JVoIP –Video Streaming

Contents

About

This document describes the usage of the JVoIP SIP video streaming features: the capability to send/receive the video stream from other sources.
The main use case is the capability to integrate JVoIP with external video recorder/playback (such as ffmpeg or any video capable app), but otherwise it is up to you how do you handle these streams (video recording, AI processing, routing it to some cloud service, etc).

In this use-case, JVoIP is mainly responsible for the signaling only (video negotiation) as the actual recording/playback will be done from external software.
Your app (from where the video stream is received) must be able to send and/or receive valid standard based RTP streams.

This document is only a help for the video streaming use-case. Refer to the JVoIP documentation and the JVoIP video documentation for the other details.
This means that the full usage is described in three documents:

1. JVoIP documentation (JVoIP.pdf): describes the JVoIP basic usage (probably you already have this implemented)
   See the "How to send a media stream" and "How to get the media stream" FAQ points for the streaming related details.
2. JVoIP video documentation (JVoIP_Video.pdf): is an extension for the above JVoIP documentation focusing on the video related functionalities
3. This document (JVoIP_Video_Streaming.pdf) : is an extension for the above JVoIP video documentation, focusing only on video streaming (changes and details about this specific use-case). PDF, CHM, HTML.

Usage

First of all, make sure that simple audio only calls are working correctly using JVoIP with your SIP server (or P2P direct calls to other peers).
Then you might test a video call with the JVoIP built-in video module (with the video_module parameter set to -1 or 2).
Once these are working, you can go ahead to implement video streaming as described in this document.
Adding video streaming support should be simple as shown in the pseudocode example:

1. Set the related parameters for video streaming
2. Make/receive a video call using the related API
   To initiate a video call, use the API_Call funcion with the calltype parameter set to 1 and/or use the other video related functions such as API_Accept with the calltype parameter set to 1 and the API_AddVideo / API_StopVideo functions.
3. Watch for the VIDEO notifications
   You can start/stop sending/receiving based on the call state (STATUS notifications) or based on the VIDEO notifications.
4. Send/receive the video RTP packets over UDP or by API function calls as described at the mode chapter

See the JVoIP video documentation for the details about the related parameters/functions/notifications.

We recommend to first test with the very basic settings a simple outgoing video call (API_Call with calltype 1):

- No any quality parameters passed (no video_quality, video_profilelevelid and other quality related parameter passed)
- No SRTP (set the mediaencryption to 0)
- Set the video_relay to 1

Once the basic video call works, you can test the advanced functionalities one by one:

- Set any quality related parameter such as video_quality and/or video_profilelevelid
- Incoming call (API_Accept with calltype 1)
- Start with audio-only call and add/remove video later using the API_AddVideo / API_StopVideo functions
- Enable SRTP if you need media encryption (set the mediaencryption to 2)
- Test with the above combinations

See the “Troubleshooting” chapter in the JVoIP video documentation if you run into any issue.

Mode

There are two ways to handle video streaming with JVoIP:

UDP

In this case your app (or an external app) must send/receive the video stream over RTP/UDP as directed by the VIDEO notifications.
You should use this mode if you are running an application capable to emit or receive standards based RTP/UDP video streams (for example ffmpeg/ffplay).

Set the following parameters:
video_module: 3 (possible values -1: default, 0: disabled, 1: old deprecated applet method, 2: video handling by JVoIP, 3: streaming)
video_relay: 0 or 1:
0: for video stream directly between your software and the peer, bypassing JVoIP
1: if JVoIP have to route the RTP (int this case it will handle all the details, including NAT’s and SRTP if required)

Pass the parameters received by the VIDEO notification to your app (port number, codec, etc). Your app will have to send/receive video RTP via UDP (like in a usual SIP call; most video).

API

In this case your app can send video RTP packets using the API_SendVideoRTP function and it can receive the incoming video packets using the API_GetMedia function.
You should use this mode if you wish to send and receive the video RTP packets directly as byte buffers, using API function calls.

Set the following parameters:
video_module: 3 (possible values -1: default, 0: disabled, 1: old deprecated applet method, 2: video handling by JVoIP, 3: streaming)
video_relay: 1

video_suppresslocalrtp: 1 (so JVoIP will not send the incoming RTP packets also on RTP/UDP)

Use the API_SendVideoRTP function to send RTP video packets and/or use the API_GetMedia function to receive the incoming RTP video packets.

Parameters

The most important video streaming use-case related parameters are the followings:
Set the video_module parameter to 3 (possible values -1: default, 0: disabled, 1: old deprecated applet method, 2: video handling by JVoIP, 3: streaming)
Set the video_relay parameters after your needs:
0 (in UDP mode only to have the video stream directly between your software and the peer, bypassing JVoIP)
1 (JVoIP route the RTP and will handle also all the related task such as NAT handling and SRTP encode/decode when required).
For "API mode" set also the video_suppresslocalrtp parameter to 1.
Optionally, set the video_config parameter for example to 4 (possible values: -1: auto, 2: never enable 4: on request only 6: ask 8: yes 10: always force)
You might specify the video quality with the following parameters:

- video_quality
- and/or video_profilelevelid
- and/or video_fps / video_bitrate / video_vwidth / video_vheight
- any other parameters after your needs (general parameters in the JVoIP documentation and video related parameter described in the jvoip video documentation.)

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.

With the "API" streaming type, use the API_SendVideoRTP function to send RTP video packets and/or use the API_GetMedia function to receive the incoming RTP video packets.

More details (describing the above mentioned methods and usage):

- JVoIP documentation (especially the "How to send a media stream" and "How to get the media stream" FAQ points)
- JVoIP video documentation

Notifications

You will receive a **VIDEO notification** when recorder/player have to be started/stopped.

- **Start the recorder** (your video recorder app such as ffmpeg) when your receive a VIDEO,START notification with the type parameter set to 3 or 4.
  Make sure to configure your recorder as suggested by the parameters.
- **Start the player** (your video player app) when your receive a VIDEO,START notification with the type parameter set to 2.
  For ffplay, instead of parsing all the parameters, you can just pass the **video_play.sdp** which will be generated by JVoIP in its work directory which is usually the mwphonedata or you can get the exact path like this: String sdppath = wobj.API_GetWorkdir()+"video_play.sdp";
- **Stop the recorder** when your receive a VIDEO,STOP notification with the type parameter set to 3 or 4.
- **Stop the player** when your receive a VIDEO,STOP notification with the type parameter set to 2.
- You might also start/stop a local video preview when a video call begins/ends (launch it when you receive START,5 and stop it on STOP,6)

See the JVoIP video documentation for the details about this notification.

Pseudocode

```
...set basic parameters such as serveraddress, username, password, loglevel, mediaencryption, srtp_suite, etc...
wobj.SetParameter("video_config", 4); //can be omitted as video is not disabled by default
wobj.SetParameter("video_module", 3); //enable streaming
wobj.SetParameter("video_relay", 1); //let JVoIP handle the video RTP, including SRTP (or set to 0 to bypass)
//wobj.SetParameter("video_suppresslocalrtp", 1); //set this only for "API" streaming type (not for RTP/UDP streaming)
```

```
wobj.SetParameter("video_quality", 12); //optionally suggest quality
wobj.SetParameter("video_profilelevelid", "xxx"); // optionally you might preset the H.254 profile-level-id
...set any other parameters as you wish...
wobj.API_Call(-1,"1234",1); //initiate video call to 1234
<- VIDEO,START,3...:  start your app recording (start sending RTP/UDP video packets to the peer or use the API_SendVideoRTP function)
<- VIDEO,START,2...:  start your app playback (start receiving RTP/UDP video packets from the peer or use the API_GetMedia function)
...video call here...
wobj.API_Hangup(); //disconnect the call (or it can be disconnected by the peer)
<- VIDEO,STOP,3...:  stop your app recording (don't send any more video RTP packets to the peer)
<- VIDEO,STOP,2...:  stop your app playback (you will not receive any more RTP video packets from the peer)
```

Relay

Use the video_relay parameter to specify whether to route or bypass video in "UDP" node:

- 0: Send/receive the video RTP stream directly to/from the peer from your app. In this case you are responsible for all the processing (including SRTP encrypt/decrypt if required)
- 1: Let JVoIP to route the video and perform the SRTP encrypt/decrypt if required (recommended). In this case your app will send/receive video to/from local JVoIP which then will forward to/from the peer. This is the default value.

In "API" mode this parameter must be always set to 1.

With variant 0 you might run into NAT handling issues if your app is not capable to send/receive from the same port.
There is little difference between relay 0 and 1 from your app point of view, except that with variant 0 you must handle the SRTP encryption yourself.
The IP and port in the VIDEO notification will be set correctly by JVoIP depending on this video_relay parameter.

- For variant 0 you will receive the peer IP:port for recording (where your app have to send the video RTP) and the localip:port to be used for playback (where ffplay have to listen for the incoming RTP)
- For variant 1 you will receive the JVoIP IP:port for recording (where your app have to send the video RTP) and the 127.0.0.1:port to be used for playback (where ffplay have to listen for the incoming RTP)

To enable SRTP negotiations, set the mediaencryption parameter to 2. You can suggest a preferred algorithm with the srtp_suite parameter.

- For variant 0, then you must send the recorded RTP stream with SRTP encryption and decrypt the incoming SRTP stream for playback.
- For variant 1 then SRTP is handled transparently by JVoIP (you will need send/receive unencrypted RTP).
- No SRTP encoding/decoding is required if you don't receive the SRTP keys with the VIDEO notifications

Recording

Here the "recording" means saving the video call to file (not media recording from a video device).
The video can be already streamed as described in this document and from there you can process it (save it to file) as you wish.
A simplified method for video saving will be also added in the upcoming new major release.
Contact us if you are interested in this functionality.

FFmpeg

FFmpeg/FFplay related suggestions (in case if you are using this for video recording/playback).

Payload
Make sure to generate the RTP packets with the same payload number as specified by the payload parameter in the VIDEO notification (FFmpeg -payload_type parameter)

Port
Ideally for SIP the same port should be used for sending/receiving.
This is useful for NAT traversal and some SIP server expect it the same way and might not be able to handle separate ports correctly.
You might run into NAT handling difficulties if you set the video_relay parameter to 0 (streaming from/to peer directly from your software) and your external app(s) is incapable to send/receive from the same port.
Regarding SIP server compatibility: hopefully your server will handle this correctly, otherwise you must always use variant 1.

Regarding NAT handing: we implemented a trick in our software to send a few empty (CRLF) packets from the playback port before emitting the VIDEO,START,2 notification (also will close the UDP socket before the notification so ffplay will be able to listen on the same port). These packets should open the NAT device so you should be able to receive the incoming RTP streams even from external networks. This feature can be controlled with the video_relay_opennat parameter (set to 0 to disable, 1 to enable; default is 1).

In case if somehow you wish to launch the video player before actual playback then you can suggest a port number to be used for playback (where the remote audio will be sent) with the video_port_ffplaylisten parameter. JVoIP will try to use this port, but otherwise you must use the port number received in the VIDEO notification or as stated in the video_play.sdp.

Delay
FFmpeg and especially FFplay can introduce a delay for the video streams. You should fine-tune them with the minimal possible buffering.
Contact our support for the optimal flags to minimize ffmpeg/ffplay delay.

ffmpeg parameters
Here is a list of the ffmpeg parameters to be set regarding the received VIDEO notification parameters:

0. VIDEO: header
1. startstop: START
2. type: start ffmpeg if type is 3 or 4
3. reason: not applicable
4. ip: rtp://IP:port
5. port: rtp://ip:PORT
6. codec: -c:v CODEC
7. payload: payload_type PAYLOAD
8. quality: you might calculate other parameters based on this such as bandwidth, fps, profile, etc
9. bw: -v:b BANDWIDTHk -maxrate BANDWIDTH -bufsize 2*BANDWIDTH
10. max_bw: : -v:b BANDWIDTHk -maxrate BANDWIDTH -bufsize 2*BANDWIDTH
11. fps: -framerate VFPS -r VFPS
12. max_fps: -framerate VFPS -r VFPS
13. width: –vf scale="'min(WIDTH,iw)':-2"
14. height: –vf scale="'min(WIDTH,iw)': 'min(HEIGHT,ih)'"
15. profilelevelid: calculate the profile,pixelformat and level from this if set
16. profile: -profile:v H264_PROFILE
17. pixelfmt: -pix_fmt H264_PIXELFORMAT
18. level: suggested H.264 level. -level:v H264_LEVELH264_EXTRA
19. pm: if 0: -rtpflags h264_mode0
20. sprop: not supported by ffmpeg (should use in-band)
21. srtp_alg: haven't checked. See the ffmpeg documentation.
22. srtp_key: haven't checked. See the ffmpeg documentation.
23. srtp_remotekey: haven't checked. See the ffmpeg documentation.
24. device: -i video="CAMERADEVICENAME"
25. fmtp: not applicable

FFplay parameters
To simplify the video player usage, you can ignore the VIDEO notification parameters and just pass the generated video_play.sdp to ffplay.
Note: instead of using ffplay (which is just a playback specific variant for ffmpeg), you might use ffmpeg also for playback.

See the ffmpeg documentation for more details.