JVoIP Video

SIP video calls with the MizuTech Java SIP Library

Contents

About

This document will explain how to make and receive video calls with JVoIP, describing all the video related details, including the video related parameters, API's and notifications.

- Supported video codec's: VP8, H261, H263, H264
- Experimental/beta: VP9, Theora, MPEG1, MPEG2, MPEG4 (ask us if you need any of these and you run into some problems)
- Recommended codec's: VP8, H264 (set by default to offer and highest priority for accept)

Beside webcam streaming, screen recording/sharing and local streaming is also supported.
A list of other video related functionalities are also supported such as re-invite with video, hold/reload or video device enumeration.
There is no local RTP mixer built in JVoIP for multiline video conference calls as this is usually handled on the server side in SIP networks.

This document describes is available as PDF, CHM, HTML and describes only the video related functionalities of the JVoIP Java SIP library.
For general usage, see the JVoIP main documentation instead.

Usage

Video module
The video module might be enabled by default or you can enable it by setting the video_config to 4 or higer.
If the video module is not already installed, then JVoIP will automatically download it on demand. More details here.

Parameters
You might fine-tune the video related parameters if you wish to change the defaults such as the video_quality, video_codec and video_device.
The video related parameters are listed below.
You can pass these as command line parameters or using the API_SetParameter or API_SetParameters function.
The parameters can be passed during startup or just before a video call (most parameters can be changed at runtime).

API
Use the API_Call funcion with the calltype parameter set to 1 to initiate an outgoing video call.
Use the API_Accept function with the calltype parameter set to 1 to accept a video call.
You can also initiate audio only calls by the API_Call funcion (with the calltype parameter set to 0), and add/remove video later using the API_AddVideo / API_StopVideo functions.
The video related API functions are listed here.

Pseudocode
Making a video call from JVoIP is as simple as this:

```
wobj.SetParameter("video_config", 8); //enable video with auto-accept (optional)
wobj.SetParameter("video_quality", 12); //set HD quality  (optional)
wobj.API_Call(-1,"1234",1); //make a video call to 1234
```

Main Parameters

The most important video related parameters are listed below.

video_config

(int)
Enable/disable SIP video.
This is a simplified configuration for has_video, video_autoaccept, video_autosend and video_sendsdp. (So you don't need to set these if you set the video_config parameter)
Set to 2 to disable video or to 4 to enable video on request.
All possible values:

- -1: auto (automatically activated on API requests)
- 2: never enable
- 4: on request only
- 6: ask (send offer, ask to enable)
- 8: yes (send/receive automatically)
- 10: always (force video send/accept)

Default is -1.

Note:

- *When using the API_Call or API_Accept functions with no calltype or the calltype set to -1, then JVoIP might automatically decide to initiate/accept call with video or audio only.*
- *If you wish to explicitly control the call type, then just set the video_config parameter to 4 and use the API_Call and API_Accept functions with the desired calltype parameter.*
- *Video will be automatically enabled if you make or accept a video call explicitly (by using the calltype parameter)*

video_device

(string)
Specify the name of the video camera device to be used.
If empty or set to "default" then will use the default camera device found.
Valid value: name of the webcam device (string)

video_preview

(number)
Specify if a local video screen is needed before/during video calls.
Possible values:

-1: auto (usually defaults to 4 if not headless and there is a video device)
0: no
1: yes, direct playback before start sending (at setup/ring until connect)
2: yes, local stream during call (after call connect only)
3: yes, always with direct playback first (until call connect) and then local streaming
4: yes, always from local streaming for incoming calls, direct + streaming for outgoing calls

Default is: -1

video_codec

(string)
[Same as “vcodec”]
Specify the video codec to be used as a string with values separated by comma. The followings are the possible values:

- H261
- H263
- H264
- H265
- VP8
- VP9
- mpeg1
- mpeg2
- mpeg4
- Theo (Theora)

By default the H264 and VP8 codec is enabled.
Examples:

- To enable only VP8, set this parameter to "VP8”.
- To enable only VP8 and H263, set this parameter to "VP8,H263”

video_prefcodec

(string)
Set the preferred video codec if more codecs are enabled by the video_codec parameter.
If the codec was not enabled before (with the video_codec parameter) then it will be enabled automatically by video_prefcodec.
This parameter will set the codec priority to 3 and will lower all other codec priority below 3.
Instead of video_codec and video_prefcodec, you might use the use_codec parametes to explicitly set the codec priorities.

video_quality

(number)
Suggest default video quality.
Possible values:

0: auto (this is the default value and will defaults to medium)
2: lowest [qcif] (use this with outdated peers or on very low bandwidth)
    Bitrate: 128
    Resolution: 176x144
    FPS: 5
    Profile: baseline
    Level: 1.1
4: low [qvga] (use this with outdated peers or on very low bandwidth)
    Bitrate: 192
    Resolution: 320x240
    FPS: 10
    Profile: baseline
    Level: 1.2
6: lower [vga] (use this with peers supporting only low quality)

Bitrate: 800
Resolution: 640x480
FPS: 18
Profile: baseline
Level: 2.2

7: medium [720p] (with baseline profile)
Bitrate: 2000
Resolution 1280x720
FPS: 24
Profile: baseline
Level: 4.1

8: medium [720p] (with main profile)
Bitrate: 2000
Resolution: 1280x720
FPS: 24
Profile: main
Level: 4.1

9: medium [720p] (with high profile)
Bitrate: 2000
Resolution: 1280x720
FPS: 24
Profile: main
Level: 4.1

10: higher [uxga] (high quality video)
Bitrate: 3000
Resolution: 1600x1200
FPS: 25
Profile: high
Level: 4.2

12: high [HD] (high quality video; recommended if the peers have support for this)
Bitrate: 5000
Resolution: 1920x1080
FPS: 30
Profile: high
Level: 4.2

15: very-high [2K] (recommended only on high bandwidth networks with very high-end devices)
Bitrate: 8000
Resolution: 2560x1440
FPS: 30
Profile: high
Level: 5.1

18: ultra-high [4K] (recommended only if you have a powerful machine with a 4K input and output device)
Bitrate: 11000
Resolution: 3840x2160
FPS: 30
Profile: high
Level: 6.2

Note:
The bitrate is the upper target here and might go lower mostly depending on the source.
The default video quality is 720p, which might be too high for average devices with low network bandwidth. You might set the lower (6).
The actual video quality depends on many factors, including the other parameters, the codec, peer capabilities, server capabilities, video tool quality and parameters, network quality and hardware.

Advanced Parameters

You should change the below settings only if you fully understand their role and meanings.
Some parameters are meaningful only if you force a specific video codec such as H264.

video_module

Set the video module to be used
(int)

- -1: default
- 0: don't load any module
- 1: old applet (deprecated. The old documentation can be found here)
- 2: built-in ffmpeg (will auto download if not exists and launch on demand)
- 3: external app streaming (more details here)
- 4: RTSP (android specific)

Default is -1 which usually defaults to 2.

video_relay

Specify whether to route or bypass video if the external API is used (with video_module set to 3).
(int)

- 0: don't route the video RTP
- 1: route the video RTP

Default is 1.

If set to 1, then SRTP is also handled by JVoIP.

video_fps

(number)
Specify frame per second value.
Valid values are above 3.
This will overwrite the suggested value by the video_quality parameter but might be capped by the remote peer during the negotiation.
Default is -1 (best match after the video_quality parameter and auto-negotiated).

video_bitrate

(number)
Specify the desired bitrate in kbit/s.
This will overwrite the suggested value by the video_quality parameter but might be capped by the remote peer during the negotiation.
Default is -1 (best match after the video_quality parameter and auto-negotiated).

video_vwidth

(number)
Video width (scale for recording).
By default it is 1280 which is often changed depending on the other parameters (video_quality) and auto negotiated.

H.263 might work only in qcif (176x144) mode with some peers.
Other standard formats: subqcif (128x96), qsif (160x120), qcif (176x144), sif (320x240), cif (352x288) or vga (640x480), 800x600,1024x768,1600x1200

video_vheight

(number)
Video width (scale for recording).
By default it is 720 which is often changed depending on the other parameters (video_quality) and auto negotiated.
To scale the video, usually it is enough if you specify the video_vwidth and the height will be resized automatically after to the source by keeping the aspect ratio.

video_packetizationmode

(string)
Force a packetization-mode to be negotiated by SDP such as “0”, “1” or “2”.
Special values: "-" means always disable, "x" means auto, "d" means not set in offer
For H.264 you should always prefer 1 which should be negotiated by default.
Default is "d" which means don’t send in offer and auto-negotiated if required.

video_profilelevelid

(string)
Force a fixed profile-level-id to be negotiated by SDP such as “42e015” .
Special values: "-" means always disable, "x" means auto, "d" means not set in offer
Default is "x" which means auto-negotiated.

video_profile

(string)
Force a specific H.264 profile (“baseline”, “main”, “high”).
Default is empty which means auto-negotiated.

video_pixelfmt

(string)
Force a specific H.264 pixel format such as “yuv420p”.
Default is empty which means auto-negotiated.

video_level

(string)
Force a specific H.264 level such as “4.1”.
Default is empty which means auto-negotiated.
More details here.

video_sprop

(string)
Force a sprop-parameter-sets be negotiated by SDP.
Special values: "-" means always disable/ignore, "x" means in-band, "d" means not set in offer
Default is "x" which means negotiating for in-band sprop.

video_ffconfig

(string)
Extra config to be negotiated by SDP useful for tools such as ffmpeg, ffplay or vlc.
Special values: "-" means always disable/ignore, "x" means in-band, "d" means not set in offer
Default is "d" which means don’t send in offer and auto-negotiated if required.

video_extraparams

(string)
Any extra parameters to be sent with the SDP by the a=fmtp: videocodec line.

video_startplay

(int)
Specify when to start video playback:
0: on demand only on first incoming video RTP packet

1: when negotiated (on video call connect)

Default is 0.

video_stoponstop

(int)
Specify if to really stop the player/recorder when stopping video streams.
1: stop only the rtp stream
2: stop the player/recorder.

Default is 2.

video_holdonstop

(int)
Specify if to send a hold request on video stop.
0: no (don't send video video hold request. just stop the RTP streaming and/or the player/recorder)
1: yes (send video hold request if required)

Default is 1.

keepvideosdponhold

(int)
Specify if to keep the SDP video part if video is stopped or holded.
-1: auto (default)
0: no (remove video from SDP and set the video port to 0: m=video 0 …)
1: yes (keep the video SDP and just set the hold flag a=inactive when video is not required)

defsetholded

(int)
Set this parameter if you wish to start the audio on hold by default for new calls.
-1: not used (default)
0=no
2=sendonly
3=recvonly
4=both in hold

defsetholdedvideo

(int)
Set this parameter if you wish to start the video on hold by default for new video calls.
-1: not used (default)
0=no
2=sendonly
3=recvonly
4=both way on hold

use_codec

(int)
Instead of just using the video_codec parameter to list the allowed codec's and the video_prefcodec parameter to specify one higher priority codec, you might use the use_codec settings to better control the priorities (replace the codec substring with the codec name, such as use_vp8 or use_h265).
Possible values:
0=never

1=don't offer (might use only if there are no other possibilities)
2=yes with low priority
3=yes with high priority

Default is 2 for use_h264 and use_vp8 and 1 for the rest.
Example:

use_H264=3
use_VP8=2
use_H263=1
use_H265=1
use_H261=0
use_VP8=0
use_mpeg1=0
use_mpeg2=0
use_mpeg4=0
use_theo=0

The above example will do the following:

- enable H.264 with highest priority
- enable VP8 with lower priority
- enable H263 and H264 only if the peer offers only one of these (so these codecs will not be offered by default by JVoIP
- completely disable H.261, VP9, theora and the mpeg codec's

payloads

(number)
The RTP payload number is auto negotiated, but you can specify the default offers by the following codec_payload parameters (make sure to set it to a value which is in the dynamic range and doesn't match another existing audio or video codec):

- h263_payload (default is 103)
- h264_payload (default is 126)
- h265_payload (default is 125)
- vp8_payload (default is 123)
- vp9_payload (default is 124)
- mpeg4_payload (default is 99)
- theora_payload (default is 100)

display

There are 3 different video playback windows that might be displayed:

- *Remote video*
- *Local video preview direct (a separate process which will try to record directly from local camera)*
- *Local video preview stream (playback the locally recorded stream which is also sent to the remote).*

Most video recorder devices are limited to single access so the direct preview stream usually can't be used while we also stream to the other party. This is handled automatically by JVoIP (will close the direct preview when streaming to the peer begins)

You might wish to embed the video player window in your app. For this, you can use the OS API for window positioning.
For example on Windows (winapi):
First use FindWindow after the below mentioned "video_title_remote" to find the playback window handle (or EnumWindows to find it after partial title text)
Then use some of the following functions to position the window after your needs: ShowWindow, SetParent, SetWindowPos, SetWindowLong, MoveWindow.
Use the above logic from a timer or when your windows position change.
For Windows the above functions can be also offered by JVoIP (so you can just use the JVoIP API from Java for windows positioning. Ask us for the details if you need this).
A similar logic can be implemented also for other operating systems with the OS/X11 API functions or you might use something like xdotool.

You can control the playback window display with the following parameters:

video_border: Show/hide window border. 0: no, 1: yes (default)
video_alwaysontop: Position the playback window above all other windows. 0: no (default), 1: yes
video_preview_alwaysontop: Position the preview window above all other windows. 0: no (default), 1: yes
video_width: Set the width of the playback window. -2: ffplay default, -1: jvoip default (default), others: specify
video_height: Set the height of the playback window. -2: ffplay default, -1: jvoip default (default), others: specify
video_preview_fwidth: Set the width of the preview window. -2: ffplay default, -1: jvoip default (default), others: specify
video_preview_fheight: Set the height of the preview window. -2: ffplay default, -1: jvoip default (default), others: specify
video_left: Set the left position of the playback window. Default is -1 (auto)
video_top: Set the top position of the playback window. Default is -1 (auto)

video_preview_left: Set the left position of the preview window. Default is -1 (auto)
video_preview_top: Set the top position of the preview window. Default is -1 (auto)
video_title_remote: Set the title of the playback window. Default is “App name –Remote video from: peername”
video_title_local_direct: Set the title of the direct preview window. Default is “App name -Local video preview direct”
video_title_local_stream: Set the title of the streaming preview window. Default is “App name -Local video preview stream”

others

Some more advanced internal parameters are listed below:

has_video: Enable/disable video features. Valid values: true/false.
video_autosend: 0=disable,1=no (default) ,2=auto send video,3=auto send screen (to disable video set the acceptvideo and sendvideo to 0)
video_autoaccept: 0=never,5=ask,10=yes,15=always (extra settings: 3=no if not my contact, ask otherwise, 6=auto ask, 7=auto if my contact ask otherwise). Default is 6 (ask)
video_autosendonrec: 0=never,5=ask (default),10=yes,15=always (extra settings: 3=no if not my contact, ask otherwise, 7=auto if my contact ask otherwise)
video_sendsdp: 0=never,1=when needed (default; on video calls only),2=always,3=always screen (this is to accept video; a more technical setting than the followings). Set whether to initiate video with SIP INVITE.
video_fakeinput: you can set it to a video file name to use it instead of webcam recoding
video_testmode: useful for internal testing only: 0: no, 1: record from movie.mkv, 2: other test things, 3: local loopback between ffmpeg and ffplay (use the same port, not routed via jvoip)
video_keyframes: keyframe interval for video recording (GOP length). Maximum distance between I-frames. Default is 60. Lower values will require higher bandwidth, but the receiver side will be able to start the playback earlier. Higher values consume less bandwidth but the start of the playback on the other end might be delayed
video_payloadrewrite: 0=don't touch (default), 1= rewrite outgoing, 2=rewrite incoming, 3=rewrite incoming to 33, 4=rewrite both, 5=rewrite both (incoming to 33), other=rewrite to number specified ,6=rewrite both in/out other=rewrite to number specified
video_payloadrewrite_rtp: rewrite if the signaled payload doesn’t match with the RTP (some devices might send incorrectly). 0=never (default),1=on ts mux,2=always
video_relay_opennat: will send NAT open packet(s) on the player port, useful for external video which can’t send/receive packets from the same port. 0: no, 1: yes.
video_rtprewrite: if to rewrite the RTP header SSRC and sequence number for the video streams. 0: no, 1: rewrite for outgoing video
video_port_ffmpegsendto: suggest local video port
video_port_ffplaylisten: suggest player video port
video_device: Specify the name of the video camera or webcam device to be used. You can set this also with the API_SetVideoDevice() function

API

Video related API’s are listed details.
For more details about the JVoIP general API see the JVoIP documentation “API” chapter.
Most of these API’s are also listed also in the base documentation, with less video specific details.

boolean API_VideoPreview(int line, int type)

Launch local video preview.
Both the line and the type parameters are optional. The default value for line is -1 and for the type is -2.
If the line parameter is 3 then the preview will be launched separately (not attached to any SIP endpoint).
The type parameter can have the following values:

- -2: default (loaded from the video_preview parameter)
- -1: auto (usually defaults to 4 if not headless and there is a video device)
- 0: stop video preview
- 1: direct playback before start sending (at setup/ring until connect)
- 2: local stream during call (after call connect only)
- 3: always with direct playback first (until call connect) and then local streaming
- 4: always from local streaming for incoming calls, direct + streaming for outgoing calls

Example:

```
wobj.API_VideoPreview(); //start video preview
wobj.API_VideoPreview(-2, 0); //stop video preview
```

You should set the video_preview parameter to 0 if you wish to control the preview explicitly, otherwise JVoIP might automatically launch a video preview window.

boolean API_Call(int line, String peer, int calltype, int audiodirection, int videodirection)

Video calls can be initiated by calling this function with the calltype parameter set to 1 or 2.
Initiate call to a number or sip username.

Parameters (all parameters are optional):

Line:

specify line with incoming call (or -2 for all or -1 for auto guess)

Peer:

Phone number, Username/Extension or SIP URI to dial.
If the peer parameter is empty, then will redial the last number.

Calltype:

-1: auto/default
0: initiate voice call
1: initiate video call
2: initiate screen-sharing session

Audiodirection (set initial hold state for audio):

-1: inactive
0: send only (JVoIP will record only, don't playback; JVoIP will send a=sendonly; the remote should answer with a= recvonly)
1: receive only (JVoIP will playback only, don't record; JVoIP will send a=recvonly; the remote should answer with a=sendonly)
2: both (default)

Videodirection (set initial video hold state applicable for the video stream if any):

Same as audiodirection, but for video (if video media exists)

Example: `wobj.API_Call(-1, , "DESTINATION", 1); //initiate a video call`

boolean API_Accept(int line, int calltype, int strict, int audiodirection, int videodirection)

Connect incoming (video) call.
You should call this function when there is an incoming ringing call if you wish to connect the call.
For example you might present an Accept/Reject popup on "Ringing" STATUS and call this function when the user press the Accept button.

Parameters (all parameters are optional):

Line:

specify line with incoming call (or -2 for all or -1 for auto guess)

Calltype:

-1: auto/default
0: audio only
1: with video
2: force video

Strict:

Set to 1 to treat the line parameter strictly.
If set to 1, then the function will fail if there is no incoming call on the specified line.
Otherwise, if set to 0, then it will auto guess the call to be accepted if there is no call on the specified line, but there is a call on other line.
Default is 0.

Audiodirection (set initial hold state for audio):

-1: inactive
0: sendonly
1: recvonly
2: both (default)

Videodirection (set initial video hold state applicable for the video stream if any):

Same as audiodirection, but for video (if video media exists)

Example: `wobj.API_Accept(-1, 1); //accept (connect) the current incoming call with video`

Note:

If the call was initiated without video capabilities by the remote peer, then video might not be added even if you call the API_Accept with video. In this case you should call the API_AddVideo for a re-INVITE with video capabilities.
Alternatively you might just set the enableautoaccept parameter to 3 if you wish to auto answer all calls.

boolean API_AddVideo(int line, int calltype, int direction)

Add video media for an existing voice call.
Can be also used as video reload if it was put on hold before.

The call-type can have the following values:
- 1: add video
- 2: add screen-sharing

The direction can have the following values:
- -1: inactive
- 0: sendonly
- 1: recvonly
- 2: both

Note: You might start an audio-only call (with the API_Call, API_Call or incoming call) and add video later calling this function which will send an UPDATE or reinvite to start video capabilities negotiation.

boolean API_StopVideo(int line, int direction)

Will remove the video stream at the specified line.
Can be also used as an alternative for video hold.

The direction parameter can accept the following values:
- 0: stop send (will keep only receiving from remote and playback if any)
- 1: stop receive (will keep only recording and streaming to remote if any)
- 2: stop both way

Note: you might change the video_stoponstop and the video_holdonstop parameters to modify the behavior on video stop/remove.

boolean API_Hold(int line, int direction, int media)

Put call on hold. This will set the hold state of the endpoint and might trigger an UPDATE or a reINVITE request.
Parameters (all parameters are optional):

Line:
- -3: to be applied for the next action (next endpoint default/INVITE/200 OK within 10 seconds)
- -2: all lines
- -1: current line (default)
- 1+: for the specified line

Direction:
- -2: no hold change
- -1: default (holdtypeonhold)
- 0: unhold
- 1: mute instead of hold (not recommended. use API_Mute instead)
- 2: send only (mute playback. JVoIP will send a=sendonly)
- 3: receive only (mute recording. JVoIP will send a=recvonly)
- 4: both in hold (inactive)

Media:
- 0: audio
- 1: video
- 2: both (default)

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

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.
The calltype can have one of the following values to specify what kind of call have to be initiated to the transfer target:

-1: auto guess (default)
0: voice call
1: video call
2: screen-sharing

boolean API_IsIncomingVideo()

Check if current incoming call is video call.
Returns true if there is an inbound video call (ringing)

int API_GetIncomingVideoEndpoint(int line)

Return line number of the incoming call if any, otherwise -1.
If the line input parameter is higher then 0 then it will check only the supplied line and will return the same line number back if there is incoming video on the line or -1 otherwise.
If the line input parameter is less then 0 then it will return a line number with incoming video call or -1 if there is no any incoming video call.

int API_GetVideoEndpoint(int line)

Return line number of a video call if any, otherwise -1.
If the line input parameter is higher then 0 then it will check only the supplied line and will return the same line number back if there it is a call with video, otherwise will return -1.
If the line input parameter is less then 0 then it will return a line number with video call or -1 if there is no any video call in progress.

boolean API_SendVideoRTP(int line, byte[] buffer, int length)

Stream video to remote peer by passing video RTP packets with this function.
This is required only if you wish to send video from your app or from an external app (not using the JVoIP built-in video recording functionality)
See the Streaming chapter for more details.

byte[] API_GetMedia(int timeout)

You might use this function to retrieve the incoming video RTP packets in case if you wish to process them in your app.
This is required only if you wish to receive the video packets to your app or to an external app (not using the JVoIP built-in video playback functionality)
See the "API_GetMedia" function description and the "How to get the media stream" FAQ point in the JVoIP documentation and the Streaming chapter for the details.

String API_GetVideoDeviceList(int async, boolean strict)

Returns a list of installed camera devices or empty string if none are found.
Both the async and strict parameters are optional so you can just call API_GetVideoDeviceList();
Example: `wobj.API_GetVideoDeviceList(); //return a list of video devices (using async 1 and strict false)`

If async is 2, then will return immediately (non-blocking) with cached device list (but will continue with device query in the background if necessary)
If async is 1 then will return immediately only if already obtained the devices previously
If async is 0 then will always try to reload the device list (blocking read, so the function call might take some time to return)

If strict is false then it might failback to old cached result if can't obtain any new device or if timeouts after 10 seconds.
If strict is true then it will return empty string if can't list the video input devices.
Strict has no effect if async is 2.

String API_GetVideoDevice()

Get the selected video input device name if any (camera/webcam device).

boolean API_SetVideoDevice(String devicename)

Set the video input device (camera/webcam device).
Set to "Default" to try using the default device.

Notifications

You can usually ignore all the below mentioned video related notifications as it should not influence your app logic. You can process video calls exactly the same way as regular audio-only calls by processing the STATUS notifications to get the state of the JVoIP state machine and update your app logic as required.

Beside the usual STATUS and other notifications, you will also receive a VIDEO,START/STOP notification in case if you are interested in when exactly the video recorder/playback is started/stopped and it's details.

VIDEO

The VIDEO notification is more helpful if you are using an external video application for video record/playback (video_module=3). Otherwise you can treat it just as an informational notification if you are interested in the exact video details which are used for streaming/recording/playback.

See the SIPNotification.Video in the javadoc for the SIPNotification object details.

String format:
VIDEO,startstop,type,line,reason,ip,port,codec,payload,quality,bw,max_bw,fps,max_fps,width,height,profilelevelid,profile,pixelfmt,level,pm,sprop,srtp_alg,srtp_key,srtp_remotekey,device,fmtp

Parameters:

0. **VIDEO: header**
1. **startstop: can be START (when player/recorder starts) or STOP (when player/recorder stops)**
2. **type:** 0=undefined (ignore),1=cache (ignore),**2=player,3=webcam recorder**,4=screen recorder,5=local preview before call, 6=local display in call
3. line: endpoint line (channel) number
4. reason: a string about the video start/stop reason. Example: disconnect
5. **ip: the remote or local video RTP stream IP address**, depending on the context
6. **port: the remote or local video RTP stream port number**, depending on the context
7. **codec: video codec**. Example: h264
8. **payload: codec payload number**. Example: 123
9. quality: suggested quality (values same as for the"video_quality" parameter). Example: 8
10. bw: suggested max bandwidth in kbit/s. Example: 1000
11. **max_bw: max bandwidth to use in kbit/s**. Example: 3000
12. fps: suggested frame per second. Example: 24
13. **max_fps: max fps allowed**. Example: 30
14. width: video scale width. Example: 1280
15. height: video scale height. Example: 720
16. **profilelevelid: the negotiated H.264 profile-level-id**. Example: 42e015
17. profile: suggested H.264 profile (baseline/main/high). Example: baseline
18. pixelfmt: suggested H.264 pixel format. Example: yuv420p
19. level: suggested H.264 level. Example: 4.2
20. **pm: packetization mode**. Example: 1
21. **sprop: sprop-parameter-sets if any**. Example: Z0LACqaBQfsBEAAAAwAQAAADAkjxImo=,aM48gA==
22. **srtp_alg: SRTP crypto suite**. Example: AES_CM_128_HMAC_SHA1_80
23. **srtp_key: SRTP local key (which was sent to peer)**. Example: inline:o4vjJ1k8EpU0qkUGunhL9wq9EH4YSlkg263KMIaU
24. **srtp_remotekey: SRTP remote key** (which was received from peer). Example: inline:o4vjJ1k8EpU0qkUGunhL9wq9EH4YSlkg263KMIaU
25. device: device name to use. Example: Internal Webcam
26. fmtp: the full fmpt SDP line for the negotiated codec

Notes when used with external recorder (video_module=3):

- You must consider all the parameters with **bold** to be used for your video recording and streaming configuration. The other parameters might be ignored.
- For some of the parameters you will receive empty strings. These can be omitted or you can use any value after your app logic.
- For the video playback a video_play.sdp file is also generated which can be used to feed the player instead of parsing the VIDEO notification parameters.

STATUS

The STATUS notification also contains some video related parameters.

STATUS,line,statustext,peername,localname,endpointtype,peerdisplayname,[callid],online,registered,incall,mute,hold,encrypted,**hasvideo**,group,
rtpsent,rtprec,rtploss,rtplosspercet,**videohold**,**videosent**,**videorec**,serverstats

HasVideo: specifies if call is with video: 0: no, 1: any video, 2: received offer, 3: video socket ok, 4: video playback/recording started, 5: video streaming
VideoHold: video hold status: 0=no,1=undefined,2=sendonly, 3=recvonly,4=both in hold
VideoSent: sent video RTP packets
VideoRec: received video RTP packets

LOG

With the LOG,RTP notifications you can receive some more video related statistics at the end of the calls.
Example: RTP: sent 15695 lasts: 0 (p2p: 0 sdp: 0 rrp: 15695 tnl: 15701), rec: 17281 lastr: 0, loss: 201 1%, **vsent: 0/0, vrecv: 0/0**, cpu: 0.0%, cpurel: 0.0% (0.0), srvsent: 10326 srvrec: 10302 srvloss: 10 0%

For vsent the first number is the recorded video packets. The second number is the RTP packets actually sent to the peer.
For vrecv the first number is the received video RTP packets from the peer. The second number is the packets actually pushed to playback.

FFmpeg

JVoIP implements SIP video calls by using an external video recorder and player.

Download
If ffmpeg/ffplay is not found on the system then JVoIP will download them on demand automatically.
You can modify this behavior with the following parameters:
video_download: -1: auto (default), 0: disable (never download), 1: download on demand, 2: pre-download if missing, 3: bundle (but never download), 4: bundle and auto download, 5: bundle and pre-download if missing
video_recorder_path: ffmpeg path (directory or full path; otherwise auto guess or it will download if doesn't exists)
video_player_path: ffplay path (directory or full path; otherwise auto guess or it will download if doesn't exists)
video_download_win: video module download URL for windows (if not bundled and you wish to modify the default download path)
video_download_lin: video module download URL for windows (if not bundled and you wish to modify the default download path)
video_download_osx: video module download URL for windows (if not bundled and you wish to modify the default download path)

If you wish to specify the download paths, then you must provide the HTTP/HTTPS download URL for a zip file which contains the ffmpeg and ffplay executables.

In case if your use-case is video calls, then you might bundle the ffmpeg/ffplay executables with JVoIP (just copy them near the JVoIP.jar file).
You might download the official versions from here or here or our builds from here: windows, linux, macos.

If ffmpeg is not found and it was not configured to be pre-downloaded and it the network speed is too slow, then the first video call might fail if JVoIP is unable to download it in time.

Using another video recorder / player
FFmpeg/FFplay was heavily tested for JVoIP and are used in a highly optimized and efficient way (bandwidth efficient, compatible, minimized delays, etc).
However, it is also possible to use another video recorder/player instead of ffmpeg/ffplay. For example you might use mpv, mplayer, vlc or other software capable for video recording, playback and streaming.
Use the following parameters to specify the path to your video recorder/player executables with the ffmpegpath and ffplaypath:
video_recorder_path: path to your video recorder executable
video_player_path: path to your video player executable

In this case you must set the command line by the following parameters:
cmd_video_devices: video device list command
cmd_video_play: video playback command
cmd_video_play_self: self-camera command
cmd_video_play_self_stream: self-camera from local stream command

cmd_video_rec_camera: video recording command
cmd_video_rec_screen: screen recording command

It is also possible to completely bypass the JVoIP built-in video recording and playback and process the video stream in your own app instead (implement any custom video recorder and player or just process the video streams after your needs).
In this case you will need to set the video_module parameter to 3, so JVoIP will not record or playback the video streams itself, but it will receive/send to your app.
See the below Streaming chapter for the details.

Streaming

Video streaming is also supported by JVoIP.
This is about processing the video streams in your app or by an external service/app (not the usual SIP video call streams exchanged with the remote VoIP caller/caller party).
This is required only if you wish to send/receive the video from/to your app (or from/to an external app such as ffmpeg/ffplay) not using the JVoIP built-in video recording/playback functionality.

Streaming is already described also in the JVoIP documentation at the "How to send a media stream" and "How to get the media stream" FAQ points, but that description focuses on audio streaming (although some details described there applies also for video).

We separated the description of the video streaming specific functionality into a separate document to avoid any confusions:
SIP video streaming documentation

Troubleshooting

Common failure reasons:

- Unable to auto-download ffmpeg/ffplay (copy these binaries near the JVoIP.jar)
- Camera device not available (plugin the USB for external devices)
- Peer or server doesn't have (proper) video capabilities or not conform to SIP standards
- Peer or server don't have support for the video codec's enable in JVoIP (adjust the codec's with the video_codec parameter)
- Too high quality in the video offer (decrease the video_quality parameter value)
- Peer is not offering/responding with video capabilities in the SDP (m=video)
- Peer is not sending the keyframe periodically
- Peer is not sending video (received to video port X from A:B, video rtp received from A:B, RTPVideoReceive: start video player on receiving video RTP from remote peer)
- Make sure to test with a peer, which can correctly handle video. Here are our test results (might be generated with older software versions):
    - linphone: works, but it has vp8
    - zoiper: unreliable (vp8, h263; sends only one single video packet -but accept works correctly)
    - bria: unreliable (h263 only; at first sends video port 0. might activate only later)

For testing you should always set the jvoip loglevel parameter to 5 (or higher if you wish to see also RTP details, but that will increase I/O).
Search for "video" in the logs for the video related details.

If you need our help, please contact us by email with the followings:

- Detailed problem description (including What you did? What happened? What should happen?)
- Detailed JVoIP log (the generated webphonelog.dat logfile, the java console output or as described at the "How to generate and send logs from JVoIP" FAQ point in the documentation)
- Wireshark pcap trace if the problem is RTP related

Resources

JVoIP homepage: https://www.mizu-voip.com/Software/SIPSDK/JavaSIPSDK.aspx
JVoIP demo: https://www.mizu-voip.com/Portals/0/Files/JVoIP.zip
JVoIP documentation: https://www.mizu-voip.com/Portals/0/Files/JVoIP.pdf
This document (PDF): https://www.mizu-voip.com/Portals/0/Files/JVoIP_Video.pdf
This document (CHM): https://www.mizu-voip.com/Portals/0/Files/JVoIP_Video.chm

This document (HTML): https://www.mizu-voip.com/Portals/0/Files/documentation/jvoip_video/index.html
Video streaming: https://www.mizu-voip.com/Portals/0/Files/JVoIP_Video_Streaming.pdf
Old (deprecated) video module: http://www.mizu-voip.com/Portals/0/Files/websipphone_video.zip
SIP RFC: https://www.rfc-editor.org/rfc/rfc3261
FFmpeg home page: https://ffmpeg.org/
FFmpeg binaries: https://www.gyan.dev/ffmpeg/builds/packages/
FFmpeg windows binaries from mizutech: https://www.mizu-voip.com/portals/0/files/ff.zip
x264-ffmpeg parameter mapping: https://sites.google.com/site/linuxencoding/x264-ffmpeg-mapping
VP8 SDP and RTP payload: https://tools.ietf.org/html/rfc7741
VP9 SDP and RTP payload: https://tools.ietf.org/html/draft-ietf-payload-vp9-11
MPEG-4 SDP and RTP payload: https://tools.ietf.org/html/rfc6416
H264 SDP and RTP payload: https://tools.ietf.org/html/rfc6184
H264 codec: http://en.wikipedia.org/wiki/H.264/MPEG-4_AVC
H.265 SDP and RTP payload: https://tools.ietf.org/html/rfc7798
Contact email: info@mizu-voip.com