_gsm, use_speex, use_speexuwb to 2 or 3 to enable them. (Make sure you have the proper G.729 licenses)

How to prefer one codec?

Set its priority to 3 and set the priority for all other codes to 2. In this case you preferred codec will be used whenever the other endpoint supports it and other codec's are used only if otherwise the call would fail.
For example the following parameters will set g.729 as the preferred codec and will enable also pcmu and gsm:

```
-use_g729=3
-use_pcmu=2
-use_pcma=1
-use_gsm=2
-use_speex=1
-use_speexwb =1
-usespeexuwb =1
-use_ilbc =0
```

How to force only one codec?

Enable only one codec by parameters and disable all others. In this case the call might fail if the other end doesn't support the selected codec.

For example the following parameters will force the softphone to use only pcmu:

```
-use_pcmu=3
-use_pcma=1
-use_g729=1
-use_gsm=1
-use_speex=1
-use_speexwb =1
-use_speexuwb =1
-use_ilbc =1
```

Caller ID display

For outgoing calls the Caller ID (CLI)/A number display is controlled by the server and the application at the peer side (be it a VoIP softphone or a pstn/mobile phone).
You can use the following parameters to influence the caller id display at the remote end:

-username
-authusername/sipusername
-displayname

Some VoIP server will suppress the CLI if you are calling to pstn and the number is not a valid DID number or MVoIPSDK account doesn't have a valid DID number assigned (You can buy DID numbers from various providers).
The CLI is usually suppressed if you set the caller name to "Anonymous" (hide CLI).

For incoming calls MVoIPSDK will use the caller username, name or display name to display the Caller ID. (SIP From and Contact fields).
You can also use headers such as preferred-identity to control the Caller ID display.

Transfer API usage

With unattended transfer the transfer will be executed immediately once you call the API_Transfer function.
With attended transfer (transfertype 5) you can use the following call-flow (supposing that you are working at A side and wish to transfer B to C):

1. A call B (outgoing) or B call to A (incoming)
   A speaking with B
2. Call API_Transfer(-1,C)
   The call between A and B will be put on hold by A
   A will call to C and connect (consultation call; if call fails, then the call between A and C will be un-hold automatically)
   A speaking with C
3. The actual call transfer will be initiated when A disconnect the call (API_Hangup)
   REFER message will be sent to B (which tells to B to call C. usually by automatically replacing the A-B call with A-C)
4. After transfer events:
   -If transfer fails (B can't call C) the call between A and B will be will be un-hold automatically (if server sends proper notifications)
   -If the transfer succeeds (B called C) the call between A and B will be will be disconnected automatically (if server sends proper notifications)
   -If the server doesn't support NOTIFY, then the call can be un-hold or disconnected from the API (API_Hold(-2,false), API_Hangup(-1))
5. B speaking with C at this point if the transfer was successful

How to get the audio stream?

The VoIP SDK is capable to stream the received/sent media to a custom UDP port (to your local application or other application which will process the audio data).
This is useful if you wish to make some processing on the audio streams such as speech to text or other analysis (for example real-time translation via the Azure or Google Cloud API). For example you can use this code if you are using the SDK with C#.
Just launch an UDP socket on any port and set this port number as the sendmediain_to and/or sendmediaout_to parameter for the SDK. Then you will receive the media streams (the audio packets) from the calls being made.

Parameters:
sendmedia_type

Specify the media stream format sent by the SDK to your app:
0: raw wave format (linear PCM) for narrowbad (8 kHz 16 bit mono PCM files at 128 kbits - 15 kb/sec) or 16 kHz for wideband.
1: for RTP format (RTP header + payload with the actual media codec)
2: convert sample rate to raw PCM 8kHz (useful if original stream is in 16kHz format such as Opus or Speex wideband, but you need 8kHz PCM)
3: convert sample rate to raw PCM 16kHz (useful if original stream is in narrowband 8kHz format but you need 16kHz PCM)
4: RTP data only, without RTP header (raw codec format)
Default is 0.

Note: It is recommended to set this to 0 or 1 and force a codec to match your needs (use OPUS or Speex if you need 16kHz PCM wideband or other codec if you need 8kHz PCM narrowband), however if you need raw audio in other sample rate then you might set it to 2 or 3 for sample rate conversion.

sendmediain_to

Specify your UDP port where you wish to receive remote audio (received from other peer for playback on local speaker)
Default is 0 which means no streaming.

sendmediaout_to

Specify your UDP port where you wish to receive local audio (recorded from microphone, which is sent to the other end)
Default is 0 which means no streaming.

sendmedia_line

Specify if UDP packets should have a line number header. Useful in case of you wish to handle multi lines in the same stream (sent to same UDP port).
0: no header (default)
1: add line header string. In this case the packets will begin with the line number, followed by comma, followed by audio binary data.
Example: 2,xxxxx
(You must extract the line number from each packet: get the string until the first comma and convert it to int. The bytes after the comma will be the audio data)
2: add line and SIP Call-ID string header. Example: 3,abc,xxxxx
3: add line header byte. In this case the first byte in the packet will be the line number (0-127)
4: add line header and VAD status byte. The VAD status (second byte) will be 0 if unknown, 1 if silence, 2 if speaking.

sendmedia_marks

Specify if you wish to receive BOF/EOF packets (udp packets containing 3 bytes at the begin and end of streams).
Default is 0. Set to 1 if you wish to have these packets.

Notes:
You might force G.711 codec for the VoIP SDK to simplify the audio formats conversion (disable all codec's except PCMU and PCMA)

This is about streaming to your own or to a third-party application.
If you need to stream an audio file to a remote SIP endpoint, just use the API_PlaySound.

Work with audio streams

If you have set audio streaming as it was discussed above, then you have the following possibilities to handle it:

1. **Third party software:**

Use some third-party software which is capable to playback or process the audio packets (such as for playback, transcription or monitoring)

2. **Cloud service:**

Use some third-party software which is capable to playback or process the audio packets (such as for playback, transcription or monitoring)
Stream the audio to any cloud service for further processing (for example Google or Azure speech to text API)
For the Google speech API you can find more details here. Contact us if you need some C# sample code *(ref num 8117459355)*.

3. **RTP stream:**

If you have a software which is capable to process RTP packets then just set the sendmedia_type to 1 so the SDK will emit RTP packets which can be processed as is by such software.

4. **PCM stream:**

Use the default PCM stream if you don't have any ready to use software and you have to write it yourself.
The packet emitted by the SIP media stack can be processed as-is by any audio player module.

5. **Wave files:**

If you don't need real-time audio, then just use the voicerecording and related parameters to obtain voice files in wave or other formats at the end of each call.

6. **Barge-In:**

If you just wish to listen into conversation, then you can use the SIP SDK itself for the job. Its barge-in feature allows you to bare into any call and create a hidden conference endpoint, so you can hear the conversation. This is often used in callcenters by supervisors. Contact our support if you need this module.

7. **SIP media streaming:**

Use the API_PlaySound function if you need to stream an audio file to a remote SIP endpoint.

How to generate and send logs from MVoIPSDK

If you have any problem with the VoIP SDK, Mizutech support most probably will ask you to send a detailed description and logs about the problem.
For this the "loglevel" parameter must be set to 5 and then you can find the logs in the app folder which is usually located at:
C:\Program Files (x86)\ MVoIPSDKService*log.dat
C:\Program Files (x86)\ MVoIPSDKService\content\native\mwphonedata*log.dat

If the logs can't be found in the app directory then check the data directory:

C:\Users\secondaryacc\AppData\Roaming\MVoIPSDKService

If you are running your own purchased build then you might need to search for “YourBrand_Service” instead of “MVoIPSDKService”.
Internal SIP engine logs can be found at \content\native\mwphonedata*log.dat.

Once you reproduced the problem, send the content of the logs to info@mizu-voip.com

Note:

- If the problem is call related, then make sure to have only one call in the log. That one in which the problem was reproduced.
- In addition to the log file, Mizutech support might ask one or two SIP account valid on your server to be able to reproduce the problem

Resources

VoIP SDK homepage : https://www.mizu-voip.com/Software/Softphones/WindowsSIPSDK.aspx
Download (demo package): https://www.mizu-voip.com/Portals/0/Files/MVoIPSDK.zip
For help, contact info@mizu-voip.com