2020

JVoIP -Java VoIP SDK

A full featured, flexible SIP client in a single JAR file

The Mizu Java VoIP SDK (JVoIP) is a lightweight standards based VoIP phone that can be used as a library or as a standalone application/applet. Based on the industry standard SIP, RTP and related protocols, it is compatible with all common VoIP devices, servers and softphones, providing easy integration capabilities with any application.

-
Mizutech
9/14/2020

Contents

About

The Mizu Java VoIP SDK (JVoIP) is a SIP client implemented as a platform independent java library. Since it is based on the open standard Session Initiation Protocol, it can inter-operate with any other SIP-based device (servers and clients).

The VoIP SDK can be used in many ways:

- as a library added to your java project (use the VoIP client API)
- as a command line Java VoIP client (with stdin input / stdout output)
- as a standalone desktop application (it has a built-in minimalistic GUI to ease such kind of usage)
- as an applet embedded to a web page

With the Java VoIP SDK you have an easy to use full featured SIP/media stack in a single jar file, easy to integrate or embedded in your desktop, server or web application. For example it can be integrated with callcenter software or embedded in VoIP devices such as PBX or gateways so users will have a fully functional VoIP softphone without the need to download any other third-party software. You can also 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).

See all related resources here.

Requirements

Any OS with Java SE support (Linux, Windows, MAC, Others)
JVM: Works with most JVM's including Oracle and OpenJDK. Download.
Minimum Java version: J2SE 5.0+ (all java versions since 2004)
Maximum Java version: no limitations, the library doesn't use special modules or API's which might be deprecated in further versions. Works well with Java 16 and expected to work well with all further Java JVM/JRE/JDK releases.
Programming language (when used as a library): any JVM based language such as Java, Clojure, Kotlin, Groovy or Jython
Development: any tools, any OS and any IDE can be used (the library is a single jar file which can be easily added to your project regardless your environment)
Audio device: Headset or microphone/speaker for audio (will work also without audio device for streaming or voice recording)
CPU: minimum 350 MHz P3 or similar (runs well also on embedded devices such as Raspberry Pi)
RAM: minimum 10 MB (above the JVM basic requirements)
Disk space: 2 MB (additional data such as call recording or detailed logs might take more space)
A SIP account (At any VoIP service provider or your own IP-PBX/Softswitch/SIP sever. Can be also used without registration for peer to peer SIP calls)

Note: If you are looking for a SIP stack for your Android project, then you should use the AJVoIP SDK instead of this library (AJVoIP has the same API as this JVoIP SDK, but targets Android instead of Java SE)

Usage

The SDK can be used in many ways: as a standalone application, embedded into your project or integrated with your website.
The settings can be specified from: API, command line, config file, URL or sent via SIP signaling.
It can be used as a standalone app or integrated into other app with or without using the API.
The API can be accessed from: Java and any other JVM language, JavaScript, UDP, TCP or HTTP (clear text, URL, JSON, XML)
The API can be directly used from Java or other JVM based application such as Kotlin, or you can use the API via UDP/TCP or HTTP from any development environment and programming language such as C, C++, C# or Delphi.

Download: Java SIP SDK (this is the demo version)

Note: Optionally (for better voice quality) you might also copy the mediaench files to your app folder (near your java or jar files). This contains some platform dependent native binaries to optimize audio processing and the sip client will select the correct one to be used automatically if found.

Using from the console/command line

This is a simplest way to test as a simple SIP client and it is often used for various automation tasks:
Example:
java –jar JVoIP.jar serveraddress=VOIP_SERVER_IP_OR_DOMAIN username=USERNAME password=PASSWORD callto=DESTINATION autocall=false loglevel=1
Here is a working example: JVoIP.jar serveraddress=voip.mizu-voip.com username=jvoiptest password=jvoiptestpwd callto=testivr3 autocall=false loglevel=1
Replace the values in uppercase accordingly and make sure to have Java installed.

To disable the GUI (use as a console application) set the iscommandline setting to true. Otherwise you can launch the GUI also from command line with the API_StartGUI function (use API_StartGUI() instead of API_Start() if you need this functionality).
By default all parameters are set to its optimal/most common values, however if you wish to further customize or you need any specific functionality then you might use any other command line options as documented below in the parameters chapter.

Note:

- The demo version doesn't work in headless mode like Linux with no X-Window installed. Contact us if you need a headless version!
- Some systems (Windows) might accept the command line also without the java –jar prefix, other systems (Linux) might require the java –jar JVoIP.jar [parameters] format.

Using as a standalone application

This means starting the app with its built-in user interface.

Create a configuration file near the JVoIP.jar with the file name: wpcfg.ini. The content should be like this:
serveraddress=VOIP_SERVER_IP_OR_DOMAIN
username=USERNAME
password=PASSWORD
callto=DESTINATION
loglevel=1
…other config options if needed
Replace the values in uppercase accordingly and make sure to have Java installed.
Then just double click on JVoIP.jar to start (or launch it from command line as java –jar JVoIP.jar)

JVoIP has a minimalistic built-in softphone user interface for your convenience (which can be modified by the appearance parameters), however you can easily create your own user interface and design when used as a library or embedded in a webpage as discussed below.

Using as a library

You can use JVoIP as an SDK if you are a Java developer to add VoIP functionality in any JVM (java) application by including this SIP library into your project.
This is the most powerful use-case to fully exploit JVoIP capabilities.
You just need to add JVoIP to your project and call its public API_XXX functions. (See the "API" chapter below for more details").

In short, the interaction with the library is done by calling it's API functions, handling the function return values (success-true/failed-false/other answers) and handling the notifications received from the library (for example the STATUS messages about the SIP stack state machine).

Example use-cases:

- Implement your custom Java SIP client
- Implement a full featured Java Softphone
- Add VoIP capabilities to any Java application (or any JVM based apps such as Clojure, Scala, Kotlin, Groovy, JRuby or Jython)
- Add SIP call capabilities to any application running over a Java virtual machine

Steps:

1. Add JVoIP.jar lib to your project
2. Instantiate a webphone object (webphone wobj = new webphone();)
3. Call the API_SetParameter() to pass any settings (parameters)
4. Call the API_Start() to start the SIP stack
5. Call any other functions (such as API_Call, API_SendChat and others as described in the API)
6. Poll for the notifications with the API_GetNotifications() or API_GetNotificationsSync() function call, preferably from a separate thread. Parse and process the notifications from the sip stack after your needs as described in the "Notifications" section (optional. For simple applications this might not be needed)

Note: all API calls are thread safe and will not throw exceptions (on exception they will send an "ERROR" notifications and/or return false/-1 depending on the context).

Example code (exception and error handling removed for simplicity):

```
----------------------------------------
File: YourCodeSipCall.java
----------------------------------------
package yourpackagename;
import webphone.*;  //add JVoIP.jar to your project for this. both the package and the main class are named "webphone"

//create webphone class object instance:
webphone wobj = new webphone();

//start your message listener:
SIPNotifications sipnotifications = new SIPNotifications(wobj);
sipnotifications.Start();


//set parameters (replace uppercase words):
wobj.API_SetParameter("serveraddress", "VOIP_SERVER_IP_OR_DOMAIN");
wobj.API_SetParameter("username", "SIP_USERNAME");
wobj.API_SetParameter("password", "SIP_PASSWORD");
wobj.API_SetParameter("loglevel", "5");  //you might set to 1 for production
```

```
//you might set other parameters here. most parameters can be also set or changed later at runtime
//start the sipstack:
wobj.API_Start();
//register to your SIP server (optional):
//wobj.API_Register();

//make a call to a user/extension/phone number/SIP URI (note: 1-2 seconds might be needed between API_Start/API_Register and API_Call for the sipstack to initialize)
wobj.API_Call(-1, "DESTINATION");
//the following lines should be used from another function
//during the call you might call call divert functions by your app logic or on user interaction, for example API_Hold()

//call hangup to end the call (possibly triggered by a “Hangup” button pressed by the user):
wobj.API_Hangup(-1);
//stop JVoIP when you don’t need it anymore
wobj.API_Unregister();
sipnotifications.Stop();
```

File: SIPNotifications.java

```
package JVoIPTestPackage; //you might change this after your package name
import webphone.webphone;


public class SIPNotifications extends Thread
{
  boolean terminated = false;
  webphone webphoneobj = null;

  //constructor
  public SIPNotifications(webphone webphoneobj_in)
  {
     webphoneobj = webphoneobj_in;
  }

  //start the thread
  public boolean Start()
  {

     try{
        this.start();
        System.out.println("sip notifications started");
        return true;
     }catch(Exception e) {System.out.println("Exception at SIPNotifications Start: "+e.getMessage()+"\r\n"+e.getStackTrace()); }
     return false;
  }

 //stop the thread
  public void Stop()
  {
        terminated = true;
  }

  //blocking read in this thread
  public void run()
  {
     try{
        String sipnotifications = "";
        String[] notarray = null;

        while (!terminated)
        {
             //get the notifications from the SIP stack
             sipnotifications = webphoneobj.API_GetNotificationsSync();

             if (sipnotifications != null && sipnotifications.length() > 0)
             {
                //split by line
                System.out.println("\tREC FROM JVOIP: " + sipnotifications);
                notarray = sipnotifications.split("\r\n");

                if (notarray == null || notarray.length < 1)
                {
                  if(!terminated) Thread.sleep(1); //some error occured. sleep a bit just to be sure to avoid busy loop
                }
                else
                {
                 for (int i = 0; i < notarray.length; i++)
                 {
                    if (notarray[i] != null && notarray[i].length() > 0)
                    {
                       ProcessNotifications(notarray[i]);
                    }
                 }
                }
             }
```

```
            else
            {
              if(!terminated) Thread.sleep(1);  //some error occured. sleep a bit just to be sure to avoid busy loop
            }
        }

      }catch(Exception e)
      {
        if(!terminated) System.out.println("Exception at SIPNotifications run: "+e.getMessage()+"\r\n"+e.getStackTrace());
      }

    }


    //all messages from JVoIP will be routed to this function.
    //parse and process them after your needs regarding to your application logic

    public void ProcessNotifications(String msg)
    {
      try{
        //frame.jTextArea1.append(msg);

        //TODO: process notifications here (change your user interface or business logic depending on the sip stack state / call state by parsing the strings receiver here).
        //See the Notifications section in the documentation for the details. Example code can be found here.
      }catch(Exception e) { System.out.println("Exception at SIPNotifications ProcessNotifications: "+e.getMessage()+"\r\n"+e.getStackTrace()); }
    }
}
```

You can download a working example from here.

Socket/HTTP API

The Socket interface is useful if you wish to use JVoIP as a library from non-JVM environment such as .NET, PHP, C++ or others.

While from Java (or other JVM based language) you can call the API functions directly, in case if you are using other development environment then you must call the JVoIP API via a socket. In this case you will have to convert your function calls to a simple string and send it over a socket. You will receive the answers for your API requests and notifications over the same socket. The send/receive string formats are discussed below.

All the functions and notifications are the exact same like you would use JVoIP as an embedded library, the only difference is that the library will run in a separate process, you connect to it via UDP, TCP or HTTP and you have to convert/parse everything (API requests, API answers and notifications) as strings.

Configuration:
First of all, you will have to launch launch JVoIP as a separate process (for example ShellExecute on Windows, but for first you can just launch it manually from command line), then send commands to the JVoIP listening port by:

- UDP (configurable by the wpapiudplistenport parameter. 19422 by default)
- TCP (configurable by the wpapilistenport parameter. 0 by default which means disabled)
- HTTP (configurable by the same wpapilistenport parameter as it will auto detect raw TCP vs HTTP. 0 by default which means disabled)

Note:

- Only UDP is enabled by default. If you need TCP and/or HTTP set the wpapiudplistenport to 0 and the wpapilistenport to any value, for example 19422.
- Some very restrictive firewalls might block also localhost connections. If you have installed such a firewall on your PC, make sure to allow JVoIP/Java.
- If you start to use the API_GetNotifications or API_GetNotificationsSync function (using the API directly from Java) the JVoIP will stop sending out notifications via sockets

The following **formats** are accepted for requests and answers:

- clear text (so you can easily test it manually with telnet)
- HTTP (GET/POST/AJAX)
    - URL parameters (if used from a browser as an applet)
    - JSON
    - form post
    - XML
    - SOAP

Requests:
Example from browser (HTTP with URL parameters): http://127.0.0.1:19422/?function=API_Call&line=-1&peer=1234
Example from telnet (clear text TCP):

```
telnet 127.0.0.1 19422
?function=API_Call&line=-1&peer=1234
```

For the function parameters you can use the exact same name as in this documentation or you can use param1, param2 … paramN format.

From local applications we recommend UDP with clear text. For faster processing you can also split the commands between EOFCOMMAND/BOFCOMMAND and the parameters between EOFLINE/BOFLINE.
For example: BOFCOMMANDBOFLINEfunction=API_CallEOFLINEBOFLINEparam1=-1EOFLINEBOFLINEparam2=1234EOFLINEEOFCOMMAND

Answers:
Answers are sent in the following clear text format:
APIREQUEST:API_CallRPARAM1:1RPARAM2:1234APIRESULT:**RESULT**
The first part (until "APIRESULT:") might be useful if you are using the API asynchronously, so you will always know for which request did you received the answer. The important part is the RESULT string (after "APIRESULT:") and usually begins with OK or ERROR. For example: OK: initiated or ERROR: failed

Notifications:
If you are using simple UDP or TCP (not HTTP) then the NOTIFICATIONS are also sent automatically on the same socket. Otherwise you can use API_GetNotifications to receive the notifications.

From HTTP clients we recommend AJAX HTTP requests with URL parameters. Also you should set the polling parameter to 3 and use API_GetNotifications to receive the notifications (since in case of HTTP there is no any live socket stream where the notification might be sent, it is recommended to constantly poll the JVoIP for these events using the API_GetNotifications function from a timer in around every 200 milliseconds).

Summary by protocol:

- UDP:
  - You might change the wpapiudplistenport parameter if you wish. Default is 19422.
  - It is recommended to bind your UDP socket to 127.0.0.1:19421 (the socket in your own app, not for the JVoIP) and change your target port to the port from where you receive messages on UDP. This helps if the JVoIP was unable to bind to the specified port and is running on a different port as it will send messages to 19421 by default (and changes the target port to the source address once receives any API command).
  - Send commands as clear text in UDP packets to port 19422 (or to the port from where you received the last udp message from JVoIP)
  - You will receive the notifications events from the same port

- TCP:
  - Set the wpapilistenport parameter to 19422
  - Connect to port 19422 and send commands as strings
  - You will receive the notifications events in the same TCP stream
  - You can easily test with a telnet client by connecting to 127.0.0.1 19422 and sending "?function=API_TestEcho&echo=test"

- HTTP:
  - Set the following JVoIP parameters:
    - wpapiudplistenport: 0
    - wpapilistenport: 19422
    - polling: 3
  - Send commands as HTTP GET requests to http://127.0.0.1:19422
  - You can use various formats such as XML or JSON (default is clear text)
  - You need to poll for notification events using API_GetNotifications (from a timer in every 200 msec)
  - You can easily test with a browser: http://127.0.0.1:19422/?function=API_TestEcho&echo=teeeest

Using on a website

You might use VoIP SDK as an applet embedded into your webpage and it will run fine on all Java capable browsers.
Copy JVoIP.jar to your webserver near your html's and refer to it from anywhere from your html with the "applet" tag. Set at least the "serveraddress" parameter to the IP or domain name of your VoIP server (Softswitch/IP-PBX/SIP proxy)

Note: Java applets are deprecated in latest Chrome and Firefox. If you wish to use the SIP library as a Java Applet then we recommend to use IE on Windows.

```
<applet
 archive  = "JVoIP.jar"
 codebase = "."
 code     = "webphone.webphone.class"
 name     = "JVoIP"
 width    = "300"
 height   = "330"
 hspace   = "0"
 vspace   = "0"
 align    = "middle"
 mayscript = "true"
 scriptable = "true"
 alt="Enable or install java: http://www.java.com/en/download/index.jsp"
>
<param name = "serveraddress" value = "SERVER_ADDRESS">
<param name = "loglevel" value = "1">
<param name = "MAYSCRIPT" value = "true">
<param name = "scriptable" value = "true">
<param name = "pluginspage" value = "http://java.com/download/">
```

```
<param name = "permissions" value = "all-permissions">

<b>You must enable java or install if not already installed from <a href="http://www.java.com/en/download/index.jsp"> here </a> </b>

</applet>
```

Embed to your website as an applet and use it with any java enabled browser.
You can also disable the default java user interface (compact=true, width=1, height=1) and create your own using html/css or any other teqniques.

The same API is available also from java script which you can use to implement a VoIP application on your website. You might choose to do some actions in the background, present a single “call” or callback button or to display a phone interface. The important steps are the followings:

1. write a function named “webphonetojs” to catch all messages from the VoIP SDK. Regarding the incoming messages you can display the status of the phone (registered,ringing,in-call,etc), the most important events and alert the user about incoming calls or chat messages.
2. load JVoIP as applet with the webpage using applet tags or with the “toolkit” method with the proper parameters (serveraddress, etc) (the parameters can be preconfigured, but you can also pass them via the API)
3. get a handle for JVoIP (document. applets[0] or check the skins in the demo package for a more complex example)
4. optionally call the API_Register automatically or when the user click on the “Connect” button (by default if you provide a username and a password parameter, the java voip client will register automatically on startup)
5. call the API_Call function when the user clicks on your “Call” or “Dial” button
6. popup a window (or enable an “accept” button) when you receive notifications about incoming calls to your “webphonetojs” function. Then call API_Accept or API_Reject according the user action
7. if you will present a dial pad for the users, then you might call the API_Dtmf function whenever the user presses a button
8. you might put additional buttons or other controls on your interface for the following functions: audio settings, logout, hold, mute, redial, transfer, conference and others

Short example:

```
<SCRIPT LANGUAGE="javascript">
 function webphonetojs(message)
 {
   //this is an optional function if you would like to be notified about JVoIP events
   //this function will be called by JVoIP
   //you will have to parse the message and act accordingly
   alert(message);
 }

 function do_something()
 {
   document.applets[0].functionname();
 }
</SCRIPT>

<input type=button value='JVoIP action' onClick='do_something()'>
```

Example call-flow:

```
var wp = document.applets[0];

wp.API_Register(“11.12.13.14”, “sipusername”, “sippassword”);  //connect to the SIP server
wp.API_Call(-1, “+363012345678”); //make a new call
//wait for connect message by checking the message received on webphonetojs function
//notify the user when the call is ringing or connected
wp.API_MuteEx(-2,true,0);  //can be called when the user press a button
wp.API_MuteEx(-2,false,0);  //reenable audio when the user press a button
wp.API_Hangup(-2); //disconnect all calls

...also call the wp.API_HTTPKeepAlive in every 5 minute to avoid session timeout (defined by the httpsessiontimeout parameter)
```

More details about using the webphone on a webpage can be found here and here.

See the API and the parameters sections below for the details about the usage.

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: auto, UDP, TCP, TLS, TCP tunnel, SOCKS proxy traversal, HTTP proxy traversal, HTTP, VPN tunneling
- NAT/Firewall support: stable SIP and RTP ports ,keep-alive, UPnP, rport support, fast ICE/fast STUN protocols and auto configuration

- Encryption: **TLS/SRTP**, tunneling and peer to peer encrypted media (if direct routing is not disable by your SIP server SDP negotiation)
- 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, FTP or HTTP upload in wav, mp3, gsm or ogg format), custom audio streaming
- DTMF (INFO method in signaling or RFC2833)
- **IM/Chat** (RFC 3428), group chat, SMS, **BLF** and **presence** capability
- Offline chat (late delivery if peer is offline)
- 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
- High availability: auto server failover, transport failover, DNS SRV based failover, backup server configuration, auto-reconnect, network auto-recover
- A long list of other minor features (see the parameter list and the API for the details)
- Unlimited lines, multiple accounts
- Flexibility (all parameters/behavior can be changed/controlled by parameters and/or the API)
- Cross-platform (any OS with Java SE support including Windows, MAC, Linux)
- Works with all JVM versions (backward compatible until JVM v.1.5 / J2SE 5.0 which was released in 2004)
- Always backward compatible including the parameters and the API (you will never have to change your code when upgrading to new versions)

See the software homepage for a general presentation.

Licensing

The Mizu Java VoIP SDK (JVoIP) is sold with life-time 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. You can find the licensing possibilities on the Java VoIP SDK page. After successful tests please ask for your final version at webphone@mizu-voip.com. Mizutech will deliver the JVoIP 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).

The code was written by Mizutech developers from scratch, including both the SIP and media stack, except the third-party modules listed below, with the goal of creating a compact but full featured, robust and easy to use SIP library for Java. The SDK is free from any patents or legal obligation toward third-parties.

The following freely and legally distributable open-source third-party modules might be used in your copy:

- DNS: dnsjava under BSD license
- Opus: the open-source opus codec implementation transformed to Java under three-clause BSD license.
- Speex: the open-source speex codec implementation transformed to Java under revised BSD license.
- G.729: based on open-source G.729 ITU-T implementation transformed to Java. Patent expired in 2017.
- GSM: open-source gsm codec from Tritonus under the GNU Library General Public License.
- OGG/Vorbis: open-source implementation transformed to Java under BSD license.

The library doesn't have any external dependencies (the above libraries are also compiled into the .jar).

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.

You may:

Use JVoIP on any number of computers
Use JVoIP as an SDK embedded in your project
Give the access to JVoIP for your customers or use within your company
Use JVoIP 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 JVoIP
Sell JVoIP as a standalone application (you are allowed to use it only coupled with your project which purpose should not be the same as JVoIP's)
Sell "JVoIP" services for third party VoIP providers and other companies
Use JVoIP with VoIP servers not communicated with Mizutech
Reverse engineer, decompile or disassemble JVoIP
Modify the software in any way (except modifying the parameters and using it via the public API)

Oracle Java licensing:

JVoIP has nothing to do with Oracle licensing.
You can use it with any JVM, including old Oracle Java SE versions (since this is a client library, it has little security implications), new free Oracle Java SE versions or any other compatible Java implementations such as OpenJDK, IceTea, AdoptOpenJDK or any JVM bundled with your OS.

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 JVoIP instances 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)
- will work only maximum 20 minutes and after that you have to restart the JVoIP library or your application
- verifications against the mizu license service
- no headless mode

Note: for the first few calls there might be fewer limitations than described above.
On request we can also send a trial version (will expire after some time but doesn't have the above demo limitations).
To upgrade your demo/trial to an unlocked license, see the details here.

API

You can use JVoIP as an SDK directly embedding into your Java application. Additionally you can access the same API also from JavaScript or via UDP, TCP, HTTP from any application.
Notifications (status updates, events, etc) from JVoIP are sent to UDP localhost:19421 or to your javascript function named "webphonetojs".

Functions

there is a long list of API/parameters but you have to use only a few of them (example). No need for tweaking, already optimized, except if you have some specific needs.
The Android SIP SDK exposes numerous API functions, however this doesn't mean that the usage is difficult. You can ignore most of these functions and use only those few relevant for your needs also outlined in the usage example. For example if you just need to make simple calls, then the following three functions will cover all your needs: API_Start, API_Call, API_Hangup.

Most of the functions **return a boolean** value. True when the operation was completed or initiated successfully, otherwise false.
Some of the functions are executed asynchronously (API_Call, API_Register, etc). This means that it can return a true value immediately and fail later. For example for API_Call the return value means only that the call was initiated successfully. At this point we don't know if the call will be successful (connected) or not. You can get the call status by parsing the notification messages or you can periodically poll JVoIP status with the API_GetStatus function.
Most of the functions doesn't block (except GetNotificationsSync).

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 (this should not be received/sent for endpoints in call, but might be used for other endpoints such as register endpoints)
- 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 JVoIP can do this automatically allocating new channels when needed.

Function string parameters can be 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 except gui parameters (like the colors). Some parameters can take effect only when JVoIP is reinitialized.
This function should be used only in special cases. You should be able to control the java sip client without to use this function by using parameters.

boolean API_SetParameters(String parameters)

You can pass a set of parameters with this function in value=key lines separated by CR (\r\n).

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

Will set the SIP 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 JVoIP.

boolean API_Start()

This has to be called only if you use JVoIP as and SDK or as a java voip library. Don’t call it from JavaScript. Will start the engine.
The autostart behavior can be altered with the startsipstack setting.

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 JVoIP 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: 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.
Accounts parameters:

- Server address
- Username
- Password
- Register interval
- SIP proxy
- SIP realm
- Auth user name (if separate extension id and authorization username have to be used)

Most of the parameters are optional.
The parameters have to be separated by comma (,) and the accounts have to be separated by semicolon (;).
All have to be passed in a single line which should look like this: server,usr,pwd,ival;server2,usr2,pwd2,ival2;

Example: sip1.mydomain,com,1111,xxx,300;sip2.mydomain.com,2222,xxx;

Note: instead of using this API, you can just set the extraregisteraccounts parameter to the exact same string.

boolean API_Unregister(int waitfor)

Will stop all endpoints (hangup current calls if any and unregister).
The waitfor parameter can be set to -1 (auto), 0 (do not wait) or 1 (wait for unregister to complete before to return).
You can modify the function behavior with the waitforunregister and clearcredentialsonunreg parameters.

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 (the SIP client 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

string API_GetLineDetails(int line)

Get details about a line. The line parameter can be -1 (will return the "best" line status).
Will return the following string:
LINEDETAILS,line,state,callid,remoteusername,localusername,type,localaddress,serveraddress,mute,hold,remotefullname

Line line number (might be useful if you pass -1 as line parameter
State same as in STATUS notification
CallID: SIP session id (SIP call-id)
Remoteusername is the other party username (if any)
Localusername is the local user name (or username).
Type is 1 from client endpoints and 2 from server endpoints.
Localaddress: local IP:port
Serveraddress: remote IP:port
Mute: is muted status. 0=no,1=undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Hold: is on hold status: 0=no,1= undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Remotefullname is the other party display name if any

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.
Note: In case if your SIP server (set by the serveraddress parameter) is responsible to handle the call to the target number/extension/user, then you should pass only the peer number, extension or username to this function and not the full SIP URI (the SIP stack will construct the full SIP URI with the serveraddress already set)

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

Note: you should pass only the peer number, extension or username to this function and not the full SIP URI (the SIP stack will construct the full SIP URI with the serveraddress already set)

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.

boolean API_Accept(int line)

Connect incoming call.
You should call this function when there is an incoming ringing call if you wish to connect the 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).
SIP call forward is actually a call redirect (similar to HTTP redirect).
This function should be called only before call is connected. If you wish to forward a call after call connect, you should use the API_Transfer.

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)
This function should be called after call connect as most SIP servers doesn't support REFER before call connect.

More details about call transfers can be found here.

boolean API_TransferDialog()

Instead of calling the API_Transfer function and pass a number, with this function you can let JVoIP 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 be set to one of the followings:

- 0: mute in and out
- 1: mute out (will mute the speakers, so the local user will not hear anything)
- 2: mute in (will mute the microphone, so the other party will not hear anything)
- 3: mute in and out (same as 0)
- 4: mute default (set by the "defmute" parameter, which is "mute microphone only" by default)

Note:

- The mute function should be used only for endpoints in call and it will be ignored if not in call
- If you wish the audio to be always muted, then you might just set the defsetmuted parameter to 1.
- If you don't wish to use audio device at all, then you might set the useaudiodeviceplayback and/or useaudiodevicerecord parameters to false.
- You can also change the automute parameter to automatically mute other lines on new calls.

- If you wish to keep sending RTP packets (with silence) while muted, then set the sendrtponmuted parameter to true
- For more details see here.

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.

Hold means that the media stream is muted in one direction or both directions.
You can modify the holdtypeonhold parameter after your needs (also at runtime, by changing the holdtypeonhold just before calling API_Hold).
This function should be used only after call connect as many SIP servers doesn't support UPDATE/re-INVITE before call connect.

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.

JVoIP is capable for both 3 way SIP conferencing and also to create and handle conferences with unlimited number of participants using its own built-in RTP mixer. Conference parties can use various codec's, including both 8kHz narrowband and 16 kHz wideband.

boolean API_ConfEx(int line, String peer, boolean add)

Add/remove people or line to conference.
If peer is empty than:
-if add is true:
-if line is -2 then it will mix all the currently running calls (if there is more than one call)
-if line is not -2, then it will add the channel to the conference
-if add is false:
-if line is -2 then it will destroy the conference (but will keep the calls on individual lines)
-if line is not -2 then it will remove the selected line from the conference
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, RFC2833 or In-Band 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.

Valid dtmf characters in the passed string are the followings: 0,1,2,3,4,5,6,7,8,9,*,#,A,B,C,D,space.

The dtmf messages are sent with the protocol specified with the dtmfmode parameter.

Feedback about the message delivery can be received with the INFO notifications.

Example: API_Dtmf(-2,” 12 345 #”);

Note: dtmf messages can be also sent by adding it to the called number after a comma. For example if you make a call to 123,456 then it will call 123 and then it will send dtmf 456 once the call is connected.

For more details see the How to DTMF FAQ point.

boolean API_Info(int line, String msg)

Send any SIP INFO message as defined in RFC 2976.

Use this API to send any custom INFO messages to the server or to the connected peer (in case if your server will forward it to other peer and not discard it).

The msg string will be sent in the INFO body.

The Content-Type can be specified with the infocontenttype parameter. For example you might set the infocontenttype to “application/mydata”. If the infocontenttype is not set then it will be auto guessed as application/json, application/xml or application/octet-stream”

Feedback about the message delivery can be received with the INFO notifications.

Example: API_Info(-2,”MyMessage”);

For more details see the Custom INFO messages FAQ point.

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

Send a chat message. (SIP MESSAGE method after RFC 3428)

Peer can be a phone number or SIP username/extension number.

The group parameter have to be sent to the group name only for group chat, otherwise it can be an empty string.

Parse the CHAT notification to check if delivery succeeded or failed.

On successful delivery you will also receive a log like: EVENT, chat sent successfully

On failed delivery you will also receive a log like: WARNING, chat message not delivered

You can also send typing notifications with the SendChatIsComposing API.

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

Send a SMS message if softswitch has SMS delivery capabilities (Otherwise might try to deliver as IM).

The message is delivered as a standard SIP MESSAGE with X-Sms: Yes header. The server is responsible to convert it to SMS.

boolean API_Chat(String peer)

Instead of calling the API_SendChat function and pass a message, with this function you can let the voip client 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.

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

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)
- Line: set to -2 for global or specify channel

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 also played by calling the API_PlaySound with the file set to “lastvoicerecord”.

In case if you wish to receive the audio packets/audio stream in real time, then you can use the local audio streaming capabilities.

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 (raw linear PCM) are supported in the following format: PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits).

This function can be used to play audio locally, but also for remote streaming.

For remote playback, make sure to force a narrowband codec if your file is narrowband and wideband codec (speex, opus) if your file is wideband (16 kHz, 16 bit mono in little-endian).

The file must be found near JVoIP.jar.

- 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 (only for local playback)
- async: false if no, true if playback should be done in a separate thread (false can be used only for local playback, not for streaming)
- islocal: true if the file have to be read from the client PC file system. False if remote file (for example if the file is on the webserver)
- 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: whether this sound is a ringtone/ringback

Examples:

-playback a file locally (mysound.wav must exist near the JVoIP.jar file):

API_PlaySound(1, "mysound.wav", 0, false, false, false, -1,"",false)

-playback a file to the connected remote peer (mysound.wav must exist near the JVoIP.jar file):

API_PlaySound(1, "mysound.wav", 0, false, false, true, -1,"",false)

-stop the playback:

API_PlaySound(0, "", 0, false, false, false, -1,"",false)

Note:

- *The singleaudiostream parameter must be set to 0 if you wish to play simultaneous streams to remote (for multiple concurrent calls)*
- *You might set the useaudiodevicerecord parameter to false if you need streaming but don't have an audio recorder device installed.*
- *If you plan to use this functionality running JVoIP on a server, then have a look at the Using JVoIP on a server FAQ*

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

Stream from PCM audio buffer.

At least the following format is supported: PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits).

Make sure to force a narrowband codec if your buffer is for narrowband audio (8Khz) and wideband codec (speex, opus) if your buffer is wideband (16 kHz, 16 bit mono in little-endian).

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
- len: length of the buff

Notes:

- The buff should not contain any file header (only raw PCM audio data)
- You might set the "useaudiodevicerecord" parameter to "false" if you need streaming but don't have an audio recorder device installed.
- Alternatively you can use the API_StreamSoundStream function to pass an InputStream instead of buff and len.

boolean API_AudioDevice()

Open audio device selector dialog (built-in user interface).

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:

Instead of using this function, you might just call the "API_AudioDevice" to let the users to change their audio settings.

When using the native winapi audio engine, the device names might be truncated to the first 30 character due to the wave audio API limitations.

You can pass the same back to the webphone and it will select the audio device correctly.

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 voip application 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.
Instead of this function you might just call the "API_AudioDevice" function which will present an user interface 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(int line)

Returns voice activity statistics.
It will return the VAD status string as described at the VAD notification.
If line is 0 or negative then it will report global send/recv statistics.
If line is positive then it will report receiver state for the requested line.
You should set the vad parameter to at least 2 (4 for full report), altrough calling this function will also set it, but at first call will not have statistics if it was not set .
Note: if you call this function, VAD notifications will not be sent automatically anymore. (So you will need to continue to poll for the details).
More details here

String API_RTPStat()

Returns media statistics. See the RTPSTAT notification for more details.
For this to work you should set the vad parameter to at least 2 (4 for full report).
Note:
RTP statistics can be sent also automatically if you set the rtpstat parameter.
if you call this function, RTPSTAT notifications will not be sent automatically anymore
More details here.

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 JVoIP instead of continuously polling it with this function call.

String GetNotifications()

Return the notification strings. You should poll for the notifications periodically from a separate thread. It will return accumulated events since the last function call (notifications separated by \r\n -CRLF).
You might set the "polling" parameter to 3 if you wish to use this function (and not socket or webphonetojs events)

More details here.

Note:

- *We are using simple polling for this since Java doesn't have function pointers to be used as a callback and other methods are either deprecated (Observer interface) or too complex (external libraries).*
- *Once a notification string have been read then it will be cleared from the internal list, so it is guaranteed that you will never receive duplicates.*
- *If you are using the library from javascript then just use a timer instead of a thread*

String GetNotificationsSync()

Same as above, but will blocking wait for data (more efficient).

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 above JVoIP 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 softswitch 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 voip endpoint instance preconfigured with the actual contact ("callto" parameter).

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

Contact 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 SIP endpoint (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
    - presenceexpire = 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 JVoIP 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 softswitch 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). *This function will start to send SUBSCRIBE requests.*
5. Use the API_SetPresenceStatus(statustring) function call to change the local user online status with one of the followings strings: Online, Away, DND, Invisible , Offline (case sensitive) . *This function will start to send NOTIFY requests to subscribed parties.*
   Note: presence per user can be also set by upresence_USERNAME parameters.
6. Once these are done, the following notifications can be received from java sip stack:
   NEWUSER,peerusername,displayname,email,URI
   PRESENCE,peerusername,status,details,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/NotExists [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

7. You might use the API_UnSubscribe() API to unsubscribe all endpoints (this includes presence, voicemail and BLF subscribes).

Note: presence is initiated automatically with called contacts after call disconnect if it was n

BLF

BLF (Busy Lamp Field) can be used to monitor the state of an extension and it is implemented as SUBSCRIBE/NOTIFY with dialog event package as described in RFC 3265 and RFC 4235.

The main difference between presence and BLF are the followings:

- *They are implemented differently, based on different RFC's (both based on SUBSCRIBE/NOTIFY, but different message exchange.*
- *Presence is focusing for availability (online/offline), BLF is focusing on call state (ringing/speaking)*
- *BLF usually have to be enabled explicitly to work with selected peers, presence might be enabled by default for all "friends"*
- *Presence can be used to track friend's online state, while BLF is used mostly in offices to track another device call-state such as a secretar tracking it's boss phone to know when it is available/off-hook (for example if it is available to transfer a new call to it).*

Set the enableblf parameter to enable/disable BLF. The following values are defined:

0: disable BLF
1: auto (from here it will auto switch to 0 or 2 regarding the circumstances –whether BLF was initiated and succeed/failed)
2: enable BLF
3: force always (if you set to 3 then it can't be switched off later and will use BLF even after failure)
Default value is 1.

To subscribe to other extensions call state, you can set the blfuserlist parameter or use the API_CheckBLF(userlist) to subscribe to other extension(s) state changes (users separated by comma). To remove an extension, just modify the blfuserlist parameter or call the API_DisableBLF(userlist) API.

Make sure that the peer extension(s) also has BLF support (so it can respond with NOTIFY for the BLF SUBSCRIBE).
Once the state of the remote extension(s) are changed, you will receive BLF notifications in this format:

BLF,peerusername,direction,state,callid

Fields:

- BLF: state header string
- peersusername: extension username
- direction
  - undefined (not for calls or not announced by the peer)
  - initiator (outgoing call)
  - receiver (incoming call)
- state
  - trying (call connect initiated)
  - proceeding (call connecting)
  - early (ringing or session progress)
  - confirmed (call connected)
  - terminated (call disconnected)
  - unknown (unrecognized call state received)
  - failed (BLF subscribe or notify failed)
- callid: optional value if reported from the remote extension (the SIP call-id of the call)

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()
This is needed only if used from web via JavaScript.
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 JVoIP instances (when your html page was closed or crashed but JVoIP 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 Java 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_StartGUI()
Launch the built-in simple phone user interface even if the app was launched from command line.

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

boolean API_Exit()
Will stop all endpoints and terminates the java sip application. This function call is optional when you unload the JVoIP java module 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”)

void API_CheckConnection()
You might call this function to quickly recover from connection failures on events not know by JVoIP such as app switched to foreground (otherwise JVoIP should auto-recover from network failures)

String API_LineToCallID(int line)
Get the SIP Call-ID for a line number.

int API_CallIDToLine(String callid)
Get the line number for a SIP Call-ID.

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).
Example: API_SetSIPHeader(-1, "X-MyData: VALUE”);
Multiple headers can be separated by CRLF (\r\n). You can also set this with the customsipheader parameter.

String API_GetSIPHeader(int line, String hdr)
Return a SIP header value received by JVoIP. 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_GetSIPMessage(int line, int dir, int type)
Return the last received or sent SIP signaling message as raw text.
Dir:
- 0: in (incoming/received message)
- 1: out (outgoing/sent message)

Type:
- 0: any
- 1: SIP request (such as INVITE, REGISTER, BYE)
- 2: SIP answer (such as 200 OK, 401 Unauthorized and other response codes)
- 3: INVITE (the last INVITE received or sent)
- 4: the last 200 OK (call connect, ok for register or other)

String API_GetLastRecInvite()
Return the last received INVITE message.

String API_GetLastSentInvite()
Return the last sent INVITE message.

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

String API_GetLastRecFileName (String line)
Returns the last recorded file name.

String API_SendChatIsComposing (String line, String number)
Send typing notification.

String API_GetAddress()
Return the local SIP listener address (IP:port)

boolean API_IsOnline()
Return true if network is present

boolean API_IsRegistered()
Return true if JVoIP is registered (“connected”) to the SIP server.

int API_IsRegisteredEx()
Returns extended registration state: 0=unknown,1=not needed,2=yes,3=working yes,4=working unknown,5=working no,6=unregistered,7=no (failed)

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_GetDiscReasonText()
Return the disconnect reason of the last disconnected call.

String API_GetGlobalStatus()
Returns SIP stack state/details.

int API_GetAccountRegState(String domain, String proxy, String username, String sipusername, boolean pushnotify, boolean strict)
Query account register state. Useful if you are using multiple accounts.
Returns -1: no such account register ep, 0: working, 1: success, 2: failed, 3: unregistered.
Parameters domain, proxy, username, sipusername will be used to match the account.
Always set the pushnotify to false.
If the strict parameter is false, then will check the best matching account after the passed parameters.
If the strict parameter is true, then will enforce exact parameter match when checking after the endpoints (server domain/username/etc).

String API_GetAccountRegStateString(String domain, String proxy, String username, String sipusername, boolean pushnotify, boolean strict)
Query account register state. Useful if you are using multiple accounts.
Return status text (for example "Registered" if the account is registered to the server).
Parameters domain, proxy, username, sipusername will be used to match the account.
Always set the pushnotify to false.
If the strict parameter is false, then will check the best matching account after the passed parameters.
If the strict parameter is true, then will enforce exact parameter match when checking after the endpoints (server domain/username/etc).

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.

String API_GetProfileStatusText(String username)
Get the profile status text for the user.
Note: our profile status text can be set with the profilestatustext parameter.
This feature might be useful to implement a “My Profile” page in certain softphones.

String API_GetMySIPURI(boolean all)
Will return the SIP URI on which the current endpoint can be reached such as username@server.com or username@localip:port.
If all is true, then it will return all possible URI’s separated by comma.

boolean API_ShowLog()
Show a new window with logs.

void API_AddLog(String msg)
Add a message to JVoIP 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 JVoIP 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

boolean API_ClearCredentials()
Clear existing user account details.

boolean API_DelSettings (int level)
Delete settings and data. Levels: 0: nothing, 1: settings, 2: everything

int API_IsEncrypted ()
Returns whether the connection is encrypted. -1: unknown, 0=no,1=partially/weak yes,2=yes,3=always strong

String API_GetBindir()
Returns the application path (folder with the app binaries or app home folder)

String API_GetWorkdir()
Returns the application working directory (data folder).

String API_GetAltWorkdir()
Returns the application alternative working directory (such as folder on external SD card).

int API_ShouldReset()
Check if the sipstack should be restarted. Usually this is required only on local network change (such as IP change or changing from mobile data to wifi).
Possible return values: 0=no,1=not registered for a while,2=network changed

boolean API_ShouldResetBeforeCall()
This function might be called before calls and you should quickly restart the sipstack if returns true (the continue to make the call).

boolean API_RecFiles_Del()
Delete local recorded files.

boolean API_ReStart()
Restart the SIP stack.

boolean API_GetDeviceID()
Returns an unique device ID.

Notifications

Notifications means events received from the SIP library.
For maximum flexibility, the JVoIP SIP library implements multiple ways to receive the SIP notifications

- polling with API_GetNotifications or API_GetNotificationsSync (preferred method when used as a Java library as described here)
- webhonetojs (useful only if you use the library as an applet from JavaScript as described here)
- socket (notifications sent via UDP or TCP if you use the library via socket as described here)

You will have to parse the received strings from your code by first splitting them by line, since more than one line can be received at once (separated by CRLF or with ",NEOL \r\n ", each line represents a new notification). The most important notification is the STATUS message which can be used to learn the SIP stack state (global state or per line state).
Notifications might be prefixed with the "WPNOTIFICATION," string (you should remove this before parsing the rest of the line).

The parameters are separated by comma ‘,’. 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.
A code example can be found here.
The easiest way to get started is to just log out all messages at first and from there the usage should be more obvious.

Line parameter:
Most of the messages contains a line parameter which is set to -1 for the global state or 0,1,2,3… for the individual lines. For this reason you might see duplicated notifications: one sent for the global state, another sent for the actual line state.
For you wish to implement something basic, then it is enough of you check the notifications only for the global state and ignore the line states.
If you wish to handle multi-lines explicitly then you should check also the state of the individual lines (you might ignore the notifications about global state in this case).

The following notification messages are defined:

STATUS,line,statustext,peername,localname,endpointtype

You will receive STATUS notifications when the SIP session state is changed (SIP session state machine changes) or periodically even if there was no any change.
A typical status strings looks like this:
STATUS,line,statustext,peername,localname,endpointtype,peerdisplayname,[callid],online,registered,incall,mute,hold,encrypted,video,group,rtpsent,rtprec,rtploss,rtplosspercet,serverstats

The **line** parameter can be -1 for general status or a positive value for the different lines.
General status means the status for the “best” endpoint.
This means that you will usually see the same status twice (or more). Once for general sip client status and once for line status.
For example you can receive the following two messages consecutively: STATUS,-1,Connected,…
You might decide to parse only general status messages (where the line is -1), messages for specific line (where line is zero or positive) or both.

Status notifications are sent on state change but also at a regular interval (so you can see the same notification again and again even if there was no state change).

The following **statustext** values are defined **for general status (line set to -1):**

- Starting...
- Idle.
- Ready
- Connecting…
- Securing…
- Register…
- Registering… (or “Register...”)
- Register Failed
- No network
- Server address unreachable
- Not Registered
- Registered (or “Registered.”)
- Unregistered
- Accept
- Starting Call
- Call
- Call Initiated
- Calling…
- Ringing…
- Incoming…
- In Call (xxx sec)
- Hangup
- Call Finished
- Chat (or Messaging)

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 (voip library started)
- Ready (sip stack started)
- Outband (notify/options/etc. you should skip this)
- **Register** (from register endpoints) (or “Register…” or “Registering...”)
- Unregister
- 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). Please note that these are sent only for the actual line (1,2,etc) and not as global state (-1).

Peername: the other party username (if any)
Localname: the local user name (or username).
Endpointtype: 1 from client endpoints and 2 from server endpoints.
Peerdisplayname: the other party display name if any
CallID: SIP session id (SIP call-id)
Online: network status. 0 if no network or internet connection, 1 if connectivity detected (always the global status)
Registered: registration state. 0=unknown,1=not needed,2=yes,3=working yes,4=working unknown,5=working no,6=unregistered, 7+ =no (failed) (always the global status)
InCall: phone/line is in call. 0=no,1=setup or ringing,2=speaking
Mute: is muted status. 0=no,1=undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Hold: is on hold status: 0=no,1= undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Encrypted: encryption status: -1: unknown, 0=no,1=partially/weak yes,2=yes,3=always strong (or if you wish to simplify: 0 and -1 means no, others means yes)
Video: set to 1 for video calls (with video media offer). Otherwise 0.
Group: group string for group chat and conference calls (members separated by |)
RTPSent: number of sent RTP packets (only if endpoint is in call)
RTPRec: number of received RTP packets (only if endpoint is in call)
RTPLoss: number of lost RTP packets (only if endpoint is in call)
RTPLossPercent: percent of the lost RTP packets (only if endpoint is in call)
ServerStat: RTP statistics received from the server, if any (only if endpoint is in call)

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 periodic status messages during calls (might be useful if you are interested in RTP statistics during a call).

PRESENCE,peername,state,details,displayname,email

This notification is received for presence changes (peers online status).

Line: used phone line
Peername: username of the peer
State and details: presence status string; one of the followings:
CallMe,Available,Open,Pending,Other,CallForward,Speaking,Busy,Idle,DoNotDisturb,DND,Unknown,Away,Offline,Closed,Close,Unreacheable,
Unregistered,Invisible,Exists,NotExists,Unknown,Not Set
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).

BLF,peerusername,direction,state,callid

This notification is received for incoming BLF messages.
Described in the BLF section above.

DTMF,line,msg

Incoming DTMF notification.
Msg is a string parameter representing the incoming DTMF digit. Usually one of the followings: 0123456789*#ABCDF
Supported receive dtmf methods are SIP INFO in SIP signaling and RFC 2833 in RTP.
Example: DTMF,1,7 (dtmf digit"7" received on first line)

INFO,type,line,peername,text

Notifications about incoming/outgoing INFO or DTMF messages.

Note: for incoming DTMF you should watch for the above described DTMF notification instead of this! DTMF notification will be triggered even if the message was received with SIP INFO.

Parameters:
Type: The possible values for the type parameter is the following:
ERROR: Outgoing DTMF (API_Dtmf) or INFO (API_Info) message failed
WARNING: Outgoing DTMF failover from SIP INFO to RFC 2833 or In-Band on answer timeout or error response
OK: Outgoing DTMF (API_Dtmf) or INFO (API_Info) was sent successfully
REC: Incoming INFO message receive. Note that if the incoming message is a dtmf digit, then a DTMF notification will be triggered instead of INFO
Line: Phone line
Peername: Other party username
Text: INFO message body when type is REC or any additional info if type us ERROR, WARNING or OK.

Examples:
INFO,ERROR,1,1111,timeout (on API_Dtmf or API_Info message send failed to 1111 on line 1)
INFO,WARNING,1,1111,failback to rfc2833 (on API_Dtmf failback from INFO to rfc2833 or in-band)
INFO,OK,1,1111 (on API_Dtmf or API_Info message sent successfully to 1111 on line 1)
INFO,REC,1,1111,hello (on SIP INFO message received from 1111 which is not DTMF)

Note:
When sending DTMF in RTP (inband or rfc2833) then it is not possible to get feedback whether the message was actually delivered. In this case you will receive an INFO,OK when the message send was initiated successfully. Otherwise if SIP INFO is used, then you will receive INFO,OK only on 2xx answer from the server or the other peer.

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
Note:
- *If the text begins with "_BASE64_" then you must base64 decode the rest of the string before presenting it to the user.*
- *If the text begins with "GROUP: " then it is a group chat. Example: "GROUP: Katie,John: hi". In this case you might remove the header (until the second colon) and display the message in a separate thread. If the grouop name is „null" then the message doesn't belong to any group conversation. If group is not set, then it means that the remote peer doesn't have group chat capabilities and you might lookup the group where the incoming message it belongs.*

CHATREPORT,line,peername,status,statustext,group,md5

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: 0 means chat send initialized, 1 means sending in progress, 2 means successfully sent, 3 means failed, 4 means queued for later send
Statustext: status text if any (such as delivery error reason if status is 3)
Group: optional parameter for group chat (member names separated by |)
MD5: the MD5 checksum of the message text

Note:

If the offline messaging is not disabled (offlinechat parameter is not set to 0) then JVoIP can retry later (on next start and/or when any SIP request is received from the target user) reporting status 4 instead of 3 on failed delivery. The MD5 checksum in this case will be calculated from concatenated pending message text and not for the last message.

CHATCOMPOSING,line,peername,status

Chat "is composing" notifications.

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,reasontext

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=JVoIP, 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

STOP,api

This message is sent when the SIP stack is terminated/destroyed.

SHOULDRESET,reason text

You should restart JVoIP (or reinit the library) when you receive this notification.
This is usually sent when network was changed (connection type, IP address), thus it might be best to reinitialize the whole SIP stack to recalculate the optimized environment variables and perform the auto network discovery process again (such as STUN lookup, but there are many others).
The message is not sent while in call.

LINE,number

This message is sent when the active line is changed.

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 "events" parameter to 2 or higher and it also depends on the loglevel.

GROUP,line,peers

Sent for conference calls. You might update the displayed peer name with the peers strings received here.
The peers parameter is the list of members separated by |. For example: Kate | John | Linda

VREC,line,stage,type,path,reason,source

Voice upload status (for voice recording / call recording).
line: channel number *(note: with stage 3 and 4 it will always report -1 or -2 means default/not specified)*
stage: 0:disabled, 1:call record begin,2:save begin, 3: save success, 4: save fail *(note: stage 0 might not be reported)*
type: upload method: 0: unknown, 1: local file, 2: ftp, 3: http, 4: server
path: upload path/file *(note: if stage is 1 then type and path is not reported yet)*
reason: failure reason *(if stage is 4)*
source: USER or AUTO (who initiated the download)

PLAYREADY,line,callid

Audio streaming finished (initiated by API_PlaySound to remote peer).
Note: this notification is not sent with call disconnect (call disconnect will always terminate the audio streaming).

POPUP,txt

Should be displayed for the users in some way (hint/toast).

LOG,TYPE,txt

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

These messages will be received only if you set the "events" parameter to 3 and also depends on the loglevel.

VAD,parameters

Voice activity.
This is sent in around every 3000 milliseconds (3 seconds) by default (configurable with the vadstat_ival parameter in milliseconds) if you set the "vadstat" parameter to 3 or 4 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

If the vadbyline parameter is set to 1 then individual lines will report in the following format:
VAD,line: X remote_vad: ON remote_avg: 0 remote_max: 0 remote_speaking: no

More details here.

RTPSTAT,quality,sent,rec,issues,loss

Media quality reports.
Can be enabled by the rtpstat parameter.
The parameters are calculated since the last RTPSTAT notification (for the past x seconds).

Parameters:

- Quality: quality points between 0 and 5. The calculations considers many factors such as RTP issues, RTCP reports, codec problems, packet loss, packet delay, jitter and processing delay.
  The following values are defined:
  - 0: no audio or non-recognizable voice
  - 1: very bad quality

    - 2: bad quality
    - 3: medium quality
    - 4: good quality
    - 5: excellent quality
- Sent: RTP packets sent
- Rec: RTP packets received and played
- Issues: number of issues (any issues are counted, such as sequence number mismatch or packet drop)
- Loss: lost packets

More details here.

LOG,RTP,txt

RTP details at the end of the call.
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%

Other notifications

Format: messageheader, messagetext. The followings are defined:

"**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
"**SERVERCONTACTS**" contact found at local VoIP server
"**NEWUSER**" new user request
"**ANSWER**" answer for previous request (usually http requests)

Example session

Here is how the notifications looks like in a typical session (start, register, make a call, hangup, terminate):

```
WPNOTIFICATION,START,api,NEOL
WPNOTIFICATION,START,sip,NEOL
WPNOTIFICATION,STATUS,-1,Initialized,NEOL
WPNOTIFICATION,STATUS,-1,Register...,NEOL
WPNOTIFICATION,EVENT,EVENT,Connecting...,NEOL
WPNOTIFICATION,STATUS,-1,Registering...,NEOL
WPNOTIFICATION,STATUS,0,Registering,istvantest2,istvantest2,1,voip.mizu-voip.com,[2e1123294504249584311k49333rmwp],,1,4,0,0,0,0,0,NEOL
WPNOTIFICATION,CREDIT,Credit: 4.2 USD,NEOL
WPNOTIFICATION,STATUS,-1,Registered.,NEOL
WPNOTIFICATION,EVENT,EVENT,Authenticated successfully. [ep: 0 2e1123294504249584311k49333rmwp Registering 12538],NEOL
WPNOTIFICATION,STATUS,0,Registered,istvantest2,istvantest2,1,voip.mizu-voip.com,[2e1123294504249584311k49333rmwp],,1,2,0,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,-1,Starting call to testivr3,NEOL
WPNOTIFICATION,EVENT,EVENT,call [wpmain],NEOL
WPNOTIFICATION,STATUS,-1,Call,NEOL
WPNOTIFICATION,STATUS,-1,Call Initiated,NEOL
WPNOTIFICATION,STATUS,1,CallSetup,testivr3,istvantest2,1,testivr3,0,0,0,100,,[6e4351661777543861707k49334rmwp],1,2,1,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,1,Routed,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,1,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,1,InProgress,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,1,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,-1,Calling...,NEOL
WPNOTIFICATION,STATUS,1,Ringing,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,1,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,-1,Ringing...,NEOL
WPNOTIFICATION,STATUS,1,CallConnect,testivr3,istvantest2,1,testivr3,0,0,0,100,,[6e4351661777543861707k49334rmwp],1,2,2,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,1,Speaking,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,2,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,-1,Speaking (0 sec),NEOL
WPNOTIFICATION,STATUS,-1,Hangup,NEOL
WPNOTIFICATION,STATUS,-1,Speaking (2 sec),NEOL
WPNOTIFICATION,EVENT,EVENT,hangup [wpmain],NEOL
WPNOTIFICATION,STATUS,1,CallDisconnect,testivr3,istvantest2,1,testivr3,98,98,0,100,,[6e4351661777543861707k49334rmwp],1,2,0,0,0,0,0,NEOL
WPNOTIFICATION,STATUS,-1,Call Finished,NEOL
WPNOTIFICATION,CDR,1,testivr3,istvantest2,testivr3,voip.mizu-voip.com,1499,1969,1,User Hung Up (exD),[6e4351661777543861707k49334rmwp],NEOL
WPNOTIFICATION,STATUS,1,Finishing,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,0,0,0,0,0,NEOL
WPNOTIFICATION,EVENT,EVENT,Call duration: 2 sec [ep: 1 6e4351661777543861707k49334rmwp Finishing 12538],NEOL
WPNOTIFICATION,STATUS,1,Finished,testivr3,istvantest2,1,testivr3,[6e4351661777543861707k49334rmwp],,1,2,0,0,0,0,0,NEOL
```

WPNOTIFICATION,STATUS,-1,Registered.,NEOL
WPNOTIFICATION,STOP,api,NEOL

Parse notifications code example

Below is a short code snippet demonstrating basic notification parsing.
Notifications handling is all about a bit of string parsing so maybe better if you just implement it from scratch so you will be more familiar with your own logic then the string handling practices in this example code.
In this example we assume that we receive the notifications in a function called ProcessNotifications.

```
public void ProcessNotifications(String receivednot)
{
    if (receivednot == null || receivednot.length() < 1) return;

    // we can receive multiple notifications at once, so we split them by CRLF or with ",NEOL \r\n" and we end up with a
String array of notifications
    String[] notarray = receivednot.split(",NEOL \n");
    for (int i = 0; i < notarray.length; i++)
    {
        String notifywordcontent = notarray[i];
        if (notifywordcontent == null || notifywordcontent.length() < 1) continue;
        notifywordcontent = notifywordcontent.trim();
        notifywordcontent = notifywordcontent.replace("WPNOTIFICATION,", "");

        // now we have a single notification in the "notifywordcontent" String variable
        Log.v("JVOIP", "Received Notification: " + notifywordcontent);

        int pos = 0;
        String notifyword1 = ""; // will hold the notification type
        String notifyword2 = ""; // will hold the second most important String in the STATUS notifications, which is the
third parameter, right after the "line' parameter

        // First we are checking the first parameter (until the first comma) to determine the event type.
        // Then we will check for the other parameters.
        pos = notifywordcontent.indexOf(",");
        if(pos > 0)
        {
            notifyword1 = notifywordcontent.substring(0, pos).trim();
            notifywordcontent = notifywordcontent.substring(pos+1, notifywordcontent.length()).trim();
        }
        else
        {
            notifyword1 = "EVENT";
        }

        // Notification type, "notifyword1" can have many values, but the most important ones are the STATUS types.

        // After each call, you will receive a CDR (call detail record). We can parse this to get valuable information
about the latest call.
        // CDR,line, peername,caller, called,peeraddress,connecttime,duration,discparty,reasontext
        // Example: CDR,1, 112233, 445566, 112233, voip.mizu-voip.com, 5884, 1429, 2, bye received
        if (notifyword1.equals("CDR"))
        {
            String[] cdrParams = notifywordcontent.split(",");
            String line = cdrParams[0];
            String peername = cdrParams[1];
            String caller = cdrParams[2];
            String called = cdrParams[3];
            String peeraddress = cdrParams[4];
            String connecttime = cdrParams[5];
            String duration = cdrParams[6];
            String discparty = cdrParams[7];
            String reasontext = cdrParams[8];
        }
        // lets parse a few STATUS notifications
        else if(notifyword1.equals("STATUS"))
        {
            //ignore line number. we are not handling it for now
            pos = notifywordcontent.indexOf(",");
            if(pos > 0) notifywordcontent = notifywordcontent.substring(pos+1, notifywordcontent.length()).trim();

            pos = notifywordcontent.indexOf(",");
            if(pos > 0)
            {
                  notifyword2 = notifywordcontent.substring(0, pos).trim();
                  notifywordcontent = notifywordcontent.substring(pos+1, notifywordcontent.length()).trim();
            }
```

```
            else
            {
                  notifyword2 = notifywordcontent;
            }

             if (notifyword2.equals("Registered."))
             {
                 // means the SDK is successfully registered to the specified VoIP server
             }
             else if (notifyword2.equals("CallSetup"))
             {
                 // a call is in the setup stage
             }
              else if (notifyword2.equals("Ringing"))
             {
                 // check the other parameters to see if it an incoming call and display an alert for the user
             }
             else if (notifyword2.equals("CallConnect"))
             {
                 // call was just connected
             }
             else if (notifyword2.equals("CallDisconnect"))
             {
                 // call was just disconnected
             }
             else if (notifyword1.equals("CHAT"))
             {
                 // we received an incoming chat message (parse the other parameters to get the sender name and the text to
be displayed)            }
        }
        else if(notifyword1.equals("ERROR"))
        {
            // we received an error notification; at least log it somewhere
            Log.e("JVOIP", "ERROR," + notifywordcontent);
        }
        else if(notifyword1.equals("WARNING"))
        {
            // we received a warning notification; at least log it somewhere
            Log.w("JVOIP", "WARNING," + notifywordcontent);
        }
          else if(notifyword1.equals("EVENT"))
        {
            // display important event for the user
            Log.v(notifywordcontent);
        }
    }
}
```

Parameters

Parameters can be specified in the following ways:

- command line (when used as a standalone executable. Example: JVoIP.jar serveraddress=1.2.3.4 username=xxx password=xxx loglevel=5)
- config file: wpcfg.ini file in ini file format
- using the API_SetParameter function
- SIP signaling (sent from server) with the x-mparam header (or x-mparamp if need to persist). Example: x-mparam=loglevel=5;aec=0
- environment variable (prefix environment variables names with “wp_”. For example wp_username)
- if used from a webpage:
  applet parameters (by using the applet tag and set the parameters like: <param name = "parameter_name" value = "parameter_value">
  webpage URL (the webhone will simply look at the embedding document/window url at startup. Prefix all parameter with “wp_”)
  cookies (prefix cookie keys with “wp_”. For example wp_username)
  skin config file: config.js (if you are using our skin templates)

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.

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 JVoIP 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 JVoIP 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.
As the SIP server you can use any softswitch, IP-PBX, SIP proxy server or SIP gateway (with or without registrar).
It might be possible that your service requires also SIP proxy configuration which can be set with the proxyaddress parameter.

username

(string)
This is the SIP username (A number/Caller-ID for the outgoing calls). The JVoIP SIP endpoint will authenticate with this username on your SIP server.
Default value is empty.

Note:
You should set only the username part of your SIP account here, not the full SIP URI. For example if your account looks like abc@xyz.com, then you should set xyz.com as the serveraddress parameter and abc as the username parameter.
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.
You might also specify the display name. More details here.

password

(string)
SIP authentication password.
If your server doesn't require digest authentication or if you wish to make peer to peer calls then this parameter can be omitted.
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 here.
Default value is empty.

register

(int)
With this parameter you can set whether the SIP UA should register (connect) to the sip server at startup.

Possible values:

- 0: no
- 1: auto guess (yes if serveraddress/username/password is set, otherwise no)
- 2: yes

Default value is 1.

JVoIP will also reregister automatically based on the registerinterval parameter (there is no need to periodically call the API_Register).

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.

Usually when this parameter is true, then the "compact" parameter is also set true. Usually when this parameter is false, then the "compact" parameter is also set 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.

Other Parameters

These parameters are more rarely used or should be used only if you have at least a minimal technical knowledge (VoIP and/or Java). 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).

codebase

(string)
Only if you use it as applet from browsers.
This optional attribute specifies the base URL of JVoIP--the directory that contains JVoIP code. If this attribute is not specified, then the document's URL is used (your html page URL).
Use it if you deploy JVoIP.jar in a different directory on your webserver (not the same directory as your webpage).
For the toolkit deployment use "JAVA_CODEBASE" instead of "codebase".
Default is '.' which means the same directory (the html document directory)

These parameters are more rarely used or should be used only if you have at least a minimal technical knowledge (VoIP and/or Java)

use_rport

(int)
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)
(You might adjust also the use_fast_stun parameter if you change the use_rport)
Default is 1

upnpnat

(int)
Nat traversal via UPnP supported routers (IGD)
0: disable
1: enable
Default is 1

use_fast_ice

(int)
Fast ICE negotiations (for p2p rtp routing):
-1: suggest server side media routing (X-P2PM: 1)
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.
If enabled, then the media might be negotiated directly between the endpoint (without the use of server suggestions in the SDP) and might be routed directly encrypted.

use_fast_stun

(int)
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
2=use only if both tests match even if not symmetric (recommended)
3=use for symmetric NAT even if only one match
4=always
5=use even on public IP
Change if you have NAT issues (depending on your server type and settings)
(You might adjust also the use_rport parameter if use_fast_stun is changed)
Default is 2

udpconnect

(int)
Specify whether the UDP have to be connected before sending on the socket. (Some IP-PBX 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

(int)
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

(int)
NAT keep-alive packet send interval in milliseconds.
Keep-alive is a mechanism to keep the NAT open between register resends (so the server can initiate a new incoming transaction on the same stream with no issues, such as incoming call).

Possible values:
- -1: auto (will default to around 28000 -28 sec- on UDP and 600000 – 10 min- on TCP)
- 0: disabled
- other: interval in milliseconds (must be above 3000, otherwise treated as seconds)

Default value is -1.

registerinterval

(int)
Registration interval in **seconds** (used by the re-registration expires timer).
Default value is -1 (auto guess)

Valid range is -1 (auto) or between 10 and 30000.

Notes:
- *Many servers will not accept values above 3600 (although the SIP stack can be auto adjust it based on server negotiation)*
- *The actual resend of the REGISTER messages might be done at a shorter interval to cover any potential packet loss and network/server delays.*
- *When set to auto guess (-1), it will calculate an optimal interval depending on the circumstances such as network type, server load, transport method. It will result to around 90-180 under normal conditions on UDP.*

- *If your softswitch 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.*
- *If the register expire interval is not accepted by the server, then the SIP stack is capable to automatically negotiate a new value as directed by the server Min-Expire header or auto-guess.*

needunregister

(boolean)
Set to false to prevent unregister messages to be sent by JVoIP (for example to prevent unregister on web page reload).
Default is true

contactaddressfailback

(int)
Specify if to try to reregister with another (better) local address if register fail or if previous (first) register went with a private address (and the server is on public IP).
Possible values:

- -1: auto (same as 2 if server is on public IP and stun/rport are not disabled and server is not known NAT friendly; otherwise same as 1)
- 0: never
- 1: if failed (will try to reregister with other local address on register failure)
- 2: always (also try to reregister with public address when possible which helps for servers with poor NAT support)

Default: -1

Note:
With the unbindbeforereregister parameter you can also specify if to unbind (unregister) the previous (private) address before registering the new one.
This might be forced on servers with no or poor call fork support, otherwise most servers should be able to just replace the previous address without the need to unregister.
Possible values: -1: auto, 0: never, 1: always (default).

extraregisteraccounts

(string)
Use this setting for multi-account registration.
You can specify multiple SIP accounts in the following format:
IP,usr,pwd,t,proxy,realm,authuser;IP2,usr2,pwd2;IP3,usr3,pwd3,t3,proxy3,realm3;
where:

- IP: is the SIP server IP or domain name
- usr: is the SIP username
- pwd: is the SIP password
- t: is the register timeout in seconds (optional)
- proxy: SIP proxy (optional)
- realm: SIP realm (optional)
- authusr: auth (if separate extension id and authorization username have to be used)

Note:
Alternatively you can just launch multiple instances for JVoIP with different accounts.
You can also use the API_RegisterEx at runtime for the same functionality

acceptsrvexpire

(int)
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

(int)
If to treat session progress (183) responses as ringing (180). This is useful because some IP-PBX 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 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 JVoIP 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)

allowcallredirect

(int)
Set to 1 to auto-redial on 301/302 call forward.
Set to 0 to disable auto call forward.
Default value is 1.

natopenpackets

(int)
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 parameter to specify the format. -1 means auto, 1 means short CRLFCRLF packet (default). 2 means full RTP packet with zeroed content.

earlymedia

(int)
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, JVoIP 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 softswitch.

setfinalcodec

(int)
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.

backupserver

(string)
Specify secondary SIP server address for failover in case if you have multiple SIP servers.
You can specify domain, ip and if port is not 5060 then you must also specify the port like mybackupserver.com:7080.

You might also specify a backupserverdomain parameter (use IP in backupserver and domain name in backupserverdomain).
More details can be found here.

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

(int)
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

(int)
Transport protocol for the SIP signaling.
-1: Auto detect
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 gateway or server)
5: Auto (automatic failover from UDP to HTTP as needed if tunnel gateway or server is used)
Default is -1.

Note:
To enable full encryption set both the transport and the mediaencryption parameter to 2.
If you set TLS (2), then make sure that you are using the correct serveraddress parameter. SIP servers usually listens for TLS on port 5061, thus the serveraddress should look like yourdomain:5061.
If the SIP port is configured to 5061 and the transport parameter is not set, then it might default to 2 (TLS).
If you need strict TLS verifications then you might set the tlspolicy parameter to 3 (0: def-auto, 1: allows self signed or invalid cert, 2: medium with warning if fails but continue, 3: with server cert validation and disconnect if not secure)

mediaencryption

(int)
Media encryption method for the RTP streams.
-1: auto guess (auto set to 1 if TLS)
0: not encrypted (default)
1: auto (will encrypt if initiated by other party)
2: SRTP (recommended for RTP encryption)
3: ZRTP (optional module)
Default is -1.

Note: To enable full encryption then set both the transport and the mediaencryption parameter to 2.

strictsrtp

(int)
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

srtp_suite

(string)
You might specify the preferred srtp suite.
Possible values:
AES_CM_128_HMAC_SHA1_80 (recommended)
AES_CM_128_HMAC_SHA1_32 (supported)
AES_CM_256_HMAC_SHA1_80 (beta / not tested)
AES_CM_256_HMAC_SHA1_32 (beta / not tested)

Default: AES_CM_128_HMAC_SHA1_80

JVoIP might be able to negotiate also other values as suggested by your server.

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

(int)
DTMF send method.
0=disabled
1=SIP INFO method (out-of-band in SIP signaling INFO messages)
2=auto (auto guess from peer advertised capabilities)
3=INFO + NTE
4=NTE (Named Telephone Events as specified in RFC 2833 and RFC 4733)
5=In-Band (DTMF audio tones in the RTP stream)
6=INFO + InBand

Default is 2 (and you should change it only if you have a good reason to do so, since the auto-detect should work in all circumstances, except if your SIP server is sending bogus DTMF capabilities).

Note:
DTMF digits can be sent with the API_Dtmf or if you press a digit on the built-in user interface (if that is used).
When more than one method is used (dtmfmode 3 or 6), the receiver might receive duplicated dtmf digits.
If the message sending was initiated with SIP INFO, then it might failover to rfc2833 or inband if no answer or error code was received from server.

sendearlydtmf

(int)
Specify whether to allow sending DTMF digits before call connect
0=no
1=auto (yes if rfc2833 or inband is allowed and already sent/received rtp packets)
2=yes (always send)

Default: 1

offlinechat

Will try to resend not delivered messages later (on next register and/or when any message received from peer). The offline message queuing will take care of filtering out duplicates and will try to resend the message multiple times on failure until max number of resend or timeout reached.
(int)
0=no (disable offline chat)
1=yes (default)
2=force always

voicemail

(int)
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. You might set it to 3 if your server has support for MWI to be sure that JVoIP will check the voicemail in all circumstances.

voicemailnum

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

transfertype

(int)
-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 JVoIP 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 JVoIP 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)
7=transfer with no hold and no disconnect
8=transfer with conference (will put the parties to conference on transfer)

Default is -1 (which is the same as 6)

Note:

- *Unattended means simple immediate transfer (just a REFER message sent)*
- *Attended transfer means that there will be a consultation call first*
- *The most popular transfertypes are 1, 5 and 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)*
- *More details can be found here.*

transfwithreplace

(int)
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

(int)
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 (disconnect if transfertype is 1)

disconincomingrefer

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

Default is -1

inversetransfer

(int)
Specify inverse attended transfer.
0: no. The REFER will be sent to the ep for which the API_Transfer was called. This is the standard behavior.
1: yes. The REFER will be sent to the ep for which the API_Hangup was called. Might be used only if the original ep (for which the API_Transfer was called) doesn't support transfer (REFER sip message).
Default is 0 (standard transfer behavior)

transferdelay

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

newdialogforrefer

(int)
Specify if the REFER have to be sent in a new dialog (for compatibility reasons).
0: no (after SIP standards)
1: yes, with no to tag
Default is 0

useserverdomainforrefer

(int)
Specify the domain part to be used in REFER for the Refer-To and Referred-By headers.
0: no (use default domains loaded from the dialog)
1: yes for Refer-To only
2: yes for Referred-By only
3: auto (usually yes for both)
4: force server domain for both
Default is 0.

Note: Servers should accept the domain from the dialog for the Refer-To and Referred-By headers after the SIP standards, however for some reasons a few popular server would reject the REFER request if they found some other domain (such as proxy server address) and thus this settings defaults to 3 for compatibility reasons.

checksubscriptionstate

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

Default is -1

subscribefortransfer

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

Default is -1

enabledirectcalls

(int)
Specify whether to enable direct call to SIP URI (peer to peer or via server)
0=no (so you should use only the username part of the SIP URI to make calls. set the domain part as the sipserveraddress parameter)
1=check IP in URI (will recognize full SIP URI if the domain part can be resolved to a valid IP)
2=always check (so you can make calls to full SIP URI without a SIP server to be set)
3=crossdomain (so you can register to domain A and make direct call to domain B)

Default is 1

checksrvrecords

(int)
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 records first. If fails then lookup srv record (because mostly the srv record is not set anyway)
2: lookup SRV records first for VoIP server address. If fails then lookup A record (RFC compliant)
3: always lookup SRV records first. If fails then lookup A records.
4: check also without the _sip._udp. prefix
If the SRV or A lookup returns multiple records, than SIP UA will failover to the next server on connection failures taking in consideration both record priority, weight and previous failures/successes.
Default value is -1.

dnslookup

(int)
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.

enableaudiostreams

(int)
You can disable audio playback and/or recording with this option
0=disable all
1=disable recording
2=disable playback
3=enable all
Default value is 3.

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

(int)
Generate ringtone for incoming and outgoing calls.
0=no (you can generate ringtone also by using the Java 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, java softphone will stop the local ringtone playback immediately and starts to play the received ringtone or announcement)
Default is 2.

Note: you might set the earlymedia parameter to 5 in case if you wish to hear ringtone or any announcement from your server before call connect.

ringtone

(string)
Specify a ringtone sound file to be used. (Only the file name, not the full path. Copy the file near JVoIP.jar). If not specified, then JVoIP 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).

Possible values:

- Empty/default: will try to load the embedded ringtone or from the ring.wav near the app or in path
- File name (for example “ring.wav”): will try to load the specified file from near the app or from the java path (and failover to embedded ring if not exists)
- Full path (for example “C:\webphone\myring.wav”): will try to load the ringtone from the specified path first (and failover to embedded ring if not exists)

Default value is empty.

ringincall

(int)
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

playdisc

(int)
Play a disconnect tone on call reject.
0=No
1=Auto (if there was no early media with possible disconnect reason announcement)
2=Always
3=Also for normal disconnect on connected call
Default value is 1

checkvolumesettings

(int)
Check if audio device is muted or volume settings are too low (and un-mute and/or increase volume if necessary).
0: no
1: at first run
2: always
Default value is 1

focusaudio

(int)
Specify whether to use audio “focus”. (Auto decrease the volume of other apps).
This will use VoIP optimizations on windows (WAVE_MAPPED_DEFAULT_COMMUNICATION_DEVICE) and also will enable audio ducking (auto lowering the volume for other processes while JVoIP is in call).

Possible values:

0=auto (will default to yes on windows)
1=no
2=yes
Default value: 0

Note: This new setting will deprecate the old usecommdevice parameter, but usecommdevice can be still used as-is

volumein

(int)
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.
Volume levels above 70% might result in distorted sound.
You can also change the volume from OS system level volume controls.

volumeout

(int)
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.
Volume levels above 70% might result in distorted sound.
You can also change the volume from OS system level volume controls.

volumering

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

beeponconnect

(int)
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

(int)
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.

Note: many sound devices might convert mono sources to stereo by default

plc

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

vad

(int)
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.

Notes:

- The vad parameter is automatically set to 4 by default 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:
  0=no,1=auto (default),2=detect no mic audio,3=send statistics, 4=send also when changed. See the VAD notification and API_VAD for more details.
- If you wish to force vad (usually not required), then you might set the forcevad parameter accordingly:
  0: no (default), 1: force also if conference, 2: force also if rtp muted/holded, 3: force always
- If you want to disable audio related notifications (microphone warning on no signal detected) then set the enablenomicvoicewarning parameter to 0
- More details here

rtpstat

(int)
Enable/disable RTP statistics by triggering RTPSTAT notifications.
Possible values:
-1: auto (will trigger RTPSTAT events in every 6-7 seconds and more frequently at the beginning of the calls)
0: disabled
Positive value: seconds to generate RTPSTAT events.

Default is 0 (disabled)
More details here.

aec

(int)
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: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip
Note: the aec decision might be overwritten by the aectype parameter.

aec2

(int)
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: the aec decision might be overwritten by the aectype parameter.

aectype

(string)
AEC algorithm(s) to use.
One or more of the following strings separated by comma.
auto: will select automatically based on circumstances (CPU power, device capabilities, network)
none: disable aec
software: software aec (requires extra CPU processing)
hardware: hardware aec capabilities (not supported on all platforms)
fast: a fast software aec implementation
volume: this will just decrease the volume when speech detected from other end (using VAD)
Default is auto

Note:
It is recommended to leave both the aec and aectype values with its default values.
To force full echo cancellation in all circumstances you might set the aec to "2", aec2 to 2 and the aectype to "software,hardware,fast"
To completely disable aec you might set the aec to "0", aec2 to "0" and the aectype to "none" (for example when you know that you will have only one way audio such as IVR calls)

denoise

(int)
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: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip

silencesupress

(int)
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

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

use_ilbc

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

Default is 1.

use_speex

(int)
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

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

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

use_speexuwb

(int)
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

(int)
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

Note: The opx dll/so/jnilib files have to be placed near the JVoIP.jar for the opus codec to work.

use_opuswb

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

Note: to enable wideband in all circumstances, set the "disablewbforpstn" and "disablewbonmac" parameters to false. The opx dll/so/jnilib files have to be placed near the webhone.jar for the opus codec to work.

use_opusswb

(int)
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

Note: this codec might not work on all devices, depending on the audio device and driver capabilities!

use_opusuwb

(int)
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

Note: to enable wideband in all circumstances, set the "disablewbforpstn" and "disablewbonmac" parameters to false. The opx dll/so/jnilib files have to be placed near the webhone.jar for the opus codec to work.

disablewbonmac

(boolean)
Whether to disable wideband codec on mac devices.
Mac OS X has a JVM bug which prevents java 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

(int)
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

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

use_pcmu

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

alwaysallowlowcodec

(int)
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

(int)
Number of payloads in one UDP packet (frames per packet). This will directly influence RTP packet time (packetization interval) and packet size as shown here.

- By default it is set to 0 which means 2 frames for G729 and 1 frame for all other codec.
  This usually means 20 msec rtp packetization interval for all codec's and it is the most optimal setting, compatible with all SIP/media stack implementations.

- In case if you wish to minimize delay, then you might set the codecframecount to 1. This usually will result in 10 msec packetization interval and will increase the required bandwidth by 40% due to high header/data ratio.
- In case if you wish to minimize bandwidth, then you might set the codecframecount to 4.

udptos

(int)

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 (HEX):

- 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 (DEC):

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.

This parameter might work only in preset environments; java JVoIP might not have enough rights to modify the IP headers.
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

(int)
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

(int)
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

(int)
Specify if initial line should be put on hold on transfer.
-1=auto
0=no
1=yes, hold before transfer
2=yes, hold before transfer and reload if needed (on transfer failure)
3=yes, hold on successful transfer init (OK for REFER received or attended call progress or success received; transferred call might not be initiated/connected yet). Will reload if needed (on transfer failure).
Default is -1 (which means 3 for transfertype 5 and 6, otherwise 0)

holdtypeonhold

(int)
Specify call hold type.
Call hold is usually initiated by the API_Hold function and with parameter you can specify which type of call hold do you wish to request.
Possible values:
-2: no
-1: auto (defaults to 2)
0: no
1: reserved (not used)
2: hold (mute microphone. this: a=sendonly, peer: a=recvonly)
3: other party hold (mute speaker. this: a=recvonly, peer: a=sendonly)
4: both in hold (a=inactive)

Default is -1.

Note:
By default the hold will check the previous state. For example if previously it was local hold (2) and you switch to remote hold (3) then actually will switch o both hold.
There is also a holdexplicite parameter which if set to 1, then the hold will be done strictly after the holdtypeonhold parameter, without considering previous state. This way you can also change between local and remote hold without the need to unhold first.

muteonhold

(int)
Specify if call also have to be muted with hold (stop recording/playback and stop the according RTP stream).
Possible values:
-2: no mute,
0: mute in and out
1: mute out (speakers)
2: mute in (microphone)
3: mute in and out (same as 0)
4: mute default
5: according to call hold.

Default is 5.

Usually you should set this either to -2 or 5.

defmute

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

ackforauthrequest

(int)
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

(int)
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.

useragent

(string)
This will overwrite the default User-Agent setting.
Do not set this when used with mizu VoIP servers because the server detects extra capabilities by reading this header.
Default is empty.

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.

normalizenumber

(int)
Normalize (called) numbers by removing .-,;()[]: and space if the string otherwise doesn't contains a-z or A-Z characters (looks like a phone number).
Possible values:
-1: auto (usually defaults to 1 yes, except if our username also contains special characters)
0: no
1: yes
Default is -1.

numrewriterules

(string)
Simple called number prefix rewrite rules.
Although you can rewrite any number from your own code as you wish, you might use this numrewriterules parameter to let the SIP stack rewrite the prefix of the called numbers.
Format (parameters): from;to;minlength;maxlength
If you wish to add more rewrite rules, then you can separate them with comma.
Example:
07;00407;8;12,74;004074
This will rewrite prefix 07 to 00407 if number length is between 8 and 12 characters and will rewrite prefix 74 to 004074 regardless of the number length.

Note: this parameter is same with numpxrewrite

numrewriterulesadv

(string)
Advanced number rewrite rules.
Although you can rewrite any number from your own code as you wish, you might use this numrewriterulesadv parameter to let the SIP stack rewrite numbers.

Parameters can be separated by ; or _P_
Rules can be separated by , or _L_
Rules must begin with the separator character.

Parameters:
1. apply to sessions: 0=outgoing calls, 1=incoming calls, 2=all calls, 10=outgoing all, 11=incoming all, 12=all (Defaults to 0)
2. apply if number: 0=all numbers, 1=start with, 2=ends with, 3=contains, 4=equals, 5=not contains, 6=not start with (Defaults to 1)
3. string (used if needed for the above condition)
4. min number length (Defaults to 7)
5. action: 0=rewrite, 1=rewrite all, 2=remove, 3=add prefix, 4=add suffix / 10=auto answer, 11=ignore, 12=forward, 13=reject (Defaults to 2)
6. rewrite from (used if the above action is 0)
7. string (used if needed for the above action is less then 10)
8. stop further rule processing if this rule match (0/1) (Defaults to 0)

Example rule to insert 00 prefix if not already there for outgoing calls if number length is more than 8 digit:
_P_0_P_6_P_00_P_8_P_3_P_0_P_00_P_0_P__L_

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.

disablesamecall

(int)
This setting can be used to don't allow/block double call the same number.
Possible values:
-1: auto-guess depending on config/usage
0: allow multiple calls to the same destination username/number.
1: reject call attempt with numbers already in call when the previous call was initiated less than 6 seconds ago.
2: reject call attempt with numbers already in call.
3: reject call attempt with numbers already in call regardless if it is incoming or outgoing current call.

Default value is 1

maxsimcalls

(int)
Maximum number of simultaneous calls (channel limit/number of call limit).
Will impose a max concurrent call limit applied for both incoming and outgoing calls.
For example if you set it to 1, then the enduser can't initiate or receive new calls while already in call.
Default value is -1 which means no call limit.

Note: if set to 0 then all calls will be denied.

redialonfail

(int)
Retry the call on failure or no response.

- 0: no
- 1: yes

Default value is 1.

A related setting is the "allowrecall" parameter which can be used for more precise control:

- *0: no*
- *1: normal invite resend (same as redialonfail 0)*
- *2: yes srv record and other important retry*
- *3: always when malfunction is detected such as no answer or no incoming RTP (same as redialonfail 1)*

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.

callforwardonnoanswer

(String)

Forward incoming calls to this number if not accepted or rejected within 15 seconds. You can modify this default 15 second timeout with the callforwardonnoanswertimeout setting.
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 softswitch 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: Auto answer 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

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

calltimeout

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

startsipstack

(int)
Automatically start the sipstack after a specified time.
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

You can set to 0 if there is less change that JVoIP will be used once the users will open the app or webpage hosting JVoIP.
You can set to 1 or higher if there is a high probability that the user will use JVoIP to make calls (this will shorten the setup time for the first call).
You can also start the SIP stack explicitly by using the API Start function.

timer

(int)
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

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

mediatimeout

(int)
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).
At the beginning of the calls, the half of the mediatimeout value is applied (2.5 minute by default if there was no incoming audio at all).

mediatimeout_notify

(int)
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

(int)
RTP stream keep-alive packet send interval in milliseconds.
This is useful if your PBX has an RTP timeout setting to prevent disconnects when the java softphone 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 SIP server is malfunctioning when no RTP is received (such as dropping the call on media timeout).
Default value is false.

sendrtponfailed

(int)
Specify if to send RTP packets if unable to open the recording device.
Possible values:
0: No
1: at the beginning (to open the NAT for incoming audio)
2: yes always
3: must
Default is 1.

sendrtpondisabled

(int)
Specify if to send RTP packets if useaudiodevicerecord is set to false.
Possible values:
0: No
1: at the beginning (to open the NAT for incoming audio)
2: yes always
3: must
Default is 3.

discmode

(int)
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

(int)
Maximum time in milliseconds to wait for unregistration when the API_Unregister is called or the java sip stack is closed.
If set to 0 that an unregister message is sent (REGISTER with Expires set to 0) but JVoIP 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.

clearcredentialsonunreg

(int)
Specify whether user login details (server/username/password settings) have to be cleared on unregister or not.
Possible values:
-1: auto
0: no
1: yes
Default: -1

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

(int)
Some IP-PBX 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 the 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.
This is usually the full name of the enduser such as “John Smith”.
Default is empty (the “username” field will be displayed for the peers)
More details here.

pwdencrypted

(int)
Specify if you will supply encrypted passwords via parameters or via the Java 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 here.

voicerecording

(int)
0=no (default)
1=local
2=remote ftp or http upload
3=both

If set to 2 or 3 then either the voicerecftp_addr or the http_addr parameter have to be set.
Local recorded files are placed in the mwphonedata folder which is usually created near the JVoIP.jar if it has permission to its own folder or otherwise to another location such as the user home directory.
The voice recording can be also toggled on/off at runtime with the API_VoiceRecord function.

voicerecfilename

(int)
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

(int)
Recorded file compression.
0: PCM wave stereo files with separate channels for in/our (default)
1: raw gsm. (two 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. Backups here.)
2: ogg/vorbis (optional, on request; module not included by default; notify Mizutech to include ogg/vorbis support in your build if you need this option)
3: mp3 (Make sure that lame is near the jar or in the path. It can be downloaded from here for windows or use your package manager on linux)

voicerecordingbuff

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

Default is -1.

syncvoicerec

(int)
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

Note: you should set this to 0 if you have one-way audio calls such as IVR calls.

uploadretry

(int)
Specify whether the file upload should be retried on failure if the voicerecording parameter is set to 2 or 3.
0: no
1: once
2: until success
Default: 1
(Old parameter now was ftpretry which can be still used but now applies also for http uploads)

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

(int)
Configurations and statistics are stored in a local file to be reused in next sessions.
This is not critical and the Java VoIP client 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.

maxlogsize

(int)
Specify maximum log file size in bytes.
Default is 31457280 (which means 30 MB).

Note:
This is just an upper hard limit and it is relevant only if you run JVoIP for days without restart. Otherwise, it is unlikely that you will reach it since the logs are cleared with each restart (as specified with the above deloldlogs parameter). Under normal circumstances the log file is usually below 10 MB with loglevel 5 and below 100 KB with loglevel 1.

resetsettings

(boolean)
Set to true to clear all previously stored or cached settings.
Should be passed as command line parameter only.
Default is false.

signalingport

(int)
Specify local SIP signaling port to use.
Default is 0 (which means that a stable port will be selected randomly at the first usage and will be kept the same)

Note: this is not the port of your server where the messages should be sent. This is the local port for sip user agent and it doesn't have to be 5060.
For more details see here.

rtpport

(int)
Specify local RTP port base.
Default is 0 (which means signalingport + 2)
Note: If not specified, then VoIP SDK will choose signalingport + 2 which is then remembered at the first successful call and reused next time (stable rtp port). If there are multiple simultaneous calls then it will choose the next even number.
The RTCP port for each call will be the rtpport+1.
For more details see here.

incrtpport

(int)
Increment RTP port for each call by this value.
Might be needed only with some misbehaving routers. If not set, then JVoIP will try to use the same RTP ports for all calls if available (if can bind to it, otherwise will try the next even number).
Default is 0.

Note: You will still have a different RTP port for each simultaneous calls even if this is set to 0.

bindip

(String)
Specify local network interface IP address for the sockets to bind to.
This parameter might be used only on devices with multiple local IP addresses to force the specified IP.
This parameter should be used only with local private IP.
Default is empty (by default it doesn't bind to any IP and it is up to the OS routing table from which IP the packets are sent)

Note: JVoIP by default will auto detect the "best" IP to be used and this parameter should be used only in very specific circumstances.

localip

(String)
Specify local IP address to be used for the SIP signaling.
This parameter might be used only on devices with multiple ethernet interface to force the specified IP.
This parameter should be used only with static IP (if the device IP doesn't change dynamically from DHCP)
Default is empty (auto-detect best interface to be used)

Note: JVoIP by default will auto detect the "best" IP to be used and this parameter should be used only in very specific circumstances.

fawlocalip

(String)
Specify local subnet preference.
For example if the device where JVoIP is running might have two separate IP (such as 192.168.1.5 and 10.0.0.5) then you might set the fawlocalip to 192 to always prefer the 192.x.x.x subnet.
This parameter might be used only on devices running on a known environment (local LAN) in case if you wish to suggest the subnet to be used.
Default is empty (auto-detect best subnet to be used)

Note: JVoIP by default will auto detect the "best" IP to be used and this parameter should be used only in very specific circumstances.

bindtolocalip

(int)
Specify if sockets must bind to the configured (localip parameter) or to the auto detected local ip.
Possible values:
-1: auto (guess)
0: no
1: yes
Default is 0.

Note:

- *This setting can applied even if the localip parameter is not explicitly set (auto detected best local ip to use), but it doesn't make sense if bindip is set.*

- *The disadvantage of setting to 1/yes is that the SIP stack might not tolerate so easy the local IP change (in case if running on a dynamic IP assigned DHCP)*
- *The advantage of setting to 1/yes is that the packets will be always sent from the same address specified in the signaling (some SIP server might be confused if receives packets from a different address then the address specified in the signaling)*
- *The -1/auto will bind to local IP if detected or configured local IP is found locally assigned and it is in the same subnet with the SIP server*
- *JVoIP by default will auto detect the "best" IP to be used and this parameter should be used only in very specific circumstances.*

jittersize

(int)
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

Increase the jittersize if audio is breaking up or decrease do minimize delay. The default 3 is a good compromise between quality and latency.

maxjitterpackets

(int)
You can limit the jitter buffer size with this setting.
This is an hard limitation for the jitter algorithm and usually should not be set.
With the jittersize left as default (3) the maximum buffered packet count is limited to around 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 70 (which basically means no limitation)

allowspeedup

(int)
Specify whether to enable audio playback speedup on high queue size.
(Instead of dropping packets, it might attempt to speed up the playback rate for a short time until queue size drops below threshold)
Possible values:
-1: Auto Guess (default)
0: No (might drop packets instead of speedup)
1: Yes (might speedup instead of packet drop)

In VoIP it is very common that the call setup/audio device open might take some time (10-400 milliseconds in case of JVoIP, depending on mediaench/device capabilities/audio hardware/driver).
Most SIP clients will simply drop audio packets arrived during this time (when their jitter buffer or other queue is full) which results in a small audio loss.
However JVoIP might just speedup the playback for a while to catch-up. This might result in a little audio distortion but will prevent any audio loss in such situation.

allowfirstdrop

(int)
Specify whether to enable drop of RTP packets on media start.
Sometime the other end might start sending RTP before the media is opened at JVoIP side resulting in accumulation of packets.
Dropping the first few packets will make the playback of the rest more smooth, but might result in some loss of audible media (usually up to 100 milliseconds)
Possible values: -1: Auto Guess (default), 0: No, 1: Yes

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

(int)

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 5 means a full log including SIP signaling. Higher log levels should be avoided in production as it can slow down the application.
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 (\mwphonedata\webphonelog.dat near the app or in the java user home directory which depends on the OS/java/browser used)
-java console (if the logtoconsole parameter is set to true)

Default loglevel for demo/trial builds: 5
Default loglevel for licensed builds: 1

More details can be found here.

logtoconsole

(int)
Whether to send tracing to the java console (System.out.print or often referred as STDOUT).
Possible values:

- 0: no (disable log to console)
- 1: auto (depending on loglevel)
- 2: always (always log to stdout)

Default is 1.

canopenlogview

(boolean)
JVoIP might open a log view window if the loglevel is higher the 2.
You can disable this log window by setting this parameter to false.
Default value is true.

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.

logpath

(string)
You might specify the log file path. Environments variables enabled. It might be useful in corporate environments with centralized logging.
Default is empty (which means mwphonedata subfolder or if no write access then in user, tmp or appdata folder)

deloldlogs

(boolean)
Specify whether you wish to delete the old log files.
0: no (new logs will be appended)
1: yes (the previous log file will be kept, older files will be deleted)
3: delete old log
Default: 1

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.

keepaliveival

(int)

NAT keep-alive interval in milliseconds which is usually sent from register endpoints.
Possible values:

- -1: auto (this will default usually to 28 seconds on UDP and 600 seconds on TCP, but can be influenced by many factors)
- 0: disable
- 1-3000: invalid
- 3000 ore more: milliseconds to send keep-alive packets.

Values below 20000 (20 sec) should not be used.
Default value is -1.

recaudiobuffers

(int)
Number of buffers used for audio recording.
Default is 7.

recaudiomode

(int)
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.

maxlines

(int)
Maximum port number from 1 to 4. When set to 1, multiline functionality will be disabled (but JVoIP can have unlimited simultaneous calls).
If you would like to reject all incoming calls if JVoIP is already in a call, then use the "rejectonbusy" parameter instead of setting the maxlines to 1.
Default value is 4.

httpsessiontimeout

(int)
Maximum session time in minutes.
Used only when the VoIP SDK is used as an applet and controlled from java script to avoid situations when the java applet is still running but the user http session is already expired.
You must call the API_HTTPKeepAlive() periodically (for example in every 20 minute) to avoid the timer expiry.
Default value is 60 minute.

singleinstance

(boolean)
If set to true, it will allow only a single JVoIP app instance.
More exactly it will always allow to run the last one, killing the previous once if any.
Default is false

exitmethod

(int)
Controls how the java application or library will cleanup and exit.
Possible values:
-1: auto (default; currently it is the same as 2)
0: do nothing
1: just finish
2: finish and cleanup
3: open exiturl in _self
4: open exiturl in _top
5: call system exit
6: open url and call exit

systemexit

(int)
Specify if JVoIP can call the System.exit() method to quit the JVM when stopped.
0: never
1: auto (Yes if running as a standalone app from command line. No if used as a library)
2: Yes
Default is: 1

fastexit

(int)
Set cleanup speed and wait times.
0: slow but will more care about un-registration and cleanup
1: fast
2: very fast with no unregister
Default is: 1

exiturl

(string)
Specify the URL loaded on exit if the exitmethod is 3,4 or 6.
Default is: https://www.mizu-voip.com/F/webphoneexit.htm (an empty page)

wpapilistenport

(int)
API listen port for TCP/HTTP.
By default might be set to 19422, but you should set it explicitly to be sure.

More details here.

wpapiudplistenport

(int)
API listen port for UDP.
By default might be set to 19422, but you should set it explicitly to be sure.
More details here.

wpapiconnectport

(int)
Messages will be sent to this port if UDP socket is used.
Default is: 19421
This will be automatically set to the port from where the webphone received API requests.

More details here.

canuseudpnotifications

(int)
Specify if JVoIP should send old style outband notifications via UDP to 127.0.0.1:wpapiconnectport. This is an old deprecated method.
Possible values:
0: no
1: only if otherwise can't be sent
2: yes auto
other: must (value treated as local port number)

Default is 1.

More details here.

webphonetojs

(string)
Java script function to be called for the notifications.
Default value is “webphonetojs”

More details here.

jsfunctionpath

(string)
Only if used as an applet.
If your webphonetojs is embedded in other html elements, then you can give the path here. Example: document,externform,innerform2.
By default it is an empty string. This means that Webphonetojs must be placed on the top level (after <body> for example)

More details here.

events

(int)
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.

Old parameter name was javascriptevents, which is deprecated now (but still usable)

jsscripstats

(int)
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)

polling

(int)
To get the event notification from voip SDK to your application, you can use one of the following methods:

- polling with API_GetNotifications
- webhonetojs (JavaScript only)
- socket (notifications sent via UDP or TCP)

Specify which method to use with this parameter.
-1=auto/on demand (will turn to 3 on first API_GetNotifications; will turn to 0 on first socket receive)
0=don’t use polling (for socket or webphonejs)
1=use polling
2=use polling or socket or webphonejs
3=use polling only (webhonetojs and socket will not work)
Default is -1

Appearance Parameters

The JVoIP has a simple built-in user interface which can be convenient if you launch it as a standalone application (This is not used when you use it as a library or launch as a command line application).

These settings below can be used to control how JVoIP user interface (if any) will look like.
For more customization you can write your own user interface and use the API to control the Java SIP client library after your needs.

applet_size_width, applet_size_height

(int)
Should be used only if you use VoIP SDK as an applet embedded in a website.
The size of the space occupied by JVoIP can vary depending on the other parameters.

-if the compact parameter is set to false, than you should set applet_size_width to 300 and applet_size_height to 330.
-If the compact parameter is set to true, than you should set applet_size_width to 240 and applet_size_height to 50.
You can run JVoIP in hidden mode, when all parameters are passed from server side scripts. In this way you can set applet_size_width and applet_size_height to 1.

compact

(boolean)
False: JVoIP will be shown in its full size with username, password input box and dial pad
True: JVoIP will have only a Hangup/Call button and a call status indicator. In this mode the username, password and callto parameters are already set from parameters, so when JVoIP is launched it immediately starts dialing the requested number.
Default value is false.

Usually when this parameter is true, than the "call" is also set true.
Usually when this parameter is false, than the "call" is also set false.

multilinegui

(boolean)
Multiple lines enable the user interface to handle more than one call in the same time. (This doesn't mean multiple server accounts/registrations. If you need to use JVoIP with multiple VoIP servers at the same time, then just launch more instances)
Set to false to hide line buttons. (The phone will still be able to handle multiple calls automatically)
You can restrict the available virtual lines with the "maxlines" parameter.
When set to true, you might also have to set the "hasvolume" to 1 or 2 and the automute or autohold parameters described in this document.
Default is false.

lookandfeel

(string)
Controls the basic design settings. The following values are defined:

- mizu (on request)
- metal
- windows
- mac
- motif
- platform
- system

Default value is null (system specific design is loaded)

colors

(int)
With these parameters you can customize the colors on JVoIP.
Default value is empty.

The following parameters are defined:

- boxbgcolor
- color_background
- color_label_foreground
- color_edit_background
- color_edit_foreground
- color_buton_background
- color_buton_foreground
- color_buton_dial_background
- color_buton_dial_foreground
- color_other_background

There are 3 ways to specify the color parameter:

- integer number: This number represents an opaque sRGB color with the specified combined RGB value consisting of the red component in bits 16-23, the green component in bits 8-15, and the blue component in bits 0-7
- hex number prefixed with #: representation of the color as a 24-bit integer (htmlcolor)
- the name of the color: the following values are defined: black,blue,cyan,darkgray,gray,gren,lightgray,magneta,orange,pink,red, white and yellow

language

(string)

The following languages are built-in:

0. en: english (default)
1. ru: russian
2. hu: hungarian
3. ro: romanian
4. de: deutsch
5. it: italian
6. es: spanish
7. tr: turkish
8. pr: portugheze
9. jp: japanese
10. fr: french

The language parameter can be specified as:

- Language name (Ex: language="Spanish")
- The 2 char abverbiation (Ex: language="es")
- Number (Ex: language=6)

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 (0=all,1=auto guess,2=don't translate js api).
Other languages can be added on your request or just translate your user interface.

You might also set the **locale** parameter to an ISO 639 alpha-2 or alpha-3 language code, or a language subtag up to 8 characters in length. You might also set the country (ISO 3166 alpha-2 country code or UN M.49 numeric-3 area code) like this: locale=en-US

charset

(string)

Set the character decoding for SIP signaling.
Default is empty (will load the local system default).

Common values are UTF-8 and ISO-8859-1.
Default is UTF-8.

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
https://docs.oracle.com/javase/8/docs/technotes/guides/intl/encoding.doc.html

If you need more granular setting then you might use the following parameters:

- charset_sip: for SIP signaling
- charset_media: for audio device names (conversion required only on Windows because of the ASCII only wave audio win32 API)
- charset_bytes: for other byte array – string conversions

hasconnect

(boolean)
Set to false if you don't need the connect/register button.

hascall

(int)
0: never
1: hangup only
2: always

hasconference

(boolean)
Set to false if you don’t need the conference button and conference features.

hashold

(boolean)
Set to false if you don’t need the hold button

hasmute

(boolean)
Set to false if you don’t need the mute button

hasredial

(boolean)
Set to false if you don’t need the redial button

hasaudio

(boolean)
Set to false if you don’t need the audio button

hasincomingcallpopup

(boolean)
Set to false if you don’t need the popup for the incoming calls.
Note: this will not block the incoming calls; will only hide the default user interface –popup dialog- so most probably you will have to replace it with your own user interface or handle the incoming calls automatically from your application. The old parameter name was “hasincomigcall” and it is still valid.

detectlanpeers

(int)
Auto detect other SIP endpoints on the same LAN (broadcast message) such as colleagues at the network place or family members behind the home wifi.
When other user is found, a NEWUSER event notification is triggered.
Possible values:
0: no (will disable also the listener so other endpoints will not be able to detect it)
1: listen only (will be discoverable)
2: yes find others
Default is 2.

textmessaging

(int)

Specify text messaging mode (IM/chat/SMS/API)

-1: auto guess or ask (default)

0: disable all

1: disable incoming messages and auto guess outgoing mode

2: disable message sending and auto guess incoming mode

3: reserved

4: reserved

5: VoIP SMS

6: VoIP IM/chat (SIP MESSAGE)

Note: the old haschat and chatsms parameters are deprecated now but still supported

hasvolume

(int)

0=no volume controls
1=dynamic (default)
2=vertical
3=horizontal (useful if you disable the multiple lines)
Set to false if you don't need the volume controls button

volumeicons

(int)
0=no
1=text (if hasvolume is set to 3)
2=icons (if hasvolume is set to 2)

displaysipusername

(boolean)
Set to true to display the "Extension" edit box.
Default is false.

When sipusername is set (by parameter, user input or api) then it will be used as the sip username and the username field will be used only for authentification.
Otherwise the "username" field will be used for both.

displaydisplayname

(boolean)
Set to true to display the "display name" input box.
Default is false.

hideusernamepwdinput

(boolean)
Set to true if you wish to hide the username/password input controls. Default value is false.

hasgui

(boolean)
Set to false if you are not using the java user interface. (When a custom skin is used). This is an optional setting.
Default is true.

FAQ

How to get my own VoIP SDK?

1. Have a look at the description and pricing at the bottom of the homepage: https://www.mizu-voip.com/Software/SIPSDK/JavaSIPSDK.aspx
2. Download and try from: https://www.mizu-voip.com/Portals/0/Files/JVoIP.zip
3. Contact Mizutech at webphone@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 JVoIP 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: https://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 webphone@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 JVoIP).

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

You will receive the followings:

- JVoIP itself. This is one single file usually named as "JVoIP.jar" and you will receive your own branded (or white label) build without any demo or trial limitations with lifetime license
- 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 webphone@mizu-voip.com. Please contact us only with JVoIP related or VoIP specific projects.

Is it working with any VoIP servers?

Yes. The VoIP SKD 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, JVoIP should work without any issue.
If you have any incompatibility problem, please contact webphone@mizu-voip.com with a problem description and a detailed log (loglevel set to 5). For more tests please send us your VoIP server address with 3 test accounts.

Is it working with any mobile device?

No. The VoIP SDK works only on devices with support for Java Standard Edition. This means almost all PC/laptop OS and a few linux based phone (not Android).

For Android we have a separate VoIP library with the same API: Android SIP SDK

For Java Mobile Edition we have a separate client application that can be used to initiate calls, callbacks and phone to phone calls or send SMS message through our VoIP server. For other devices (iPhone, Android, Symbian) please check our mobile softphones: https://www.mizu-voip.com/Software/MobileSoftphones.aspx

If java voip framework doesn't start then make sure that java is working correctly: http://www.java.com/en/download/testjava.jsp

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

JVoIP is a standalone library, connecting to your VoIP server directly (or directly to SIP peers), without any dependency on our services or third-party services. With other words: if all our servers will be switched off tomorrow, you will be still able to continue using the library and it also works fine in private networks or without internet access.

However by default JVoIP might take advantage of some free online services provided by Mizutech or third parties to ease the usage and for some extra features. All of these are for your convenience. JVoIP will not "call to home" and will not send any sensitive information to Mizutech servers. Most of these are used only under special circumstances and none of these are critical for functionality; all of them can be turned off or changed. These services will not add any overhead and are implemented with minimal resource usage, usually running from low-priority background threads. The following services might be used:

- Mizutech license service: demo, trial or free versions are verified against the license service to prevent unauthorized usage. This can be turned off by purchasing a license and your final build will not have any DRM or "call to home" functionality and will continue to work even if the entire mizutech network is down.
  Note: this is not used at all (completely removed) in paid/licensed versions
- STUN server: in some circumstances JVoIP might use a random stun server 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.
  Note: you can disable this by setting the fast_stun parameter to 0 or configuring your own fast stun server.
- Alternative IP lookup: in some circumstances the SIP stack might try to auto-detect its public IP address from a randomly selected online service such as mnt.mizu-voip.com, checkip.dyndns.org, wtfismyip.com, icanhazip.com, my-ip.heroku.com, checkip.dyndns.org or ipinfo.io. This is just an extra way to detect the correct external IP which might be useful in some rare circumstances (SIP servers with poor NAT support when STUN is unavailable or bogus so at the first request JVoIP might be already able to present its public address). The failure of this service should not affect the SIP stack functionality at all.

Note: you can disable this by setting the altexternpublicip parameter to 0.

- Network connectivity: if the SIP stack can't connect to your server or network connectivity have been lost, JVoIP might ping some well know addresses such as google.com to see if there is any network connection at all. This "ping" is then used only to clarify the connectivity problem reports and print appropriate logs to ease the troubleshooting (to separate "No network" from other possible issues such as "SIP server is offline").
  Note: these might be used only if your SIP server is public (not checked if on the same LAN)
- Geolocation: at first start JVoIP might try to detect its location by a lookup via a random free service such as ip-api.com, ipinfo.io or www.geoplugin.net from a low priority background thread. This is just for your convenience so on your server side you can easily store the client location as this information is just sent by X-Country SIP header. By default this is turned off on private networks.
  Note: this feature has nothing to do with the SIP core functionality and it can be disabled by setting the geolocation parameter to 0.
- Resources: In some circumstances JVoIP 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. This happens only if these resources can't be found locally, probably due to an incorrect deploy.
  Note: you can disable this by correctly deploying the mediaench libraries.

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 to register?

SIP registration means connecting to your SIP server and authenticating. From this procedure your SIP server will learn your application address and can route incoming calls to your application (or other sessions, such as chat, presence, voicemail, etc).

The SIP stack auto-start behavior can be altered with the startsipstack setting or you can use the API_Start function to launch the instance explicitly.
When started, the SIP stack by default (unless you set the register parameter to 0) will automatically register to your SIP server if you configured the SIP account details (serveraddress, username, password and any other parameters that might be required such as the proxyaddress and sipusername).

Otherwise you might disable auto-start (startsipstack) and/or the auto register (register), pass the above parameters dynamically from your code and use the API_Start and/or the API_Register function to initiate the start/connect/register procedure.

The registration success or failure can be obtained from the STATUS notifications.

- Proceeding notification strings:
  - Register…
  - Registering… (or "Register...")
  - Register Failed
- Success notification strings:
  - Registered
  - Registered.
- Failure notification strings:
  - Connection lost
  - No network
  - Server address unreachable
  - No response from server
  - Server lost
  - Authentication failed
  - Rejected by server
  - Register rejected
  - Register expired
  - Register failed

The registered state can be requested with the API_IsRegistered or API_IsRegisteredEx API.
The failure reason can be also obtained (instead of the above STATUS strings) by using the API_GetRegFailReason API.

Check this FAQ point if you are having problems with connect/register.

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
-If you need better control for the separate accounts then you should just launch JVoIP multiple times (multiple instances) with different parameters

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 API (API_Register) or just let the user to fill in the username/password fields and click on the "Connect" button. If JVoIP is registered, then it can already accept incoming calls (it will do it automatically, or you can handle incoming calls from your application, or you can entirely disable incoming calls)
- Now you can make outgoing calls in the following ways:
    - Automatically with JVoIP start. For this you will have to preset the username/password/autocall and callto parameter. Then JVoIP 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 (if you are using the built-in GUI)
    - 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.

How to handle incoming calls?

To be able to receive incoming calls, make sure that you are registered to your SIP server first. Then you can easily test for example by using any third party softphone and make calls to your application SIP username (or extension id or full URI, as required by your server).

In your project all you have to do is to watch for incoming ringing STATUS notifications and from there you can show any user interface as you wish or handle the call from your code.
(You can find the possible strings that can be received at the "Notifications" chapter in the documentation)

To catch incoming calls, you have to look (string parse) for STATUS messages with "Ringing" text (second parameter is the state text) and the endpointtype (forth parameter) set to 2 (means incoming).
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]

JVoIP is capable for automatic line management so if you don't wish to handle lines explicitly in some specific way, then you can ignore all notifications, except those where the line (first parameter) is -1 (the global state).
If you wish to auto accept all incoming calls, you can set the autoaccept parameter to true. Otherwise use the API_Accept to connect the call (or the API_Reject to disconnect).

Check this FAQ point if you are having problems receiving incoming calls.

ERROR and WARNING messages in the log

If you set JVoIP 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 (webphone@mizu-voip.com) in a text file attachment.

JVoIP is not loading/starting

If you see a white page or just a java error on your page that usually means wrong parameters. Please inspect your code and if the problem still persist you should check the java console from your OS.

Make sure that java is working correctly: http://www.java.com/en/download/testjava.jsp

Can't connect to SIP server

If the SDK cannot connect or cannot register to your SIP server, you should verify the followings:

- You have set your SIP server address:port correctly

- Make sure that you are using a SIP username/password valid on your SIP server
- Make sure that the autostart parameter is true and the register parameter is 1 or 2. Otherwise use the API_Start() and/or API_Register() functions explicitly.
- Make a test from a regular SIP client such as mizu softphone or x-lite from the same device (if these also doesn't work, then there is some fundamental problem on your server not related to our library or your device firewall or network connection is too restrictive)
- Check if some firewall, NAT or router blocks your device or process or the SIP signaling
- Check the logs
- If there are no any answer for the REGISTER requests, turn on your server logs and look for the followings:
    - The REGISTER reaches your server?
    - Is there any response triggered (or some error)?
    - If answer is triggered, is it sent to the correct address? (it should be sent to the exact same address from where the server received the REGISTER request)
    - The answer packet reach the client PC? You can use Wireshark to see the packets at network level.
- Send us a detailed client side log if still doesn't work with loglevel set to 5 (from the browser console or from softphone skin help menu)

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).

JVoIP by default doesn't allow cross-domain calls. For example if you are registered as a@A.com (to A domain) then you will not be able to make direct calls to b@B.com (to B domain). Contact mizutech support to remove this limitation.

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 3600 second (1 hour) and you are using the java script API then please 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 for Java. In these circumstances only one audio stream can be used (JVoIP should be able to handle this automatically with default settings)
- Some OS/driver/hardware configurations might not support wideband audio (in these circumstances java voip component 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)
- The VoIP SDK is targeting Java SE platforms only. For android you should use the AJVoIP Android SIP library which is very similar to JVoIP. You might also check the cross platform webphone or the mobile softphone's.
- The demo version doesn't run in headless mode like Linux with no X-Window installed. However we do have also a headless mode build available. Just contact us and we will send the headless version on your request.

Supported programming language

The JVoIP library can be integrated with your application and used by any programming languages which is capable to emit JVM bytecode or runs or Java Virtual Machine.

This includes Java, Kotlin, Clojure, Scala, Groovy, JRuby, Jython and many more.

MAC related issues

If the java (JVM or the browser) is crashing under MAC at the start or end of the calls, please set the "cancloseaudioline" parameter to 3. You might also set the "singleaudiostream" to 5.

If JVoIP doesn't load at all on MAC, then you should check this link.

Linux related issues

Trying to launch the JVoIP GUI might thrown exception: AWTError: Assistive Technology not found: org.GNOME.Accessibility.AtkWrapper.

Solution: (re) install full JRE or uncomment the "assistive_technologies=org.GNOME.Accessibility.AtkWrapper" line in the in /etc/java-XX/accessibility.properties file.

JVoIP doesn't start from headless/console:

The default JVoIP requires X-Window or other GUI support.

Solution: ask mizutech for the headless version. For more details see here.

Audio doesn't work or can't open or read the audio device / driver issue:

Some linux audio drivers allow only one audio stream to be opened which might cause audio issues in some circumstances.

Solution: change audio driver to oss/alsa/pulse (other than your current one). Other workarounds: Change JVM (OpenJDK, Oracle, others); Change browser if you are using the JS API.

Missing audio libraries / JRE issue:

Some JRE might not include audio libraries.

Solution: switch to Oracle JRE (or other JRE with audio support).

Headless Java installs might not include audio libraries. For example you might get "no icedtea-sound in java.library.path" exceptions.

Solution: use a full JRE (or other JRE with audio support such as Oracle).

Audio reopen problems:

In case if JVoIP can't open the audio device or other apps can't open the audio after JVoIP have been used it, you should set the cancloseaudioline parameter to 1 and/or the singleaudiostream parameter to 4.

cancloseaudioline: 0=default auto,1=yes,2=no,3=never on mac,4=never

singleaudiostream: 0=no,1=auto guess, 2=one playback only,3=one recorder only (default),4=one recorder and one playback only,5=same as 4 but no tests

No full-duplex audio support by the device or driver:

Some linux devices doesn't have full duplex support.

Solution:

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.

Audio quality is very bad / cluttering:

Solution: Increase the jitter buffer value by setting the jittersize parameter to 5 or 6.

Why I see RTP warning in my server log

JVoIP 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 JVoIP 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 JVoIP NAT traversal capability)

Media statistics

Media statistics means RTP/RTCP level details, which can be used to analyze call quality or to display a call quality indicator.

- Basic details: are received with the STATUS notifications such as total rtpsent, rtprec,rtploss, rtplosspercet and server statistics if reported. These and more is also received with the RTP LOG notification.
- Quality reports: can be sent as RTPSTAT notifications if enabled by the rtpstat parameter.
  Alternatively it can be requested with the API_RTPStat function call.

- Logs: You can also see more details in the logs such as total rtp statistics reports for calls longer then 7 seconds if the loglevel is at least 3 as EVENT, rtp stat: sent X rec X loss X X% or the RTP notification. If you set the "loglevel" parameter to at least "5" than the important rtp and media related events are also stored in the logs. You might search for "call details" at the log which also includes media related reports after each call.

Voice activity detection

JVoIP has built-in VAD (voice activity detection) algorithm included.
The talking/silence state of both the local user and the remote peer can be reported.
There are two ways to get VAD details: automatic reports (VAD notifications) or using (polling) the API_VAD function.
To enable VAD reports, set the vad parameter to 4.
Once this is set, you can receive the VAD state either by calling the API_VAD function or by the VAD notifications if the vadstat parameter is set to 3 or 4.
For the notifications interval you might also adjust the vadstat_ival parameter after your needs (default is 3000 in milliseconds, which means a VAD report in every 3 seconds).
If the vadstat is set to 3 then it will report VAD state at periodically. If set to 4 then it will more quickly on state change from silence to speaking or inverse.
To receive VAD state per line, set the vadbyline parameter accordingly: 0: global only (default), 1: global and receiver by line.

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 Java VoIP toolkit 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 JVoIP behind NAT. If your server doesn't have NAT support then you should set these settings to 2. In this case JVoIP 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 or 3 (2 is recommended)

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

Ports and firewalls

The JVoIP SIP library by defaults works with all common/usual firewalls such as the Windows built-in firewall, most third-party firewalls and most routers with their default firewall settings.

On "common/usual" we mean packet filters allowing all outbound and blocking inbound only if there is no outbound leg with no explicitly blocking rules and application enabled for application-level firewalls such as the Windows default firewall.

If your firewall is more restrictive, such as allowing only specific ports, then you must enable the ports used by JVoIP on your firewall rules settings.

A typical SIP session requires the following ports:

- Server side (SIP server listens on the following ports):

    - SIP signaling: usually UDP or TCP 5060 or TLS 5061 (5060/5061 are the default SIP ports, but some SIP servers might be configured to use different ports)
    - RTP: a random UDP port (usually an RTP port range can be configured by server administrators, such as 30000-40000)
    - RTCP: usually RTP port + 1 if available (for example if RTP port is 31000, then the RTCP will use port 31001)
- Client side (JVoIP will use the following ports):
    - SIP signaling: a stable random port or as specified by the signalingport parameter.
    - RTP: a random UDP port or as specified by the rtpport parameter. For multiple simultaneous calls it will use the next even ports.
    - RTCP: usually RTP port + 1 if available

Firewall settings:

- In case if your firewall blocks based on external port, then you will have to enable the ports used by your server (usually 5060 and the RTP port range)
- In case if your firewall blocks based on internal port, then you will have to specify the signalingport and rtpport ports explicitly for JVoIP and enabled these ports on your firewall. For the RTP port range you should allow 2x the maximum simultaneous calls above the rtpport setting.

Some ISP's in some countries (such as UAE, Egypt, etc) attempt to block most VoIP traffic (forcing users to pay higher costs for the national telecom providers) usually using sophisticated deep packet inspections (DPI) to drop or delay packet streams which looks like VoIP. This kind of packet filtering can't be handled with the standard SIP/RTP protocols even if you use encryption such as TLS/SRTP. In this case we can recommend to setup a VoIP tunnel gateway which will encrypt and obfuscate all VoIP traffic. Support for this encryption/obfuscation is already built into JVoIP.

Performance optimizations

The SDK by default comes with optimal default value but in some circumstances you might need to further optimize for performance.
If you are running the SDK on some ancient device or any device with low-end CPU or low-powered devices such as Raspberry Pi, you can optimize the performance with the following settings:

- Make sure that no other processes on your device are running with abnormally high CPU usage (remove or optimize other processes that are unnecessarily consuming the CPU)
- Prioritize low computational codecs with the following settings:
  use_pcmu=3 use_pcma=2 use_g729=1 use_gsm=2 use_speex=1 use_speexwb=1 use_speexuwb=0 use_opus=1 use_opuswb=1 use_opusuwb=0 use_opusswb=0 use_ilbc=1
- Set the cpuspeed parameter to 1000 (or some other value between 300 and 2000. Default is 8000.)
- Set the loglevel to 1 (High log levels will slow down the SDK. But don't set it to 0)
- Disable extra audio processing by setting the followings to 0: aec, aec2, denoise, agc, silencesupress
- In case if there are only 1 (or 2) CPU cores available for JVoIP, then set the following parameters to 0 to lower the number of threads: audiorecthread, audioplaythread, audioplaythread2
- Increase the audioqueuemaxsize value. Default is 40. Increasing it can help with short CPU spikes and packet bursts but might increase the playback delay
- Additional optimizations:
    - codecframecount: 2 (or even 4; might be incompatible with some bogus servers)
    - voicerecording: 0
    - syncvoicerec: 0
    - vad: 1
    - plc: false
    - rtcp: false
    - aectype: none
    - enablepresence: 0
    - registerinterval: 300
    - keepaliveival: 50
    - timer: 20 (not recommended)
    - timer2: 20 (not recommended)
- In case if you are using the SDK on a server with no audio device, you should to set the useaudiodevicerecord and useaudiodeviceplayback parameters to false.
- Optimize your JVM (for example using a low delay GC) or use a more performant alternative JVM (java virtual machine)
- Optimize or change OS audio system and/or drivers (especially on linux)

It might be possible that one or two of the above already fixes any performance problems on your devices. In that case it is not necessary to change all of these settings.

Server failover/fallback

Multiple SIP servers can be configured to achieve high availability with failover/failback or load balancing.
There are multiple options to configure server failover:

Using SRV DNS records
Configure SRV DNS records for your SIP servers and let the SIP stack choose the server based on your configuration.
In this case you just need to set the JVoIP serveraddress parameter to your domain name.
JVoIP will not only handle the priority and weight correctly (which itself can be used for failover), but if you have multiple servers then it can pick another as a failover server.
This means that first it will try to connect to the "best" server returned from the DNS query and if fails then it will try to connect to another server returned by your SRV DNS.

Multiple A records
JVoIP can also failover if the server or proxy domain resolution returns multiple A records.
If you wish to disable this behavior for some reason, set the checkmultiarecords parameter to 0.

DNS based failover can be disabled by setting the dnscanfailower parameter to 0. Default value is 2 which means at which retry should attempt to failover.

Internal details: with multiple DNS records then SIP stack uses multi-level failover. First it might try in-place with the GetAlternateServerIP method, then it can try on the endpoint level with the "check next srv record" method and then it can try also globally using the "backupserver" method (which is auto set if there are multi DNS results and it was not configured initially)

Using backup SIP server setting
You can also configure a secondary server for JVoIP using the backupserver parameter, especially if you can't or don't wish to use DNS based failover.
If the SIP stack can't connect to your primary server configured with the serveraddress parameter, then it will try the backupserver.

In case if you wish the SIP stack to try the server availability at the startup procedure, then set the autotransportdetect parameter to true. This should be set if you expect frequent failures with your SIP server. If the autotransportdetect is false (default) then JVoIP will try to register to the primary server first (set by the serveraddress parameter) and will failback to the backup server only on register failure.
With other words if you set the autotransportdetect parameter to true, then there is a small additional delay (around 1 second) on startup even if the primary server is online. If the autotransportdetect parameter is false (the default) then there is no any startup delay but the failover will take a bit more if your primary server is offline.

Note:
In addition to these client side methods, you can of course (also) implement server side failover and HA (high availability) as described here (this document is Mizu server specific, but similar methods can be implemented with/in any third-party SIP servers).

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/jittter)
- Hardware: enough CPU power and quality microphone/speaker (try a headset, try on another device)
- (Missing) AEC and denoise

With loglevel 5 you can see RTP details after each call where the "loss" should be below 5%. Search for "call details" in the log for this.

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)
- try to disable the audio enhancements. Set the following parameters to 0: aec, aec2, agc, denoise, usecommdevice

- try to change your audio driver if you are using it on Linux (from oss to alsa or others)
- JVoIP logs (check audio and RTP related log entries)
- wireshark network trace (check missing or duplicated packets; playback the RTP stream from wireshark to see if there is any difference)

In case if you hear distorted/fast-pitch playback during the call or especially in the first few seconds, that might be caused by the allowspeedup setting and you might disable it if you wish by setting it to 0.

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

You can verify this by using this JVoIP: http://www.jsresources.org/apps/JSInfoApplet.html

If you have multiple sound drivers then make sure that the system default is workable or set the device explicitly from JVoIP (with the "Audio" button from the default user interface or using the "API_AudioDevice" function call from java-script)

To make sure that it is a local PC related issue, please try JVoIP also from some other PC.
If you are using Windows 10, make sure that microphone access is not blocked.

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.

Using JVoIP on a server

If you wish to use the library on a server, then make sure to use the headless version if you don't have X Window installed (X11 or other windowing system). This usually happens if you wish to run your command line app from terminal or as a daemon on a linux server.
The headless JVoIP is provided by Mizutech for no extra cost, just ask for it.

Note:

- *You can also run the headless version of JVoIP in a Docker container.*
- *You can partially test the headless mode also from X-Windows by launching it with the -Djava.awt.headless=true VM parameters (for the headless version only)*
- *Usual error message when you are trying to run the normal (full) version on linux without GUI: "java.awt.HeadlessException: No X11 DISPLAY variable was set, but this program performed an operation which requires it"*
- *Headless JRE might not come with audio libraries. The workaround for this is to install the full JRE (you can still run JVoIP in headless mode)*
  More linux related workarounds can be fond here.

In case if you don't wish to receive or send audio or your machine doesn't have audio record or playback device:

- *If the server doesn't have an audio device you might need to set the syncvoicerec parameter to 0, useaudiodevicerecord to false and useaudiodeviceplayback to false. (These are useful if you wish to perform media streaming or voice recording only).*
- *You can also set the defsetmuted parameter to 1 if no playback is required so you will not have to make a separate call to API_Mute (1 means suppressing playback to speaker, 2 means suppress recording and 3 will mute both).*

- *You might also set the mediatimeout parameter to 0 (if it is common to have one way audio such as playback only).*
- *You might also modify the sendrtpondisabled parameter after your needs.*
- *If correct RTP address negotiation is not important then then you can also set the use_fast_stun and use_fast_ice parameters to 0.*
- *If no audio recording and send is required then you might set the aec parameters to 0.*

Error creating UDP sockets

In some specific circumstances the java voip component 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

Possible workarounds:

- Make sure that you enable java on your firewall (first time it will ask the user with default settings)
- Verify your firewall (especially if you are using some third party firewall) and virus scanner.
- Disable ipv6 in your OS or set JVoIP to prefer ipv4 by launching java with the following command: -Djava.net.preferIPv4Stack=true
- Allow more sockets for java sip stack with the following JVM option: -Dsun.net.maxDatagramSockets=60

No ringback tone

Copy the ring.wav to your app folder (near the JVoIP.jar).
You can also use your custom ringtone: any wave file encoded as standard 8 kHz 16 bit mono linear PCM files 128 kbits - 15 kb/sec.
In this case you should also set the ringtone parameter to “ring.wav” or to full path.

If this doesn’t help:
Depending on your softswitch configuration, you might not have ringback tone or early media on call connect.
There are 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 or 4)
- set the “nostopring” parameter to 1
- set the “ringincall” parameter to 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). Make sure that the dll’s (shipped in the mediaench.zip folder) can be downloaded from your server (there are no warnings in the logs regarding aec initialization). You should store the mediaench.dll and mediaenchx64.dll near JVoIP.jar file on your server and enable the dll mime type (or set the “mediaenchext” parameter to “jar” and rename the dlls to jar: mediaench.jar and mediaenchx64.jar). 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).

Mediaenc download: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip

Chat is not working

Make sure that your softswitch has support for IM and it is enabled. The Java SIP client is using the MESSAGE protocol for this from the SIP SIMPLE protocol suite as described in RFC 3428.
Surprisingly most default 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).

Conference is not working

JVoIP can offer conference features even if your server doesn’t have conference support, using it’s local RTP mixer.
In case if conference doesn’t works, check the followings:

- Make sure that your softswitch has support for multiple simultaneous calls for the same user/device.
- Disable all wideband codec to avoid codec renegotiation with peers
  use_speexwb=1 use_speexuwb=1 use_opus=1 use_opuswb=1 use_opusuwb=1 use_opusswb=1 disablewbforpstn=2

Subsequent chat messages are not sent reliably

Set the “separatechatdiag” parameter to 1.

JVoIP doesn’t receive incoming calls

To be able to receive calls, JVoIP must be registered to your server by clicking on the “Connect” button on the user interface (or in case if you don’t display JVoIP GUI than you can use the “register” parameter with supplied username and password)

Once the SIP client 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 JVoIP with use_fast_stun parameter set to 0 and if still not works then try it with 2 or 4.

If the calls are still not coming, please send us a log from JVoIP (set JVoIP 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 JVoIP 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 JVoIP 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. Opus all (enabled low priority 2)
2. speex wideband (enabled low priority 2)
3. G.729 (enabled low priority 2)
4. PCMU (enabled low priority 2)
5. PCMA (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. Opus
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 JVoIP 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
-use_speexuwb=1
-use_opus=1
-use_opuswb=1
-use_opusuwb=1
-use_opusswb=1
-use_ilbc=1
```

For command line:

use_g729=3 use_pcmu=2 use_pcma=1 use_gsm=2 use_speex=1 use_speexwb=1 use_speexuwb=1 use_opus=1 use_opuswb=1 use_opusuwb=1 use_opusswb=1 use_ilbc=1

There is also a prefcodec parameter which can be used to set the highest priority codec. Example: prefcodec=g729.

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 (G.711 uLaw):

```
-use_pcmu=3
-use_pcma=0
-use_g729=0
-use_gsm=0
-use_speex=0
-use_speexwb=0
-use_speexuwb=0
-use_opus=0
-use_opuswb=0
-use_opusuwb=0
-use_opusswb=0
-use_ilbc=0
```

If you wish to force a wide-band coded, then set also the followings:

```
disablewbforpstn=0
disablewbonmac=0
alwaysallowlowcodec=0
```

If you wish to disable call-retry due to codec incompatibilities, set the codecretry parameter to false.

How to disable redial

On call failure the SIP engine might auto-redial in some circumstances with changed capabilities.

To disable this behavior, set the following parameters:

```
hasredial=false
redialonfail=0
allowrecall=0
codecretry=false
callretryonreject=0
```

Disconnect reasons

The disconnect reasons are reported in the following format: code text. (So you have the text after a space)

Code:

Is a SIP disconnect request or answer code including BYE, CANCEL or any SIP response code above 300.

If no disconnect message were received or sent then the code is -1.

Text:

The text is one of the followings:

1. local disconnect reasons (listed below)
2. disconnect reason extracted from SIP Warning or Reason headers
3. response text extracted from the first line of SIP responses (textual representation of the response code)
4. textual representation of the disconnect code

The local disconnect reasons can be one of the followings (extra details might be appended and new disconnect texts might be added in the future):

- notaccepted
- User Hung Up
- bye received
- cancel received
- authentication failed
- endpoint destroy
- no response
- call setup timeout
- ring timeout
- media timeout
- endpoint timeout
- tunneling calltime limit
- call max timer expired
- max call time expired
- max speech time expired
- not acked connection expired
- disconnect on transfer
- transferalways
- transfer timeout
- transfer fail
- transfer done
- transfer terminated
- transfer (other)
- refer received
- cannot start media
- not encrypted
- srtp fail
- srtp init fail
- disc resend
- failed media
- forward
- forwardonbusy
- forwardonnoanswer
- forwardalways
- rejectbusy
- rejectonphonebusy
- call rejected by peer
- rejected by the peer
- rejected
- autoreject
- autoignore
- ignored

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). Caller-ID means the remote username, extension number or real phone number as sent by your SIP server. The SIP server might or might not forward the remote real caller-id or might replace it to any arbitrary value (such as the CLI number associated to the user, the user extension or auth id).

You can use the following parameters to influence the caller id display at the remote end:

-username (user id or caller id)
-sipusername (auth username used for SIP digest authentication)
-displayname (sent in the from/contact headers)

Some VoIP server will suppress the CLI if you are calling to pstn and the number is not a valid DID number or JVoIP account doesn't have a valid DID number assigned (You can buy DID numbers from various providers. This is up to your SIP server configuration and has nothing to do with JVoIP).
The CLI is usually suppressed if you set the caller name to "Anonymous" (hide CLI).

For **incoming** calls the Java softphone will use the caller username, name or display name to display the Caller ID. (SIP From, Contact and Identity fields extracted from the incoming INVITE).
You can also use headers such as preferred-identity to control the Caller ID display.
The Caller-ID is received with the STATUS notifications or you can query it with the API_GetCallerID API. You can also use the API_GetIncomingDisplay API which can return some more details about the caller such as the display name.

Example (fields containing information about the caller name or number highlighted with bold):

INVITE sip:bob@biloxi.com SIP/2.0
Via: SIP/2.0/UDP pc33.atlanta.com;branch=z9hG4brn7wmmo4h
Max-Forwards: 35
To: Bob <sip:bob@biloxi.com>

From: Alice <sip:**alice**@atlanta.com>;tag=871822
Call-ID: a82c41f1b65
CSeq: 1 INVITE
Contact: <sip:**alice**@pc33.atlanta.com>
P-Asserted-Identity: "**Cullen Alice**" <sip:**alice**@atlanta.com>
P-Asserted-Identity: tel:**+14095361002**
Content-Type: application/sdp
Content-Length: 142

...sdp content...

You can learn more here and here.

How to implement IM

IM (Instant message/chat) is well supported by JVoIP. Make sure that your server has support for SIP MESSAGE.

Basic IM:

The core chat functionality can be implemented very easily:

- To send a text message just use the API_SendChat function. Example: API_SendChat(-1, "john", "", "Hi!")
- Incoming text messages can be cached by the CHAT notifications. Example: “CHAT,1,john,Hi!”

Usually you will have to maintain a separate thread with each peer (displaying the conversations on a separate page by peer).

Optional features:

Once the simple chat send/receive is working, you might add some extra functionalities:

- Handle delivery success/failure:
  - You might parse the CHATREPORT notifications to notify the user if the message was delivered successfully or failed.
- Typing notifications:
  - You might use the SendChatIsComposing function to send typing notification to the peer.
  - Also parse the incoming CHATCOMPOSING messages (usually displaying something like “John is typing…”.
- Group chat:
  - If you wish to implement group chat, then you should send/accept the group name which is usually the members separated by | . Example: "emma | john | linda"
  - When sending, use the group parameter of the API_SendChat. Example: API_SendChat(-1, "emma", "emma | john | linda ", "Hi")
  - For receiving, check if the message begins with the "GROUP: xy:" substring where xy is the group.
  - Usually you have to display each group as a separate thread (separate window for each group name). If group name doesn't exists, then use the peername.
- Offline messaging:
  Offline messaging means queuing messages for later deliver when immediate delivery fails.
  Offline messaging is automatically handled by JVoIP unless you set the offlinechat parameter to 0.

Beyond the IM SIP protocol offered by JVoIP, chat is mostly about user interface. Implement a nice GUI with handy features such as threaded messaging, send picture (using the file send API), emoticons and any other features for your users.

How to DTMF

DTMF means Dual-tone multi-frequency signaling and it is commonly used to send touch tones to the server or the other peer. On phone user interface this is usually handled by handling key press or presenting a phone interface to the user who can press the digits to be sent.

Using the SIP protocol, there are multiple ways to send DTMF digits which can be set by the dtmfmode parameter.

If you are using the built-in user interface then you can send DTMF digits by pressing the number buttons during a call and if a dtmf digit is received, then you will see it displayed on the user interface notification area.

If you are using JVoIP as a library then you can send DTMF digits using the API_Dtmf function.

Feedback about the delivery can be obtained by watching for the INFO notifications (you will receive INFO,OK when the message was successfully sent or INFO,ERROR if failed).

On incoming DTMF you will receive a DTMF notification.

Custom INFO messages

You can send and receive custom SIP INFO messages in the SIP signaling as described in RFC 2976.
Note: If you are looking to send or receive DTMF digits, then you should look at the above FAQ point instead.

To send a custom INFO message, use the API_Info function.

Feedback about the delivery (if necessary) can be obtained by watching for the INFO notifications (you will receive INFO,OK when the message was successfully sent or INFO,ERROR if failed).

INFO notifications are also triggered for incoming SIP INFO messages (INFO,REC).

Transfer API usage

The transfer mode can be set with the transfertype parameter.
The call transfer is handled with the SIP REFER method as described in RFC 3515, RFC 5589 and RFC 6665.
With **unattended transfer** using transfertype 1 the transfer will be executed immediately once you call the API_Transfer function (blind transfer; only a SIP REFER is sent).
With **unattended transfer** using transfertype 6 and holdontransfer 1 or 2 the call might be put on hold before transfer and reloaded on transfer fail.
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 might be put on hold by A if holdontransfer is 1 or 2
   A will call to C and connect (consultation call; if call fails, then the call between A and C will be un-hold automatically)
   On 180, 183 or 2xx answer (Session Progress, Ringing or OK) the original call between A and B might be put on hold if holdontransfer is -1 or 3

   A speaking with C
   If call fails, the original call between A and B might be reloaded from hold
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:
   Transfer related notifications (SIP NOTIFY) can be sent between the endpoints once the call transfer is initiated, reporting the transfer state.
   - 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, otherwise call between A and B might be reloaded

Related API's: API_Transfer, API_TransferDialog
Related parameters: transfertype, transfwithreplace, allowreplace, discontransfer, disconincomingrefer, inversetransfer, transferdelay, holdontransfer

Call transfer should be used only after call connect as most servers doesn't support the REFER method before connected. If you wish to transfer (redirect) the call before call connect, then you should use the API_Forward function instead.

How to implement PTT?

Push to talk can be easily implemented by using the API_Hold function.
If somehow call hold is not well supported by your server or the remote peer, then you might use API_MuteEx instead.

Other related functions and parameters which you might use are the followings:
API_IsMuted, API_IsOnHold, API_HoldChange, automute, autohold, holdtypeonhold, muteonhold, defmute, sendrtponmuted

How to deploy the mediaench module

The media enhancements module contains native media processing libraries which can improve JVoIP media quality (sound/audio device handling/codec).
Just copy the mediaench dll's near your JVoIP.jar file. This module is needed if you enable one of the following options: aec, denoise, silencesupress, agc.
Download from here: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip
JVoIP will automatically load the proper library (depending on the OS) at runtime if found. Otherwise it will failback to Java code.

JVoIP will search to load the library in the following paths (in this order):

1. System.loadLibrary (which scans the java.library.path)
2. System.load(def) (which will search app and system folder)
3. System.load(currentpath);
4. load from getCodeBase()
5. load from user.dir
6. download from web
7. failback to java if not found

Multiple lines

Multi-line means the capability to handle more than one call at the same time (multiple channels / multiple concurrent calls).
It is important to note that this functionality can be achieved also by just launching multiple instances. This FAQ is about handling multiple lines in a single instance.

Multiple lines can be managed by the line parameter of the API functions (most functions has a line parameter) and by checking the line parameter of the STATUS and other notifications (most notifications has a line number).

By default you don't need to do anything to have multi-line functionality as this is managed automatically with each new call on the first "free" line. This means that you just need to pass -1 as the line number with any function call and look only the notifications with the line number set to -1 (the global phone state).

If you wish to manage the lines explicitly, then you should pass the desired line numbers with the API calls and also look for the notifications received from the individual lines (such as the line parameter of the STATUS notification). This is necessary for example if you wish to create a user interface where the users can select/change to a specific line (line selector or line buttons).

Multi-line vs Conference
When we refer to "multi-line" we mean the capability to have multiple calls in progress at the same time. This doesn't necessarily means conference calls. You can initiate multiple calls by just using the *API_Call* API multiple times (to initiate calls to more than one user/phone), so you can talk with remote peers independently (all pears will hear you unless you use hold on some lines, but the peers will not hear each-others).
To turn multiple calls into a conference, you need to use the *API_Conf* API (or use the conference button from the softphone.html). When you have multiple peers in a conference, all peers can hear each-other.

Disable multi-line
You can disable multi-line functionality with the following settings:
-set the multilinegui parameter to 0
-set the rejectonbusy setting to true

Other related parameters are the automute and autohold settings.

API
Most API functions has a line parameter which you can set to specify the line number.
Also there are two specific API to set/get the current active line:
-API_SetLine(line); // Will set the current line. Just set it before other API calls and the next API calls will be applied for the selected line
-API_GetLine(); //Will return the current active line number. This should be the line which you have set previously except after incoming and outgoing calls (will automatically switch the active line to a new free line for these if the current active line is already occupied by a call)

The active line is also switched automatically on new outgoing or incoming calls (to the line where the new call is handled).

Channels
The following line numbers are defined:

- -2: all (some API calls can be applied to all lines. For example calling hangup(-2) will disconnect all current calls)
- -1: current line (means the currently selected line or otherwise the "best" line to be used for the respective API)
- 0: undefined (this should not be received/sent for endpoints in call, but might be used for other endpoints such as register endpoints)
- 1: first channel
- 2: second channel
- …
- N: channel number X

Note: If you use the API_SetLine with -2 and -1, it might be remembered only for a short time; after that the API_GetLine will report the real active line or "best" line.

API usage example:

```
API_Call(1,'1111'); //make a call on the first line
API_Call (2, '2222'); //make a second call on the second line
```

```
//setup conference call between all lines
API_Conf(); //interconnect current lines

//put first call on hold
API_Hold (1,true);

//disconnect the second call
API_Hangup(2, true);
```

Note
If your use-case requires multiple simultaneous calls, then you might wish to set the following parameters:

- focusaudio: 1
- aec: 0
- aec2: 0
- agc: 0

Some API might auto-guess the correct line to use if you supply a wrong line. For example if you call API_Hangup(X) and on the line X there is no call, the SIP stack might disconnect the call on another line if any.

You can use the API_LineToCallID and the API_CallIDToLine if you wish to convert between line number and SIP Call-ID.

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).

"Stream" and "streaming" in this context means sending the audio packets to a non-VoIP application (such as your app) in a raw format (and it doesn't meant streaming RTP audio for remote VoIP peers or from/to SIP server).

Streaming 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).
Just launch an UDP server socket to listen on any port and set the same 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 as UDP packets and from there you can process them as you wish (for example audio playback, speech to text processing or feed to other API such google cloud API for any audio processing).

A simple working Java example can be downloaded from here.
If you are using JVoIP with C#, then you might check this code.
In case if you just wish to store the audio and don't need real-time processing, then you should use voicerecording / API_VoiceRecord instead of this streaming.
If you need to stream an audio file to a remote SIP endpoint, just use the API_PlaySound.

Parameters:

sendmedia_type

Specify the media stream format sent by JVoIP to your app:
0: raw wave format (linear PCM) for narrowband (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.
With Google STT we recommend to set the sendmedia_type to 3 and set the codec encoding:LINEAR16 and sampleRateHertz: 16000 for the Google speech API.

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 JVoIP to simplify the audio formats conversion (disable all codec's except PCMU and PCMA).
In case if you wish to receive incoming streams separately for conference calls, set the sendmediain_conf parameter to 1.

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).
For example ffmpeg.

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, AWS 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).*
If you are using JVoIP with C#, then you might check this code.
For AWS you can check this code.

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 JVoIP will emit RTP packets which can be processed as is by such software.

4. **Receive the audio packets with your app:**

You just need to start and UDP server (preferably in a separate thread, listening on the port that you specified as the sendmediain_to and/or sendmediain_from JVoIP parameter) and you will receive the audio in the UDP packets.

In case if you are using Java and wish to work with streams instead of byte buffers, then just convert the received audio byte buffer to stream.
Example: InputStream is = new ByteArrayInputStream(bytearray); //you can convert to OutputStream or other stream types similar way

Example Java code:

```
//Set sendmediain_to and/or sendmediaout_to webphone parameters to any port number. Let it be 50555.

public class AudioReceiver extends Thread {

  private DatagramSocket socket;
  private boolean running;
  private byte[] buf = new byte[1500];

  public AudioReceiver() {

  }

  public void run() {
    try{
      running = true;
      socket = new DatagramSocket(50555); //listen on the same port what you set for sendmediain_to and/or sendmediaout_to

      while(running)
      {
        DatagramPacket packet = new DatagramPacket(buf, buf.length);
        socket.receive(packet);

        InetAddress address = packet.getAddress();
        int port = packet.getPort();
        packet = new DatagramPacket(buf, buf.length, address, port);

        ProcessAudioPacket( packet.getData(),packet.getLength());

      }
      socket.close();
    }
    catch (Throwable e) {
```

```
        e.printStackTrace();
      }

    }

    public void terminate()
    {
      try{
          running = false;
          socket.close();
      }
      catch (Throwable e) {
        e.printStackTrace();
      }
    }

    public void ProcessAudioPacket(byte[] buff, int len)
    {
       //process the audio data here as you wish
    }
}

//create thread somewhere in your code: (new AudioReceiver()).start();
```

A simple working example can be downloaded from here.

5. PCM stream:

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

Example for stream playback using the Java built-in audio:

```
//import required libraries
import java.io.*;
import javax.sound.sampled.*;

//initialize audio
AudioFormat format = new AudioFormat(8000, 16, 1, true, false);  //you need to use 8000 for narrowband input or 16000 for wide-band
DataLine.Info info = new DataLine.Info(SourceDataLine.class, format);
SourceDataLine line = (SourceDataLine)AudioSystem.getLine(info);
line.open(format);
line.start();

//playback (you can pass the UDP packets as-is as received from JVoIP)
//this is usually called many times from your UDP socket read thread or on UDP packet received event
line.write(buffer, 0, length);

//close
line.flush();
line.stop();
line.close();
```

Note:
You must add proper exception handling for the above code (try/catch). Exceptions might occur if there is no suitable playback audio device.
More details about Java audio can be found here.
If you are interested in both streams (in/out) then you will need 2 separate lines (don't just feed both streams to the same line).
The same can be done in a very similar way in any programming language (just search for its audio playback module/interface/functions).

6. Wave files:

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

7. Barge-In:

If you just wish to listen into conversation, then you can use the JVoIP 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.

8. SIP media streaming:

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

Quality tests

With JVoIP you can easily test your VoIP server/carrier call quality. For this a call have to be done between two extensions via your server.
Launch the java sip client app like this:
JVoIP.jar serveraddress=SERVERIP username=USR1 password=PWD1 callto=USR2 sereveraddress2=SERVERADDRESS username2=USR2 password2=PWD2 aqtest=1 loglevel=5
Accept the incoming call and keep it for a while then hangup and check the logs by searching for "aqstat display begin". You will find a line like this:
EVENT,aqstats: avg delay: 51 msec, max delay: 56 msec, loss: 0%, cseqproblems: 0%, expected: 30069, recv: 30069, cseqerr: 0, err: 0
Fields:

- avg delay: total average RTP roundtrip time to server and back in milliseconds (should be only slightly more than ping time to your server)
- max delay: maximum rtp roundtrip time (should be not too bigger than avg delay)
- loss: lost packets percent (should be 0% if you have a good internet connection. Audio quality will drop considerably above 4%)
- cseqproblems: unordered packets and other issues in percent (should be 0%)
- expected: number of expected packets (depends on the test call duration)
- recv: number of received packets (should be the same as expected or slightly less)
- cseqerr: unordered packets and other issues
- err: unrecognized received packets

After this line you will see packet arrival timing statistics like this (wrong if the packet are too spread):
EVENT,recv in 50 msec: 3030 packets
EVENT,recv in 60 msec: 26680 packets
EVENT,recv in 70 msec: 246 packets

Tunneling from behind MS Forefront TMG proxy server

TMG cannot handle streaming well. For this reason JVoIP will automatically switch off the streaming and will use packet by packet mode when used with the Mizu VoIP Tunneling. However the firewall built into the TMG server might disable this also after a long call. The best way for TMG users is to allow TCP port 443 of configure VoIP access as described below:
http://technet.microsoft.com/en-us/library/dd441021.aspx
http://technet.microsoft.com/en-us/library/ee690384.aspx

How to generate and send logs from JVoIP

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.

1) Make sure to **set the loglevel parameter to 5** and then reproduce the problem.
   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)

2) Once you reproduced the problem, you can grab the logs from any of the following locations:
   - if you haven't disabled the GUI then a log window should appear with loglevel 5. Copy its content with the Ctrl+A, Ctrl+C, Ctrl+V key combinations
   - or copy the content of the console if you started the JVoIP from command line
   - or copy the content of the java console
   - or send the generated log file (the *log.dat files from the mwphonedata subfolder)

3) Send the logs to webphone@mizu-voip.com as email file attachment with an exact and detailed description of the issue.
   Make sure to send the whole log from the very beginning, not only the relevant part.
   If possible send step-by-step instructions for the reproduction and one or two SIP accounts valid on your server which can be used by Mizutech support to reproduce the problem.

Resources

VoIP SDK home-page: https://www.mizu-voip.com/Software/SIPSDK/JavaSIPSDK.aspx
Download (demo package): https://www.mizu-voip.com/Portals/0/Files/JVoIP.zip
Test/Example: https://www.mizu-voip.com/Portals/0/Files/JVoIPTest.zip
Media enhancements: https://www.mizu-voip.com/Portals/0/Files/mediaench.zip
Documentation:

- PDF: https://www.mizu-voip.com/Portals/0/Files/JVoIP.pdf
- HTML: https://www.mizu-voip.com/Portals/0/Files/documentation/jvoip/index.html
- WinHelp: https://www.mizu-voip.com/Portals/0/Files/JVoIP.chm

JavaDoc:

- Download: https://www.mizu-voip.com/Portals/0/Files/jvoip_javadoc.zip
- Online: https://www.mizu-voip.com/Portals/0/Files/jvoip_javadoc/index.html

For help, contact webphone@mizu-voip.com