Using RTMP encoder

- Overview
    - Technical specifications
    - Codec support
    - Operation flowchart
- Quick manual on testing
- Call flow
- Parsing stream URL parameters
    - Connection parameters passing as stream parameters
- Setting a server application while RTMP stream publishing
- Sorenson Spark + Speex 16 kHz stream publishing
    - Known limits
- RTMP connection activity checking
    - Media traffic checking
    - Read timeout
    - Write timeout
    - Read and write timeout
- RTMP stream picture rotation
    - Set up
    - Testing
    - To developer
    - How to rotate stream published from ffmpeg
- Decoding buffer size management
- Incoming RTMP stream buffering
    - Stop stream buffering if stream delay grows
- Detect stream parameters by metadata or media traffic
- Known issues

To run online broadcasts you can use special hardware or software video capturing devices (Live Encoder). Such devices or programs capture a video stream and send it to the server via the RTMP protocol.
Web Call Server can receive an RTMP video stream from such a device or software (Wirecast,ffmpeg,OBS Studio,FMLEetc.) encoded to H.264 + AAC or Sorenson Spark + Speexand broadcast this video stream to browsers and mobile devices.

Overview

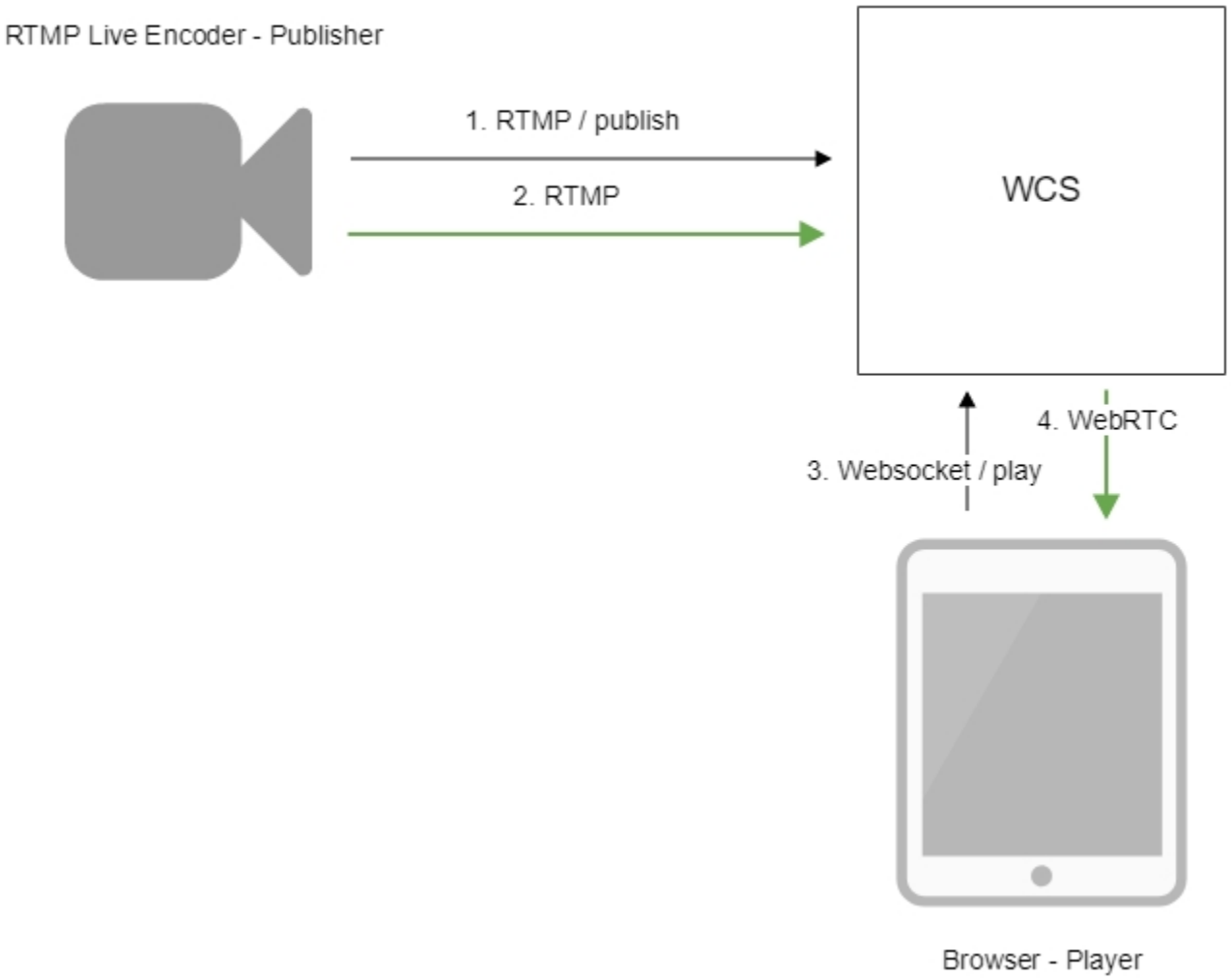
Technical specifications

- Receiving incoming audio- and video streams via the RTMP protocol
- Broadcasting of the received video stream to browsers and platforms: any among ones supported by WCS
- Uses video stream playback technologies: any among ones supported by WCS

Codec support

- Video H.264 + audio AAC
- Video Sorenson Spark + audio Speex 16 kHz

Operation flowchart



1. Live Encoder establishes a connection to the server via the RTMP protocol and sends the publish command.

2. Live Encoder sends the RTMP stream to the server.

3. The browser establishes a connection via Websocket and sends the play command.

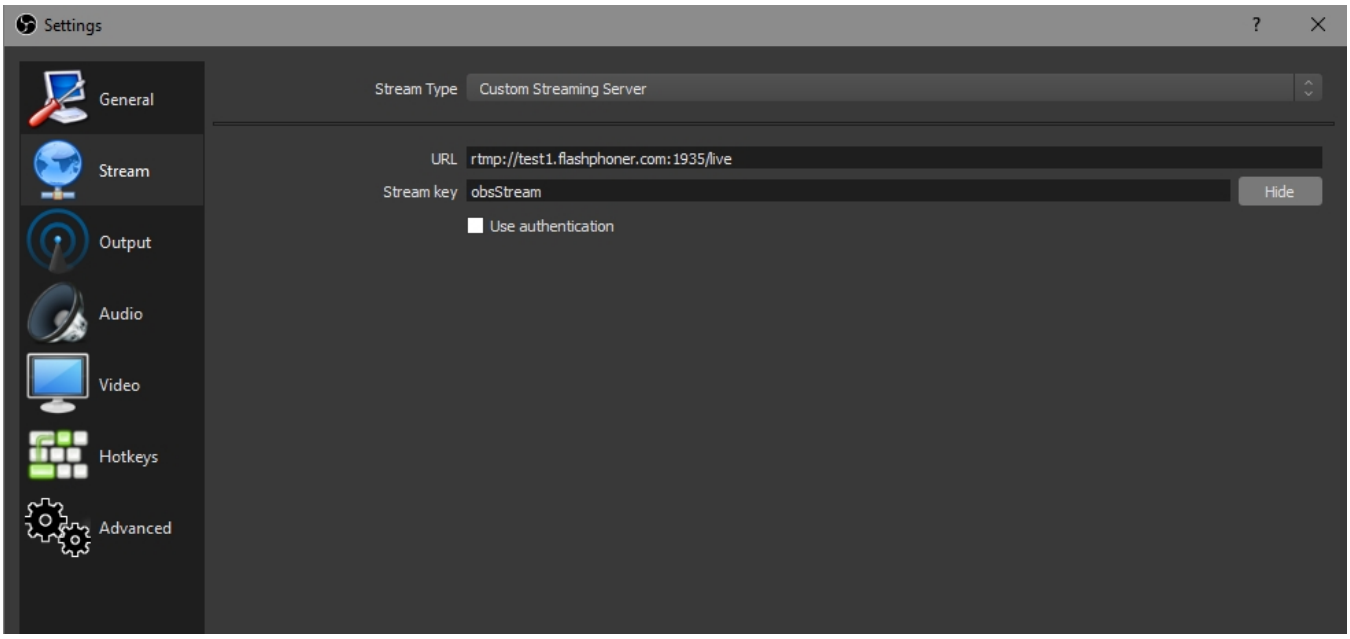
4. The browser receives the WebRTC stream and plays that stream on the page.

Quick manual on testing

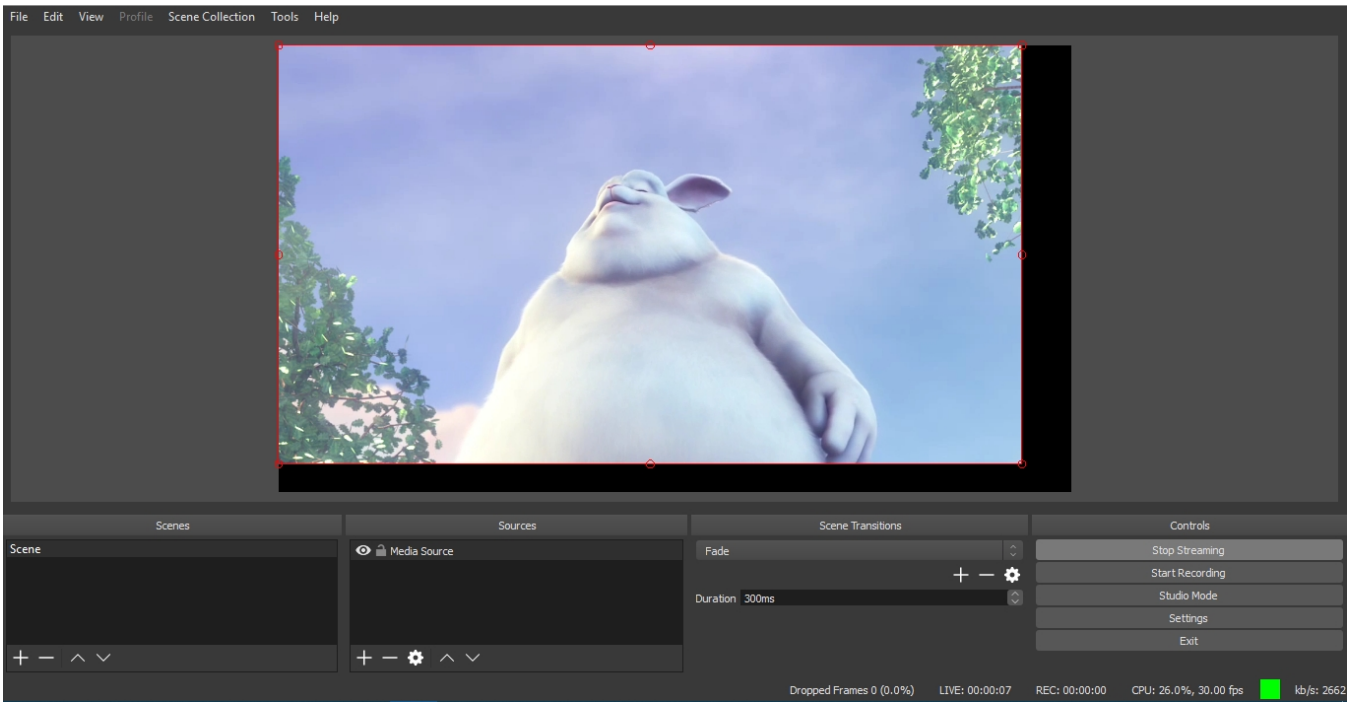
1. For test we use:

- WCS server
- OBS Studio
- Player web application in Chrome browser to stream playback

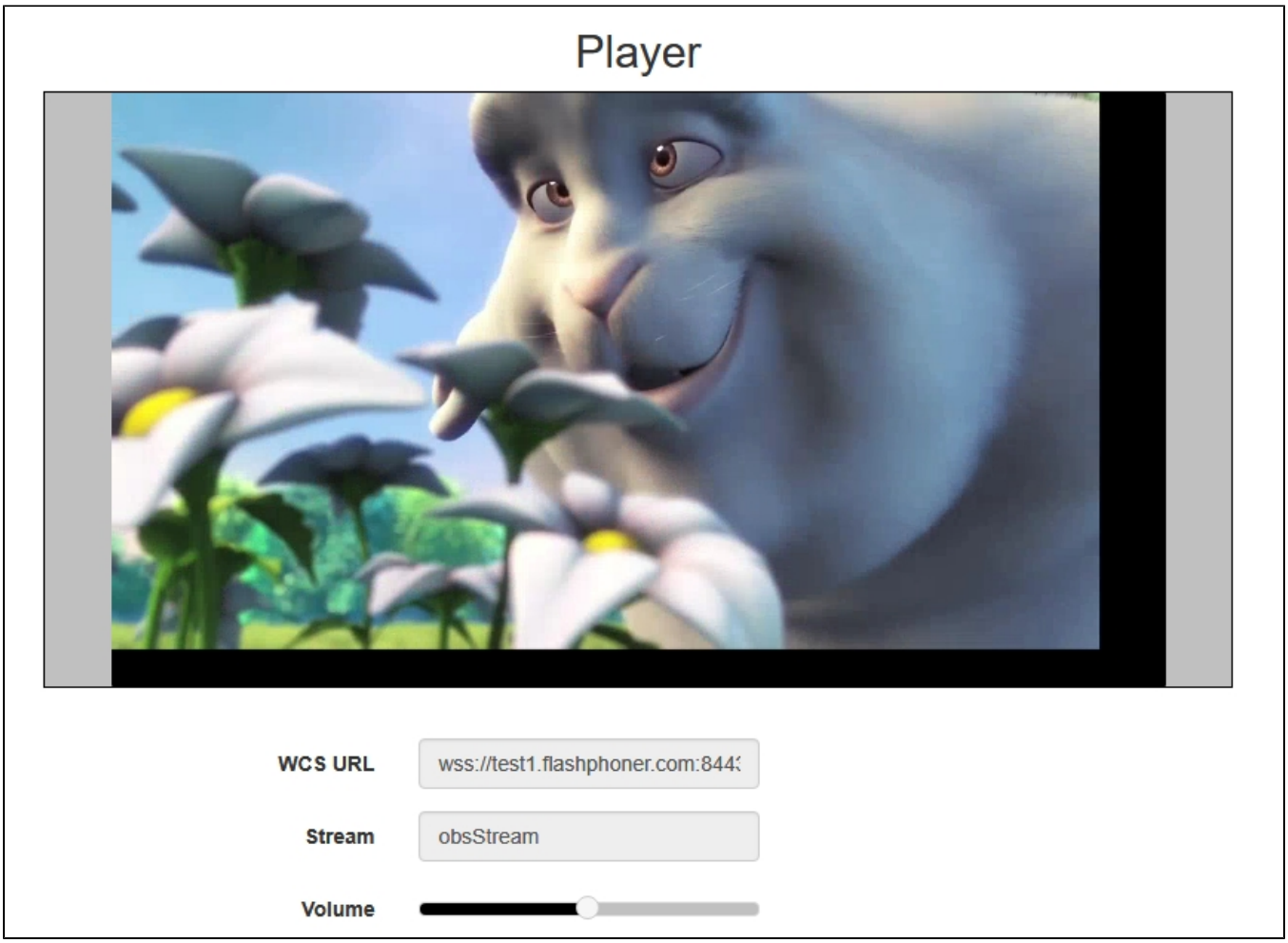
2. Set up RTMP streaming to the server address, for example rtmp://test1.flashphoner.com:1935/live/, set the stream key obsStream:



3. Start streaming in OBS Studio:

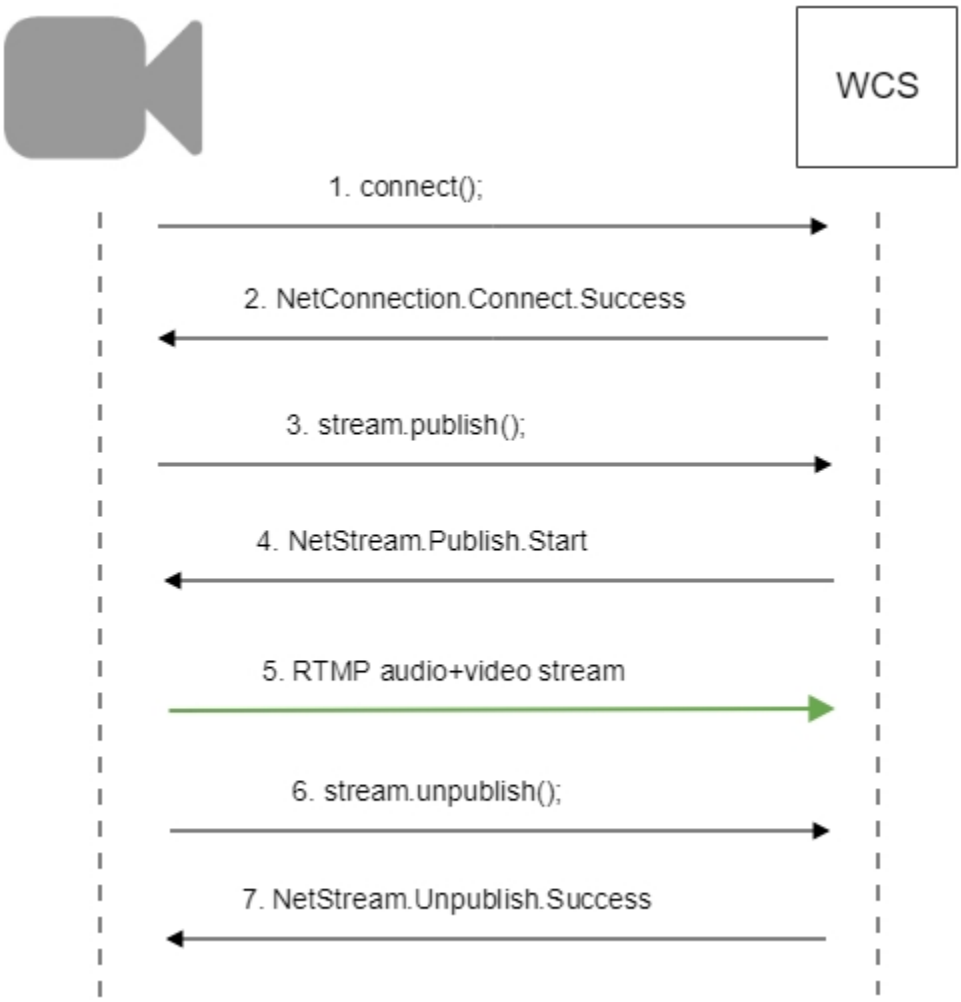


4. Open Player application. Set the stream key in 'Stream' field and press 'Start' button. The stream captured playback begins.



Call flow

Below is the call flow when an RTMP stream is broadcast from an external source (Live Encoder)to the WCS server



Parsing stream URL parameters

When RTMP stream is published or played on WCS, RTMP connection and stream parameters may be set in stream URL like this:

```
rtmp://host:1935/live?connectParam1=val1&connectParam2=val2/streamName?streamParam1=val1&streamParam2=val2
```

Where

- host is WCS server hostname;
- connectParam1, connectParam2 are RTMP connection parameters;
- streamName is stream name on server;
- streamParam1, streamParam2 are stteam parameters.

WCS server passes the parameters to backend server in REST hook in `customfield`, for example:

Connection parameters

```
URL:http://localhost:8081/apps/EchoApp/connect
OBJECT:
{
"nodeId" : "Qb3rAjf3lzoy6PEl1WZkUhRG1DsTykgj@192.168.1.1",
"appKey" : "flashStreamingApp",
"sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
"useWsTunnel" : false,
"useWsTunnelPacketization2" : false,
"useBase64BinaryEncoding" : false,
"keepAlive" : false,
"custom" : {
"connectParam1" : "val1",
"connectParam2" : "val2"
},
"login" : "rQq83sodiCPY0pJXCxGO"
}
```

Publishing parameters

```
URL:http://localhost:8081/apps/EchoApp/publishStream
OBJECT:
{
"nodeId" : "Qb3rAjf3lzoy6PEl1WZkUhRG1DsTykgj@192.168.1.1",
"appKey" : "flashStreamingApp",
"sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
"mediaSessionId" : "627990f9-8fe5-4e92-bb2a-863cc4eb43de",
"name" : "stream1",
"published" : true,
"hasVideo" : false,
"hasAudio" : true,
"status" : "NEW",
"record" : true,
"width" : 0,
"height" : 0,
"bitrate" : 0,
"minBitrate" : 0,
"maxBitrate" : 0,
"quality" : 0,
"mediaProvider" : "Flash",
"custom" : {
"streamParam1" : "val1",
"streamParam2" : "val2"
}
}
```

Playback parameters

```
URL:http://localhost:8081/apps/EchoApp/playStream
OBJECT:
{
"nodeId" : "Qb3rAjf3lzoy6PEl1WZkUhRG1DsTykgj@192.168.1.1",
"appKey" : "flashStreamingApp",
"sessionId" : "/127.0.0.1:5643/192.168.1.1:1935",
"mediaSessionId" : "stream1/127.0.0.1:5643/192.168.1.1:1935",
"name" : "stream1",
"published" : false,
"hasVideo" : true,
"hasAudio" : true,
"status" : "NEW",
"record" : false,
"width" : 0,
"height" : 0,
"bitrate" : 0,
"minBitrate" : 0,
"maxBitrate" : 0,
"quality" : 0,
"mediaProvider" : "Flash",
"custom" : {
"streamParam1" : "val1",
"streamParam2" : "val2"
}
}
```

This feature can be used for example to authenticate client on backend server while publishing or playing RTMP-stream on WCS server.

Connection parameters passing as stream parameters

In some cases it is necessary to pass RTMP connection parameters as stream parameters, authentication parameter for example

```
rtmp://test.flashphoner.com:1935/live/test?auth=key
```

This feature is enabled by the following setting

```
rtmp_use_stream_params_as_connection=true
```

In this case, the RTMP URL example above will be interpreted as

```
rtmp://test.flashphoner.com:1935/live?auth=key/test
```

Setting a server application while RTMP stream publishing

While publishing RTMP stream to WCS server, a server application can be set that will be used to backend server interaction. It can be done with parameter in stream URL:

```
rtmp://host:1935/live?appKey=key1/streamName
```

Where

- host is WCS server;
- key1 is application key on WCS server;
- streamName is stream name to publish

By default, if application key parameter is not set, the standard application`flashStreamingApp`will be used.

Besides, an application can be explicitly specified as stream URL part. To do this, the following parameterinflashphoner.propertiesfile should be set

```
rtmp_appkey_source=app
```

Then application key must be set in stream URL as

```
rtmp://host:1935/key1/streamName
```

In this case, `live`is also an application name, therefore when stream is published with URL

```
rtmp://host:1935/live/streamName
```

`live` application must be defined on WCS server.

Sorenson Spark + Speex 16 kHz stream publishing

WCS server can capture RTMP stream encoded with Sorenson Spark + Speex 16kHz to FLV container. This stream can be published, for example, using ffmpeg as follows:

```
ffmpeg -re -i BigBuckBunny.flv -preset ultrafast -ar 16000 -ac 1 -acodec speex -vcodec flv -strict -2 -f flv
rtmp://test1.flashphoner.com:1935/live/test
```

Known limits

1. To handle such stream including stream recording, the stream will be transcoded to H.264 + AAC.

2. Payload types 127 for video and 97 for audio should be set in SDP when publishing such stream, for example

```
v=0
o=- 1988962254 1988962254 IN IP4 0.0.0.0
c=IN IP4 0.0.0.0
t=0 0
a=sdplang:en
m=video 0 RTP/AVP 127
a=rtpmap:127 FLV/90000
a=sendonly
m=audio 0 RTP/AVP 97 8 0
a=rtpmap:97 SPEEX/16000
a=rtpmap:8 PCMA/8000
a=rtpmap:0 PCMU/8000
a=sendonly
```

RTMP connection activity checking

In some cases, if RTMP encoder does not support Keep Alive packets sending, or Keep Alives are disabled due to another reason with the following parameter

```
keep_alive.algorithm=NONE
```

it is necessary to control RTMP connection and close it when no data was transmitted for a long time. To do this, use the following parameters.

Media traffic checking

Since build5.2.533, RTP activity checking for RTMP streams can be enabled with the following parameter inflashphoner.properties file:

```
flash_rtp_activity_enabled=true
```

Read timeout

Read timeout is set with the following settings inflashphoner.propertiesfile:

```
rtmp.server_read_socket_timeout=120
```

In this case RTMP connection will be closed if no data was received in last 120 seconds.

Write timeout

Write timeout is set with the following setting

```
rtmp.server_write_socket_timeout=120
```

In this case RTMP connection will be closed if no data was sent in last 120 seconds.

Read and write timeout

Read and write timeout is set with the following setting

```
rtmp.server_socket_timeout=120
```

In this case RTMP connection will be closed if no data was received and sent in last 120 seconds.

RTMP stream picture rotation

When publishing RTMP stream to WCS, stream picture can be rotated using RTMP metadata. It may be useful for picture handling from mobile publishers.

To turn a picture to a certain angle RTMP metadata containing ‘orientation’ field should be sent. The following orientations are supported:

Orientation value	Rotation angle, degrees
0	0
1	90
2	180
3	270

A picture is rotated clockwise.

Set up

To enable stream rotation, set the following parameter inflashphoner.propertiesfile:

```
video_filter_enable_rotate=true
```

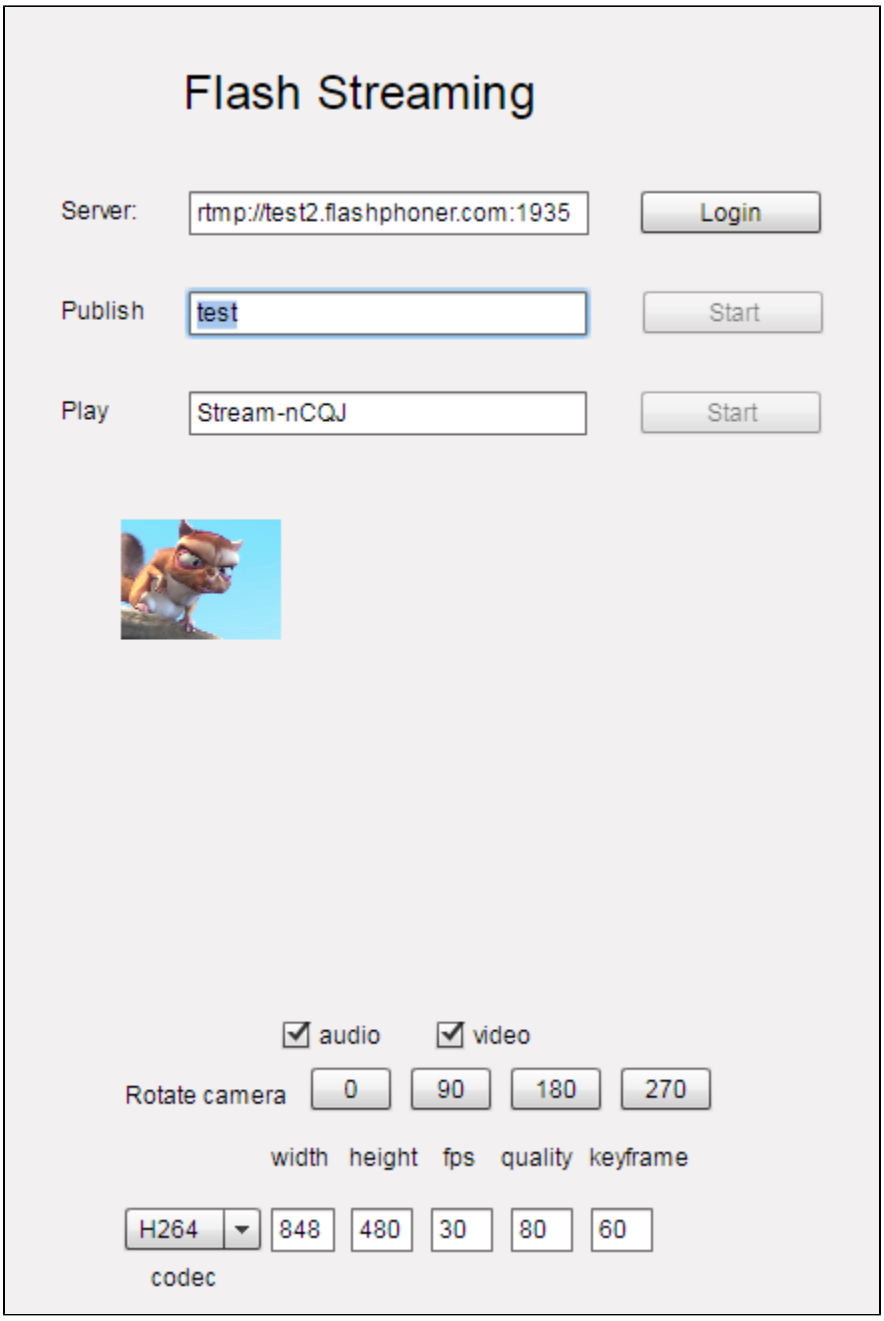
Note that stream rotation works for transcoded streams only.

Testing

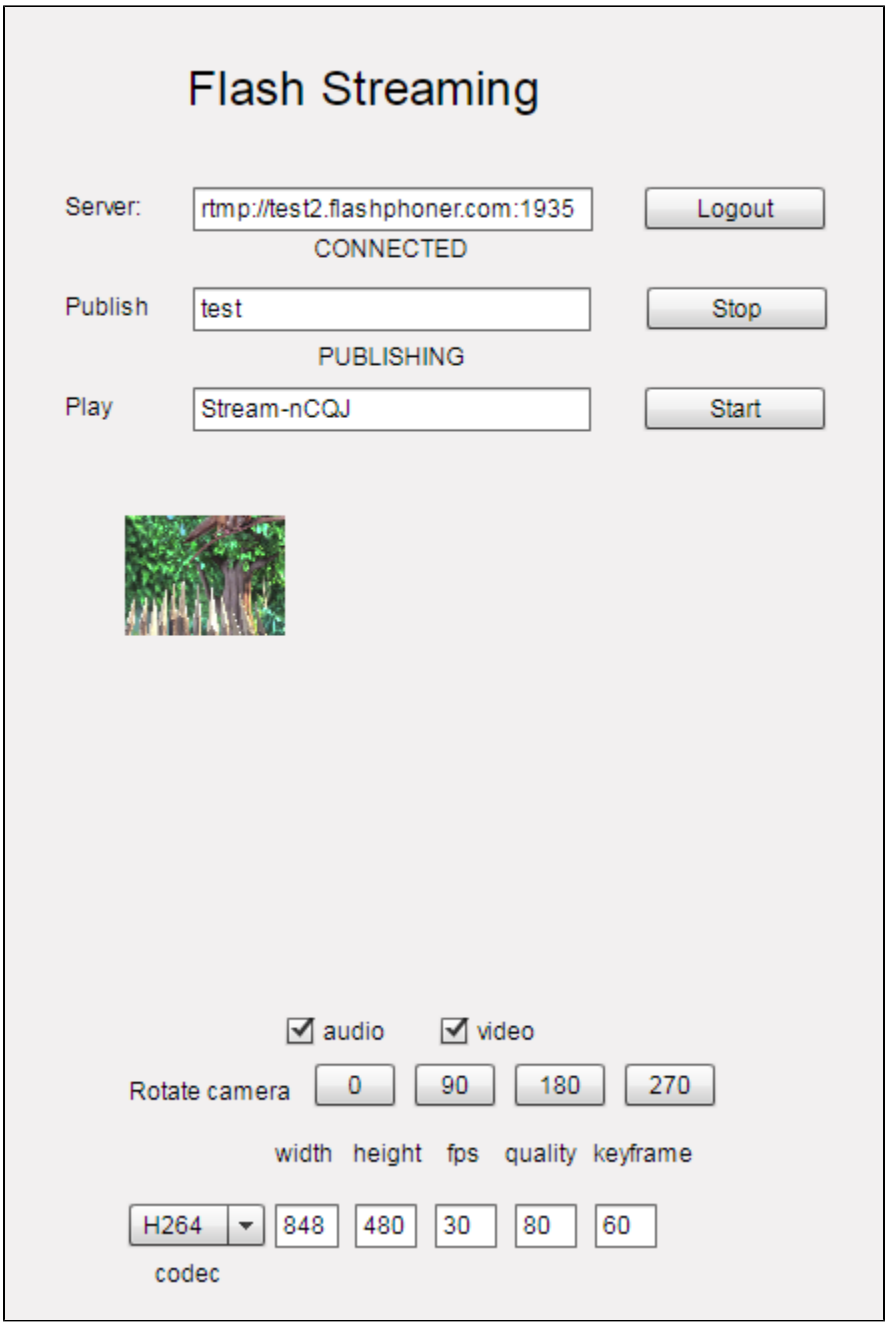
1. For test we use:

- WCS server with picture rotation enabled
- Flash Streamingapplication to publish and rotate the stream
- Playerapplication to play the stream

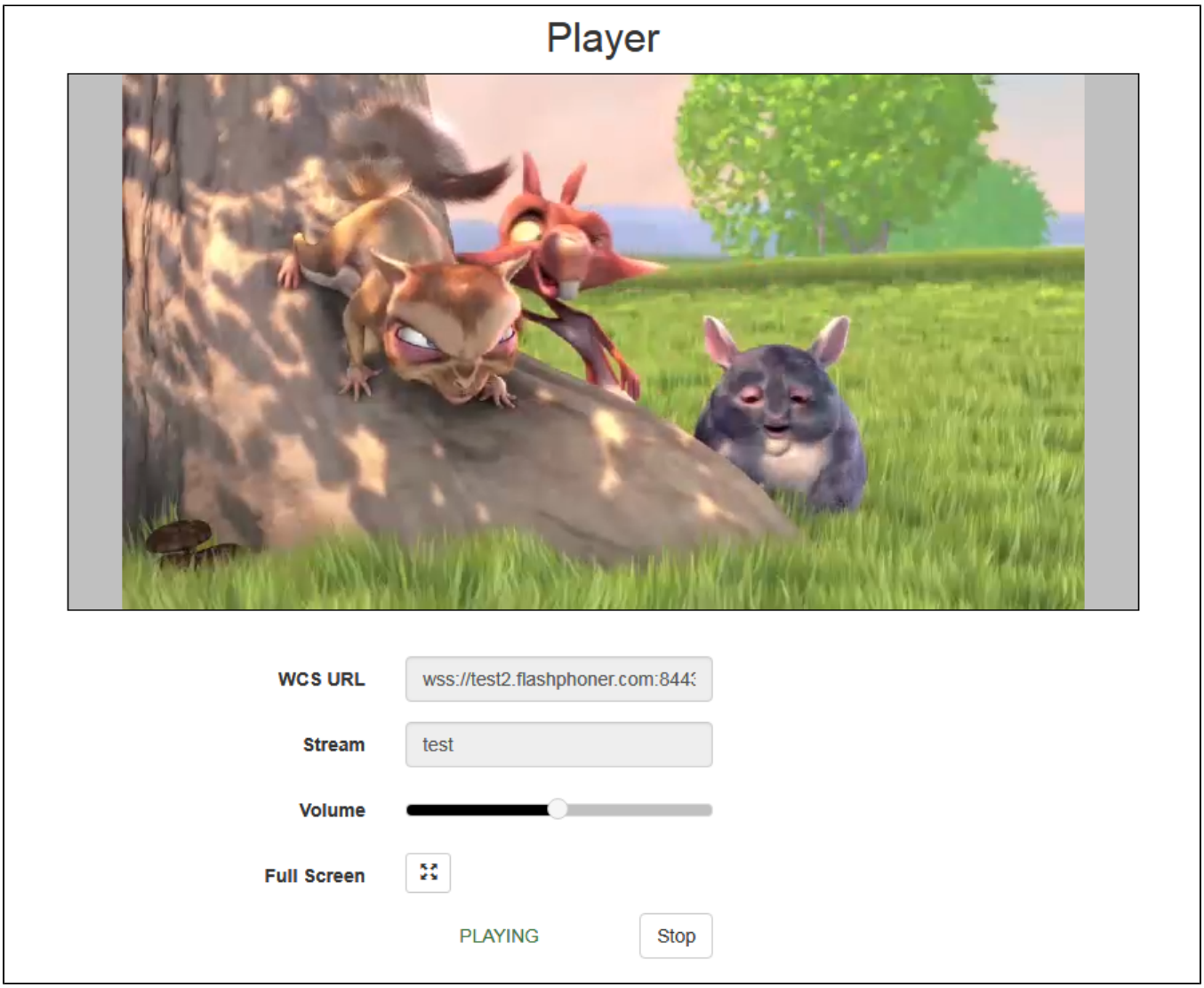
2. Open Flash Streaming application. Set stream name test and desired stream publishing parameters:



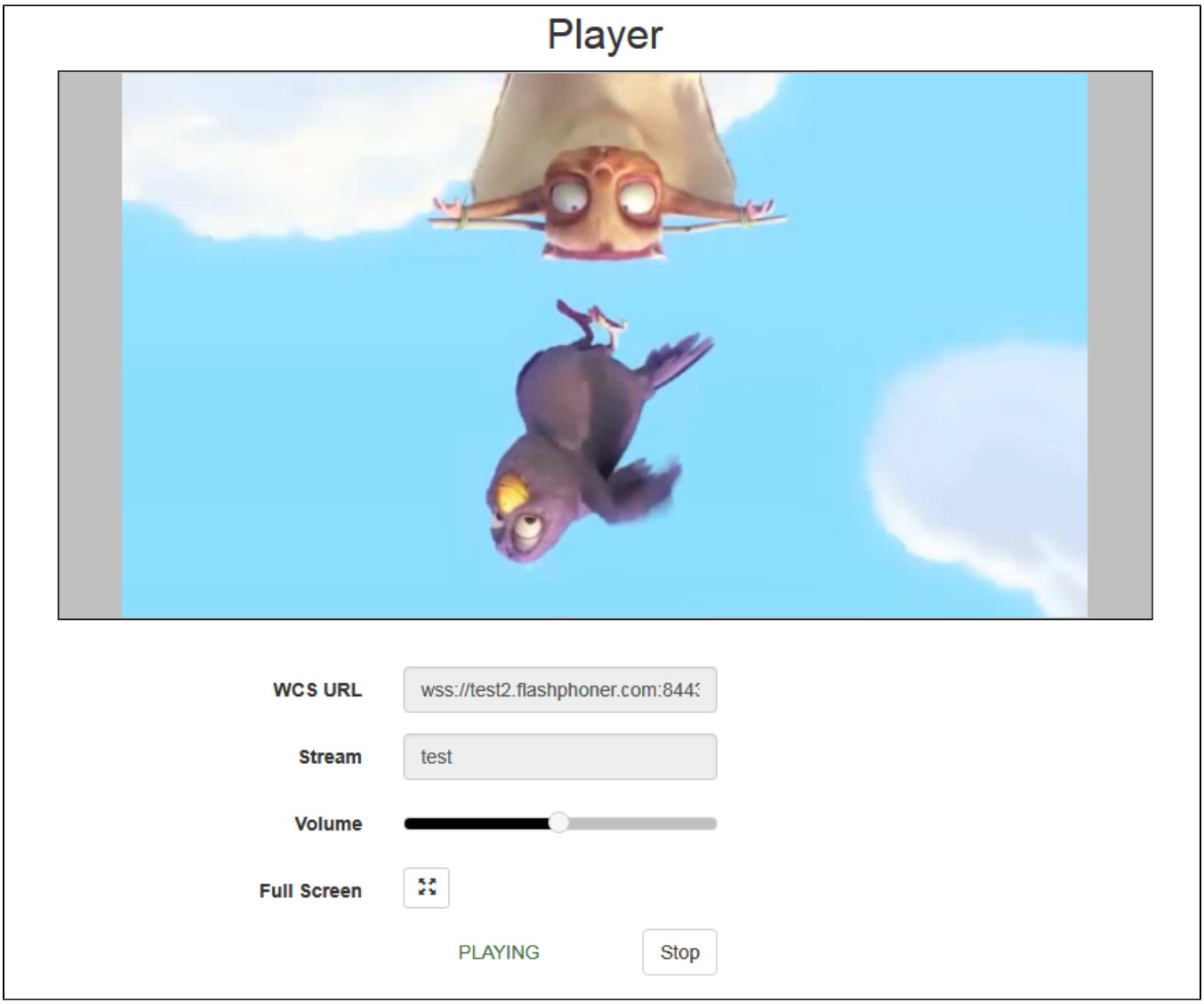
3. Click Login then Start. Stream publishing begins



4. Open Player application in anoter tab or browser, play the stream named test



5. In Flash Streaming application click 180 button in Rotate camera section. The stream rotated to 180 degrees clockwise will be displayed in Player application



To developer

Stream orientation metadata sending is implemented in Flash Streaming application as follows:

code

```
    private function rotate(degree:Number):void {
        var metaDataObj:Object = new Object();
        switch(degree) {
case 0:
    Logger.info("rotate camera to 0");
    metaDataObj.orientation = 0;
    break;
case 90:
    Logger.info("rotate camera to 90");
    metaDataObj.orientation = 1;
    break;
case 180:
    Logger.info("rotate camera to 180");
    metaDataObj.orientation = 2;
    break;
case 270:
    Logger.info("rotate camera to 270");
    metaDataObj.orientation = 3;
    break;
default:
    metaDataObj.orientation = 0;
    break;
        }
        sendMetaData(metaDataObj);
    }

    private function sendMetaData(data:Object):void{
        if (publishStream != null) {
            publishStream.send("@setDataFrame", "onMetaData", data);
        }
    }
```

Note that orientation value should be sent but not angle itself.

How to rotate stream published from ffmpeg

ffmpeg RTMP encoder allows to send orientation metadata to WCS server using command line switches:

```
ffmpeg -i input.mp4 -metadata:s:v rotate=90 -vcodec copy -acodec copy -strict -2 -f flv rtmp://test1.
flashphoner.com:1935/live/stream_ffmpeg
```

Note that ffmpeg sends orientattion value but notangle itself.

Decoding buffer size management

When RTMP stream is published from software encoder supportin hardware acceleration on NVIDIA GPU, and contains B-frames, the stream picture can twitch in some RTMP or HLS players. To workaround this, since build5.2.863the parameter was added to set the maximum decoding buffer size passed in SPS

```
h264_sps_max_dec_frame_buffering=-1
```

By default, buffer size is not limited. This eliminates picture twitching, but can inscrease latency due to excessive buffering. In this case, the buffer size can be limited to 2 frames (default value before build5.2.863)

```
h264_sps_max_dec_frame_buffering=2
```

or more frames to eliminate twitching without viewable latency.

Incoming RTMP stream buffering

RTMP stream published with high resolution and bitrate may be played non smoothly via WebRTC with freezes or low FPS if publishers channel is unstable. Incoming stream should be buffered to prevent playback issues

```
rtmp_in_buffer_enabled=true
```

Adaptive RTMP incoming buffer has the following parameters to tune:

Parameter	Description	Default value
rtmp_in_buffer_start_size	Minimum buffer volume to start, ms	300
rtmp_in_buffer_initial_size	Maxumum buffer volume, ms	2000
rtmp_in_buffer_max_bufferings_allowed	Maximum bufferings amout allowed	-1 (unlimited)
rtmp_in_buffer_polling_time	Buffer polling period, ms	100
rtmp_in_buffer_overflow_allowed_deviation	Maximum difference between minimum and maximum buffer volumes used, ms	1000
rtmp_in_buffer_overflow_deviation_window	Window size to collect the difference, ms	30000
rtmp_in_buffer_overflow_rate	Maximum buffer overflow rate	0.15
rtmp_in_buffer_clear_threshold	Clear all the data exceeding maxumum buffer size when buffer reaches the threshold, ms	30000

Stop stream buffering if stream delay grows

RTMP stream timestamps may significally differ from server time, and stream may give a delay if there is not enough systrem perfomance for software stream encoder, or if channel bandwidth decreases. Buffering of such stream will give a periodic freezes while playing it. Since build 5.2.1311 the following parameter was added to disable buffering and pass incoming traffic directly to server engine

```
rtmp_in_buffer_input_delay_threshold=0
```

By default, RTMP traffic will always be buffered if buffering is enabled. Buffering can be disabled when stream delay reaches a value defined in milliseconds

```
rtmp_in_buffer_input_delay_threshold=3000
```

In this case, buffer is cleared and goes to the `PASSTHROUGH` state. Even if stream delay becomes lower, buffer will remain in the state, and stream will not be buffered until publishing is stopped.

Detect stream parameters by metadata or media traffic

By default, a possible RTMP publishing parameters are set according to SDP settings file. Since build 5.2.1862 the following option is added to enable automatic stream publishing parameters detection by stream metadata or media traffic data

```
flash_detect_metadata_by_traffic=true
```

The option is enabled by default. In this case, WCS changes SDP according to a stream metadata received or, if there are no metadata in 1 second, according to media information from the stream media packets received.

Known issues

1. A stream containing B-frames does not play or plays with artifacts (latencies, lags)

Symptoms:

- a stream sent by the RTMP encoder does not play or plays with latencies or lags
- warnings in theclient log:

```
09:32:31,238 WARN 4BitstreamNormalizer - RTMP-pool-10-thread-5 It is B-frame!
```

Solution

- change the encoder settings so, that B-frames were not used (lower encoding profile, specify in the command line etc)

- transcode the stream, in this case there will be no B-frames in transcoded stream

2. AAC frames of type 0 are not supported by decoder and will be ignored while stream pulled playback

In this case, warnings will be displayed in theclient log:

```
10:13:06,815 WARN AAC - AudioProcessor-c6c22de8-a129-43b2-bf67-1f433a814ba9 Dropping AAC frame that starts with
0, 119056e500
```

Solution: use Fraunhofer AAC codec with the following parameter inflashphoner.propertiesfile

```
use_fdk_aac=true
```

3. When publishing and then playing and recording H264 + AAC stream video may be out of sync with sound, or no sound at all.

Symptoms: when playing H264 + AAC stream published on server, and when recordingsuch stream, sound is out of sync with video or absent

Solution:

a) set the following parameter inflashphoner.propertiesfile

```
disable_drop_aac_frame=true
```

This parameter also turns off AAC frames dropping.

b) use Fraunhofer AAC codec

```
use_fdk_aac=true
```

4. Sound may be distorted or absent when resampled to 11025 Hz

Symptoms: when H264 + AAC stream published on WCS server is played with AAC sample rate 11025 Hz, sound is distorted or absent

Solution: do not use 11025 Hz sample rate, or escape AAC sound resampling to this rate, for example, do not set this sample rate in SDP settings.

4. Some RTMP functions does not supported and will be ignored:

- FCSubscribe
- FCPublish
- FCUnpublish
- onStatus
- onUpstreamBase
- releaseStream

5. Some RTMP-encoders does not support KeepAlive.

Symptoms: disconnection occurs often while stream publishing with RTMP-encoder.

Solution: switch KeepAlive off for RTMP on the server using the following parameter inflashphoner.propertiesfile

```
keep_alive.enabled=websocket,rtmfp
```

7. When stream published with RTMP encoder is played as HLS, freezes may occur if GOP is not multiple of FPS of file published

Symptoms: freezes occur when RTMP stream is played as HLS

Solution: in RTMP encoder settings, assign GOP to value equal or multiple of FPS of file published. For example, when publishing file with FPS 25 set GOP to 50.

8.When stream published with RTMP encoder is played as WebRTC, stream audio has low quality

Symptoms: sound is good enogh for speech but not for music when RTMP is played as WebRTC

Solution: set Opus encoding bitrate as published one, for example

```
opus.encoder.bitrate=160000
```

if RTMP stream is published with audio bitrate 160 kbps

9. High CPU load when stream H264+speex is published (for example, using Flash client) with audio transcoding

Symptoms: high CPU load while audio transcoding from speex to AAC or Opus

Solution: use native speex decoder implementation

```
use_speex_java_impl=false
```

10.Stream with unsupported codecs cannot be published

Symptoms: RTMP stream with MP3 or AC3 audio is not publishing with the following warnings in server log

```
11:01:00,921 WARN         ServerHandler - RTMP-pool-15-thread-1 Codecs not supported! audio: NoCodec, video:
NoCodec
```

Solution: reencode stream to supported codecs using corresponding encoder settings