
User manual

(Media Server)

HAPPYTIME SOFTWARE CO., LIMITED

Declaration

All rights reserved. No part of this publication may be excerpted, reproduced, translated, annotated or edited, in any form or by any means, without the prior written permission of the copyright owner.

Since the product version upgrade or other reasons, this manual will subsequently be updated. Unless otherwise agreed, this manual only as a guide, this manual all statements, information, recommendations do not constitute any express or implied warranties.

www.happytimesoft.com

Table of Contents

Chapter 1 Introduction.....	4
Chapter 2 Key features.....	5
Chapter 3 Function chart	6
Chapter 4 Configuration	7
4.1 Configuration Templates	7
4.2 Configuring Node Description	11
4.2.1 <i>System parameters</i>	11
4.2.2 <i>HTTP node</i>	11
4.2.3 <i>HTTPS node</i>	12
4.2.4 <i>RTSP node</i>	13
4.2.5 <i>RTMP node</i>	15
4.2.6 <i>SRT node</i>	16
4.2.7 <i>HTTP-FLV node</i>	18
4.2.8 <i>HLS node</i>	19
4.2.9 <i>User node</i>	20
4.2.10 <i>Application node</i>	20
4.2.11 <i>Backchannel node</i>	25
Chapter 5 Data pusher.....	27
Chapter 6 RTSP over HTTP	29
Chapter 7 RTSP over Websocket	31
Chapter 8 RTP Multicast	32
Chapter 9 Audio back channel.....	33
9.1 RTSP Require- Tag.....	33
9.2 Connection setup for a bi- directional connection.....	33
9.3 Example	36
Chapter 10 Run media server	37
Chapter 11 Multiple capture devices support	38
Chapter 12 Capture application window	40
Chapter 13 Support URL parameters.....	41
Chapter 14 Support SRTP	43
14.1 SRTP for rtsp publishing	43
14.2 SRTP for rtsp playback	44

Chapter 1 Introduction

Happytime Media Server is a simple, lightweight, high-performance, and stable stream server, it outputs rtsp, rtmp, srt (Secure Reliable Transport), http-flv, hls streams.

It can stream audio and video files in various formats.

It can also stream video from camera, living screen and application windows, stream audio from audio device.

It can stream H265, H264, MP4, MJPEG video stream and G711, G722, G726, AAC, OPUS audio stream.

These streams can be received/played by standards-compliant RTMP/RTSP/RTP/http-flv/HLS media clients.

It supports rtsp/rtmp/srt proxy function.

It supports rtsp audio back channel function.

It supports rtsp over http function.

It supports rtsp over https function.

It supports rtp multicast function.

Support for data pusher function.

It developed based on C/C++, the code is stable and reliable, cross-platform porting is simple and convenient, and the code is clear and concise. The server is written to be lightweight and easy to understand, while having good performance, very low latency, video opened immediately.

Enjoying multimedia content from your computer can be a pleasant way for you to spend your free time. However, sometimes you might need to access it from various locations, such as a different computer or a handheld device, Happytime Media Server, that can help you achieve quick and efficient results.

Chapter 2 Key features

The server can transmit multiple streams concurrently

It can stream audio and video files in various formats

It can stream audio from audio device

It supports recording system sound on Windows

It can stream video from camera and living screen

It can stream video from camera and application windows

It can stream H265, H264, MP4, MJPEG video stream

It can stream G711, G722, G726, AAC, OPUS audio stream

It outputs rtsp,rtmp,srt,http-flv,hls streams

It supports rtsp over http function

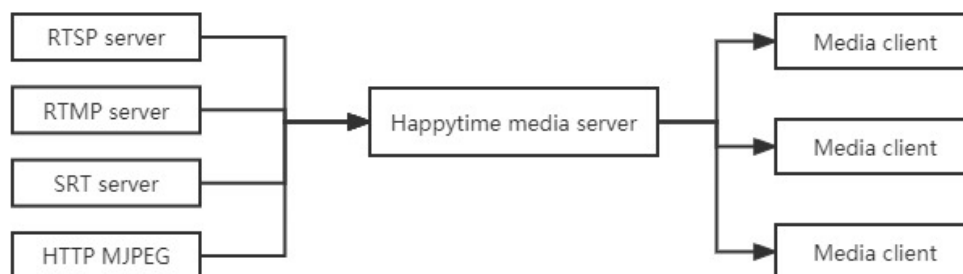
It supports rtsp over https function

It supports rtsp over websocket function

It supports rtp multicast function

It supports data pusher function

It supports RTSP/RMTP/SRT proxy function, as the following:

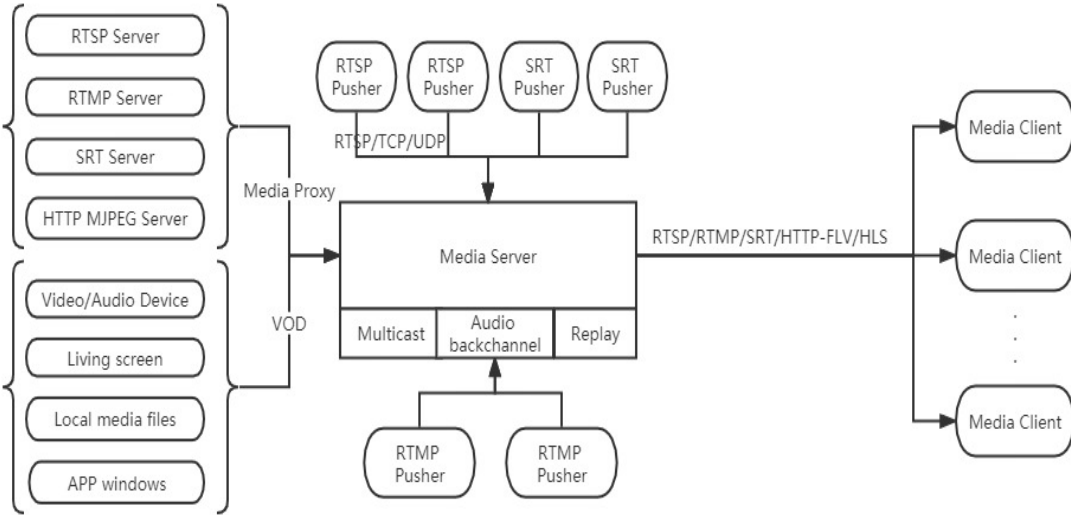


Support RTSP audio backchannel

Happytime media server comply with ONVIF backchannel specification, the url is :

<https://www.onvif.org/specs/stream/ONVIF-Streaming-Spec-v1706.pdf>

Chapter 3 Function chart



Chapter 4 Configuration

If no configuration file is specified at startup, the default configuration file `mediaserver.cfg` will be used.

4.1 Configuration Templates

```
<?xml version="1.0" encoding="utf-8"?>
<config>
  <ipv6_enable>1</ipv6_enable>
  <loop_nums>-1</loop_nums>
  <log_enable>1</log_enable>
  <log_level>1</log_level>

  <http>
    <enable>1</enable>
    <serverip></serverip>
    <serverport>80</serverport>
  </http>

  <https>
    <enable>1</enable>
    <serverip></serverip>
    <serverport>1443</serverport>
    <cert_file>ssl.ca</cert_file>
    <key_file>ssl.key</key_file>
  </https>

  <rtsp>
    <enable>1</enable>
    <serverip></serverip>
    <serverport>554</serverport>
    <multicast>0</multicast>
    <udp_base_port>22000</udp_base_port>
    <metadata>1</metadata>
```

```
<rtsp_over_http>1</rtsp_over_http>
<rtsp_over_https>1</rtsp_over_https>
<need_auth>0</need_auth>
<http_notify>
  <on_connect></on_connect>
  <on_play></on_play>
  <on_publish></on_publish>
  <on_done></on_done>
  <notify_method></notify_method>
</http_notify>
</rtsp>
```

```
<rtmp>
  <enable>1</enable>
  <serverip></serverip>
  <serverport>1935</serverport>
  <http_notify>
    <on_connect></on_connect>
    <on_play></on_play>
    <on_publish></on_publish>
    <on_done></on_done>
    <notify_method></notify_method>
  </http_notify>
</rtmp>
```

```
<srt>
  <enable>1</enable>
  <serverip></serverip>
  <serverport>8080</serverport>
  <http_notify>
    <on_connect></on_connect>
    <on_play></on_play>
    <on_publish></on_publish>
    <on_done></on_done>
```

```
        <notify_method></notify_method>
    </http_notify>
</srt>
```

```
<http-flv>
    <enable>1</enable>
    <chunked>1</chunked>
    <http_notify>
        <on_connect></on_connect>
        <on_play></on_play>
        <on_done></on_done>
        <notify_method></notify_method>
    </http_notify>
</http-flv>
```

```
<hls>
    <enable>1</enable>
    <fragment>5</fragment>
    <playlist>5</playlist>
    <cleanupdir>1</cleanupdir>
</hls>
```

```
<user>
    <username>admin</username>
    <password>admin</password>
</user>
```

```
<application>
    <name>myapp</name>
```

```
    <output>
        <url></url>
        <video>
            <codec>H264</codec>
```

```
<width></width>
<height></height>
<framerate></framerate>
<bitrate></bitrate>
</video>
<audio>
  <codec>AAC</codec>
  <samplerate>44100</samplerate>
  <channels>2</channels>
  <bitrate></bitrate>
</audio>
</output>

<proxy>
  <suffix>proxy</suffix>
  <url></url>
  <user></user>
  <pass></pass>
  <transfer>TCP</transfer>
  <ondemand>0</ondemand>
  <output>
    <video>
      <codec></codec>
      <width></width>
      <height></height>
      <framerate></framerate>
      <bitrate></bitrate>
    </video>
    <audio>
      <codec>AAC</codec>
      <samplerate>44100</samplerate>
      <channels>2</channels>
      <bitrate></bitrate>
    </audio>
```

```
        </output>
    </proxy>

</application>
</config>
```

4.2 Configuring Node Description

4.2.1 System parameters

<ipv6_enable>

Indicates whether IPv6 is enabled, 0-disable, 1-enable.

Note: If the server does not specify a server ip in **<serverip>** and the **<ipv6_enable>** is 1, and the server has an IPv6 address, the client can connect to the server through the IPv6 address.

<loop_nums>

When streaming local media files, specify the number of loop playback, -1 means infinite loop.

<log_enable>

Whether enable the log function, 0-disable, 1-enable.

<log_level>

The log level:

TRACE	0
DEBUG	1
INFO	2
WARN	3
ERROR	4
FATAL	5

4.2.2 HTTP node

<enable>

Whether to enable HTTP server, 0-disable, 1-enable.

<serverip>

Specify the IP address of http server, if not specified, it will listen on all interfaces.

<serverport>

Specify the port of http server, the default is 80.

Note: On Linux systems, ports below 1024 are reserved by the system and require root privileges to be used.

4.2.3 HTTPS node

<enable>

Whether to enable HTTPS server, 0-disable, 1-enable.

<serverip>

Specify the IP address of https server, if not specified, it will listen on all interfaces.

<serverport>

Specify the port of https server, the default is 443.

Note: On Linux systems, ports below 1024 are reserved by the system and require root privileges to be used.

<cert_file>

Specify the HTTPS server certificate file.

<key_file>

Specify the HTTPS server key file.

Note: The certificate file ssl.ca and key file ssl.key provided by default are self signed local hosts certificates, only for testing purposes (browsers may pop up untrusted certificate warnings), and cannot be used in formal deployment environments.

4.2.4 RTSP node

<enable>

Whether to enable rtsp server, 0-disable, 1-enable.

<serverip>

Specify the IP address of RTSP server, if not specified, it will listen on all interfaces.

<serverport>

Specify the port of RTSP server, the default is 554.

Note: On Linux systems, ports below 1024 are reserved by the system and require root privileges to be used.

<multicast>

Whether to enable rtp multicast function, 0-disable, 1-enable.

<udp_base_port>

UDP media transmission base port, RTSP over UDP mode assign UDP port on this base port.

Each rtsp session needs to assign 8 UDP ports, video RTP/RTCP port, audio RTP/RTCP port, METADATA stream RTP/RTCP port and audio back-channel RTP/RTCP port.

<metadata>

Whether to enable the meta data stream, 0-disable, 1-enable.

<rtsp_over_http>

Whether to enable rtsp over http function, 0-disable, 1-enable.

Need to enable HTTP service, http port is HTTP service port.

<rtsp_over_https>

Whether to enable rtsp over https function, 0-disable, 1-enable.

Need to enable HTTPS service, https port is HTTPS service port.

<need_auth>

Whether enable the user authentication function, 0-disable, 1-enable.

<http_notify>

<on_connect> :

Sets HTTP connection callback. When clients issues connect command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then RTSP session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

```
protocol=rtsp
call=connect
addr - client IP address
app - application name
url - URL requested by the client
name - stream name
clientid - rtsp session id
```

<on_play> :

Sets HTTP play callback. Each time a clients issues play command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then RTSP session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

```
protocol=rtsp
call=play
addr - client IP address
app - application name
url - URL requested by the client
name - stream name
clientid - rtsp session id
```

<on_publish> :

The same as on_play above with the only difference that this node sets

callback on publish command. Instead of remote pull push is performed in this case.

<on_done> :

Sets play/publish terminate callback. All the above applies here. However HTTP status code is not checked for this callback.

<notify_method>

Sets HTTP method for notifications. Default is POST with application/x-www-form-urlencoded content type.

Support GET and POST method.

4.2.5 RTMP node

<enable>

Whether to enable rtmp server, 0-disable, 1-enable.

<serverip>

Specify the IP address of RTMP server, if not specified, it will listen on all interfaces.

<serverport>

Specify the port of RTMP server binding, the default is 1935.

Note: On Linux systems, ports below 1024 are reserved by the system and require root privileges to be used.

<http_notify>

<on_connect> :

Sets HTTP connection callback. When clients issues connect command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then RTMP session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

protocol=rtmp

call=connect

addr - client IP address

app - application name

```
tcurl - tcUrl  
name - stream name  
clientid - rtmp session id
```

<on_play> :

Sets HTTP play callback. Each time a clients issues play command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then RTMP session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

```
protocol=rtmp  
call=play  
addr - client IP address  
app - application name  
tcurl - tcUrl  
name - stream name  
clientid - rtmp session id
```

<on_publish> :

The same as on_play above with the only difference that this node sets callback on publish command. Instead of remote pull push is performed in this case.

<on_done> :

Sets play/publish terminate callback. All the above applies here. However HTTP status code is not checked for this callback.

<notify_method>

Sets HTTP method for notifications. Default is POST with application/x-www-form-urlencoded content type.

Support GET and POST method.

4.2.6 SRT node

<enable>

Whether to enable srt server, 0-disable, 1-enable.

<serverip>

Specify the IP address of SRT server, if not specified, it will listen on all

interfaces.

<serverport>

Specify the port of SRT server, the default is 8080.

Note: On Linux systems, ports below 1024 are reserved by the system and require root privileges to be used.

<http_notify>

<on_connect> :

Sets HTTP connection callback. When clients issues connect command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then SRT session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

```
protocol=srt
call=connect
addr - client IP address
app - application name
sid - stream id
name - stream name
clientid - srt session id
```

<on_play> :

Sets HTTP play callback. Each time a clients issues play command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then SRT session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

```
protocol=srt
call=play
addr - client IP address
app - application name
sid - stream id
```

name - stream name
clientid - srt session id

<on_publish> :

The same as on_play above with the only difference that this node sets callback on publish command. Instead of remote pull push is performed in this case.

<on_done> :

Sets play/publish terminate callback. All the above applies here. However HTTP status code is not checked for this callback.

<notify_method>

Sets HTTP method for notifications. Default is POST with application/x-www-form-urlencoded content type.

Support GET and POST method.

4.2.7 HTTP-FLV node

<enable>

Whether to enable http-flv server, 0-disable, 1-enable.

Need to enable HTTP service.

<chunked>

http chunked transfer encoding, 1-on, 0-off.

<http_notify>

<on_connect> :

Sets HTTP connection callback. When clients issues connect command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then HTTP-FLV session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

protocol=httpflv
call=connect
addr - client IP address
app - application name
name - stream name

clientid - httpflv session id

<on_play> :

Sets HTTP play callback. Each time a clients issues play command an HTTP request is issued and command processing is suspended until it returns result code. If HTTP 200 code is returned then HTTP-FLV session continues.

HTTP request receives a number of arguments. POST method is used with application/x-www-form-urlencoded MIME type. The following arguments are passed to caller:

protocol=httpflv
call=play
addr - client IP address
app - application name
name - stream name
clientid - httpflv session id

<on_done> :

Sets play terminate callback. All the above applies here. However HTTP status code is not checked for this callback.

<notify_method>

Sets HTTP method for notifications. Default is POST with application/x-www-form-urlencoded content type.

Support GET and POST method.

4.2.8 HLS node

<enable>

Whether to enable hls server, 0-disable, 1-enable.

Need to enable HTTP service.

Note: HLS only supports live streaming, not on-demand streaming.

The form of the HLS stream address is as follows:

http://ip:port/myapp/live.m3u8

ip:port is the ip and port of the HTTP service.

myapp is the <name> tag under the <application> tag.

live is rtsp / rtmp / srt push stream or rtsp / rtmp / srt proxy stream, such as:

rtsp://ip:port/myapp/live

rtmp://ip:port/myapp/live

rtsp://ip:port/myapp/proxy

<fragment>

Specify the fragmentation duration, in second.

<playlist>

Specify the maximum number of ts files in the m3u8 file, the default is 5, the minimum is 3.

<cleanupdir>

Cleanup ts file directory at startup.

4.2.9 User node

<user> : Specify the login username password, it can configure multiple nodes

<username>

The login username.

<password>

The login password.

4.2.10 Application node

<application> : it can configure multiple nodes

<name>: Application Name

The rtsp stream address is :

rtsp://[serverip]:[serverport]/[application-name]/FILENAME

The rtmp stream address is :

rtmp://[serverip]:[serverport]/[application-name]/FILENAME

The srt stream address is :

srt://[serverip]:[serverport]?streamid=[application-name]/FILENAME

The http-flv stream address is :

http://[serverip]:[serverport]/[application-name]/FILENAME

The [application-name] is <name> tag value.

4.2.10.1 Output node

<output> : Specify the audio and video output parameters, it can configure multiple nodes

<url>

Match URL address, it can be filename, or file extension name. Such as:

screenlive : match live screen stream

videodevice : match camera video stream

*.mp4 : match all mp4 media file

sample.flv : match sample.flv file

If not config this node, it will match all url as the audio/video default output parameters.

The match order from top to bottom, therefore the default output configuration should be placed in the last.

<video> : Specify the video output parameters

<codec>

Specify the video stream codec, it can specify the following value:

H264 : output H264 video stream

H265 : output H265 video stream

MP4: output MP4 video stream

JPEG: output MJPEG video stream

Note : RTMP, HTTP-FLV services only support video codec H264, H265.

<width>

Specify the output video width, If 0 use the original video width (live screen stream use the screen width, camera stream use the default width)

<height>

Specify the output video height, If 0 use the original video height (live screen stream use the screen height, camera stream use the default height)

<framerate>

Specify the output video framerate, If 0 use the original video framerate (live screen use the default value 15, camera stream use the

default value 25)

<bitrate>

Specify the output video bit rate, if 0, automatically calculate the output bit rate, the unit is kb/s.

Note: This parameter is valid only if encoding is required (eg screenlive, videodevice) or if transcoding is required.

<audio> : Specify the audio output parameters

<codec>

Specify the audio stream codec, it can specify the following value:

G711A: output G711 a-law audio stream

G711U: output G711 mu-law audio stream

G722: output G726 audio stream

G726: output G726 audio stream

AAC: output AAC audio stream

OPUS: output OPUS audio stream

Note : RTMP, HTTP-FLV services only support audio codec AAC, G711A, G711U.

<samplerate>

Specify the audio sample rate, it can specify the following values:

8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000

If 0 use the original audio sample rate (audio device stream use the default value 8000)

<channels>

Specify the audio channel number, 1 is mono, 2 is stereo

If 0 use the original audio channel number (audio device stream use the default value 2)

Note : G726 only support mono.

<bitrate>

Specify the output video bit rate, if 0, automatically calculate the output bit rate, the unit is kb/s.

Note: This parameter is valid only if encoding is required (eg screenlive, videodevice) or if transcoding is required.

4.2.10.2 Proxy node

<proxy> : Specify the proxy parameters, it can configure multiple nodes

<suffix>

Specify the stream suffix, you can play the proxy stream from:

rtsp://[serverip]:[serverport]/[application-name]/[suffix]

rtmp://[serverip]:[serverport]/[application-name]/[suffix]

srt://[serverip]:[serverport]?streamid=[application-name]/[suffix]

http://[serverip]:[serverport]/[application-name]/[suffix]

http://[serverip]:[serverport]/[application-name]/[suffix].m3u8

<url>

The original rtsp/rtmp/srt stream address or http mjpeg stream address.

<user> <pass>

Specify the original rtsp/rtmp/srt stream or http mjpeg stream address login user and password information.

<transfer>

Specify the rtsp client transfer protocol:

TCP: rtsp client uses RTP over TCP

UDP: rtsp client uses RTP over UDP

MULTICAST: rtsp client uses multicast

<ondemand>

Connect on demand, 1-Connect when needed, 0-Always keep connected.

<output>

Specify the stream output parameter. If the parameter does not appear, use the parameters of the original RTSP/RTMP/SRT/HTTP MJPEG stream. If it appears and the configured parameters are inconsistent with the parameters of the original RTSP/RTMP/SRT/HTTP MJPEG stream, then the transcode output is performed.

The child nodes under this node are consistent with the meaning of the <output> node.

4.2.10.3 Pusher node

<pusher> : Specify the data pusher parameters, it can configure multiple nodes

<suffix>

Specify the stream suffix, you can play the pusher stream from:

rtsp://[serverip]:[serverport]/[application-name]/[suffix]

rtmp://[serverip]:[serverport]/[application-name]/[suffix]

srt://[serverip]:[serverport]?streamid=[application-name]/[suffix]

http://[serverip]:[serverport]/[application-name]/[suffix]

http://[serverip]:[serverport]/[application-name]/[suffix].m3u8

<video> : Specify the the input video data parameters

<codec>

Specify the video codec, it can specify the following value:

H264 : H264 video stream

H265 : H265 video stream

JPEG: MJPEG video stream

<audio> : Specify the input audio data parameters

<codec>

Specify the audio codec, it can specify the following value:

G711A: G711 a-law audio stream

G711U: G711 mu-law audio stream

G722: G726 audio stream

G726: G726 audio stream

OPUS: OPUS audio stream

<samplerate>

Specify the audio sample rate, it can specify the following values:

8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000

<channels>

Specify the audio channel number, 1 is mono, 2 is stereo

Note : G726 only support mono.

<transfer>: Specify the data transfer parameters

<mode>: Specify the data transfer protocol, it can specify the following

value:

TCP: use TCP connection to transfer the data

UDP: use UDP connection to transfer the data

RTSP: use RTSP connection to transfer the data, it support FFmpeg rtsp pusher.

<ip>: Specified data receiving IP address, if there is no configuration, the default IP address is used.

<vport>: Specify the video data receiving port

<aport>: Specify the audio data receiving port

<output>

Specify the stream output parameter. If the parameter does not appear, use the parameters of the original pusher stream. If it appears and the configured parameters are inconsistent with the parameters of the original pusher stream, then the transcode output is performed.

The child nodes under this node are consistent with the meaning of the <output> node.

4.2.11 Backchannel node

<backchannel> : Specify the audio back channel parameters

<codec>

Specify the audio back channel stream codec, it can specify the following value:

G711A: G711 a-law audio stream

G711U: G711 mu-law audio stream

G722: G726 audio stream

G726: G726 audio stream

OPUS: OPUS audio stream

<samplerate>

Specify the audio back channel sample rate, it can specify the following values:

8000, 11025, 12000, 16000, 22050, 24000, 32000, 44100, 48000

If 0 use the default value 8000

<channels>

Specify the audio channel number, 1 is mono, 2 is stereo

If 0 use the default value 1

Note : G726 only support mono.

Chapter 5 Data pusher

Data pusher means that media server receives external data sources and then sends them out as RTSP/RTMP/SRT/HTTP-FLV/HLS streams.

The data pusher support RTSP/RTMP/SRT mode.

If it is RTSP mode, it supports standard RTSP push stream, such as FFMPEG rtsp pusher.

FFMPEG rtsp over UDP:

```
ffmpeg -re -i test.mp4 -vcodec libx264 -acodec copy -preset ultrafast -f rtsp
rtsp://[serverip]:[serverport]/[application-name]/live
```

FFMPEG rtsp over TCP:

```
ffmpeg -re -i test.mp4 -vcodec libx264 -acodec copy -preset ultrafast -f rtsp
-rtsp_transport tcp rtsp://[serverip]:[serverport]/[application-name]/live
```

If it is RTMP mode, it supports standard RTMP push stream, such as FFMPEG rtmp pusher.

```
ffmpeg -re -i test.mp4 -vcodec libx264 -acodec aac -f flv
rtmp://[serverip]:[serverport]/[application-name]/live
```

If it is SRT mode, it supports standard SRT push stream, such as FFMPEG srt pusher.

You can push camera live stream by FFMPEG. Please download ffmpeg sourcecode from <https://github.com/FFmpeg/FFmpeg>, then compile FFMPEG with `--enable-libsrt` use ffmpeg to push file stream with SRT:

```
ffmpeg -re -i test.mp4 -vcodec libx264 -acodec aac -g 30 -pkt_size 1316
-flush_packets 0 -f mpegts
"srt://[serverip]:[serverport]?streamid=[application-name]/live,m=publish"
```

play the SRT stream with ffplay:

```
ffplay -fflags nobuffer -i
"srt://[serverip]:[serverport]?streamid=[application-name]/live"
```

The corresponding stream address is:

Rtsp stream : rtsp://[serverip]:[serverport]/[application-name]/live

Rtmp stream : rtmp://[serverip]:[serverport]/[application-name]/live

SRT stream : srt://[serverip]:[serverport]?streamid=[application-name]/live

HTTP-FLV stream: http://[serverip]:[serverport]/[application-name]/live

HLS stream : http://[serverip]:[serverport]/[application-name]/live.m3u8

Chapter 6 RTSP over HTTP

The key of RTSP over HTTP is to allow RTSP packets to communicate via HTTP port.

We know that the standard port of RTSP is 554, but due to various security policy configurations such as firewalls, there may be restrictions when the client accesses port 554, which prevents the normal transmission of RTSP packets.

But the HTTP port (port 80) is generally open, so there is the idea of letting RTSP packets pass through port 80, namely RTSP over HTTP

The details of RTSP over HTTP are as follows:

First, the client opens two socket connect to the rtsp server HTTP ports. We call these two sockets "data socket" and "command socket".

Step 1. The client sends an HTTP GET command through the "data socket" to request an RTSP connection.

Step 2. The server responds to the HTTP GET command through the "data socket" and responds with success/failure.

Step 3. The client creates a "command socket" and sends an HTTP POST command through the "command socket" to establish an RTSP session.

At this point, the auxiliary function of HTTP is completed, and the server does not return the client's HTTP POST command. Next is the standard process of RTSP on the HTTP port, but it needs to be completed through two sockets. The "command socket" is only responsible for sending, and the "data socket" is only responsible for receiving.

Step 4. The client sends RTSP commands (BASE64 encoding) through the "command socket".

Step 5. The server responds to the RTSP command (in plain text) through the "data socket".

Step 6. Repeat Step4-Step5 until the client sends the RTSP PLAY command and the server responds to the RTSP PLAY command.

Step 7. The server transmits audio and video data to the client through the "data socket"

After the data exchange is complete...

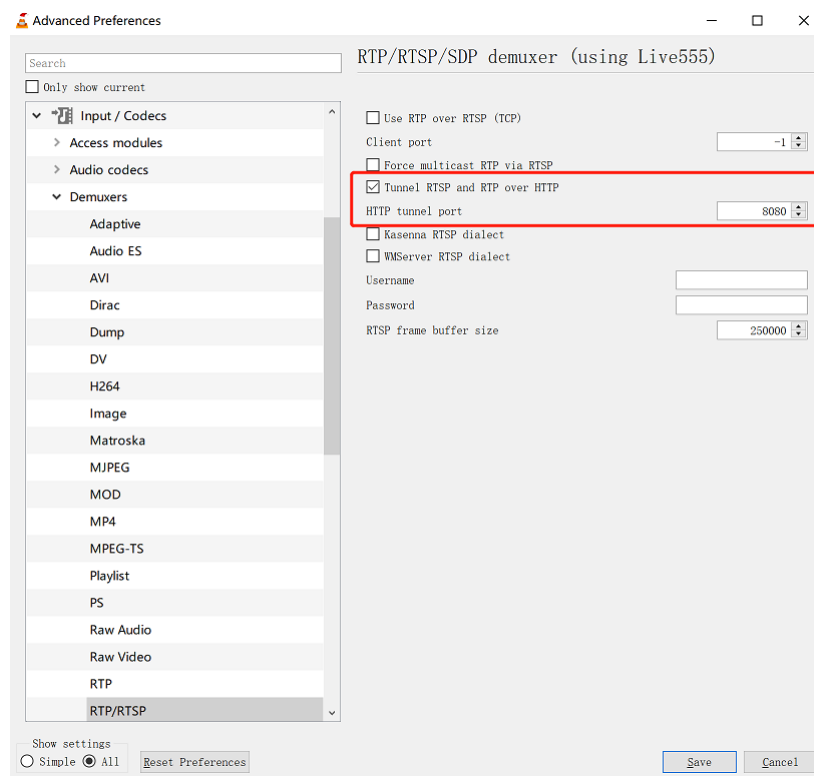
Step 8. The client sends the RTSP TEARDOWN command (BASE64 encoding and) through the "command socket"

Step 9. The server responds to the RTSP TEARDOWN command (in plain text) through

the "data socket".

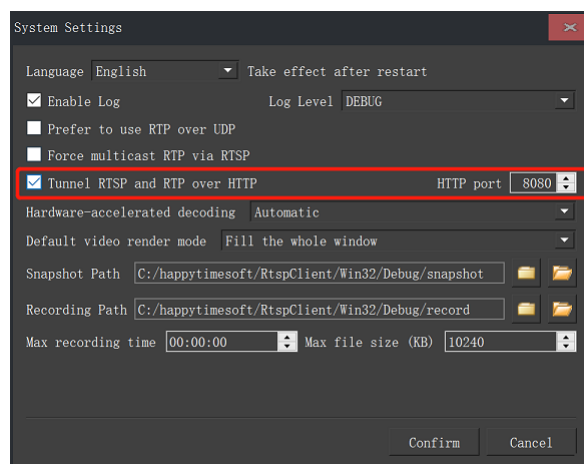
Step 10. Close the two sockets.

VLC supports RTSP over HTTP, the settings as the follows:



Happytime rtsp client

(<http://happytimesoft.com/products/rtsp-client/index.html>) supports RTSP over HTTP, The setting as the following:



Happytime rtsp client also supports rtsp streams starting with http:// or https://.

If it starts with http://, it is considered to be a rtsp over http stream.

If it starts with https://, it is considered to be a rtsp over https stream.

Chapter 7 RTSP over Websocket

First establish an HTTP connection, and then upgrade to the websocket protocol, RTSP over websocket protocol upgrade process:

C-->S:

GET /websocket HTTP/1.1 Host: 192.168.3.27

Upgrade: websocket Connection: Upgrade

Sec-WebSocket-Key: KSO+hOFs1q5SkEnx8bvp6w== Origin: http://192.168.3.27

Sec-WebSocket-Protocol: rtsp.onvif.org Sec-WebSocket-Version: 13

S-->C:

HTTP/1.1 101 Switching Protocols Upgrade: websocket

Connection: Upgrade

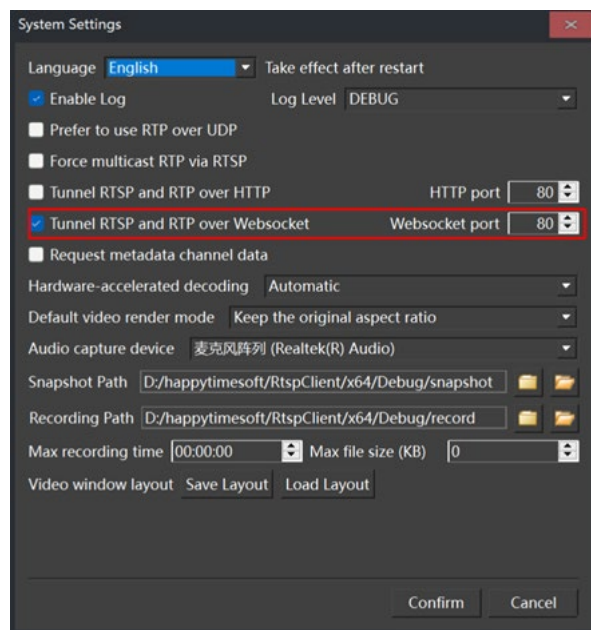
Sec-WebSocket-Accept: G/cEt4HtsYEnP0MnSVkKRk459gM= Sec-WebSocket-Protocol: rtsp.onvif.org

Sec-WebSocket-Version: 13

After the protocol upgrade is successful, perform normal rtsp protocol exchange, and send and receive data through websocket connection.

Happytime rtsp client

(<http://happytimesoft.com/products/rtsp-client/index.html>) supports RTSP over Websocket, The setting as the following:



Happytime rtsp client also supports rtsp streams starting with ws://.

If it starts with ws://, it is considered to be a rtsp over websocket stream.

Chapter 8 RTP Multicast

To enable the rtp multicast function, it need to specify the <multicast> to 1 in the configuration file.

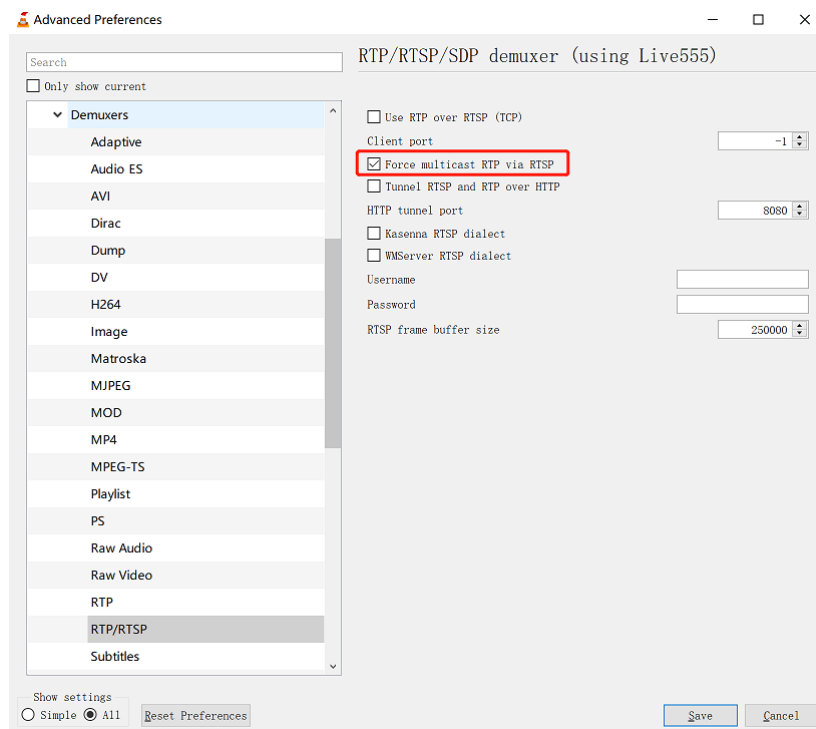
The rtsp server does not support the configuration of multicast addresses.

Different rtsp stream addresses use multicast, randomly assigned multicast addresses starting with 232.

Different rtsp sessions use rtp multicast to play the same rtsp stream, using the same multicast address. Only the first rtsp session sends audio and video data, and subsequent sessions refer to the first rtsp session.

The rtp multicast stream address is the same as the other rtsp stream address.

Use VLC to test rtp multicast, use the following settings:



Chapter 9 Audio back channel

The backchannel connection handling is done using RTSP [RFC 2326]. Therefore a mechanism is introduced which indicates that a client wants to built up a backchannel connection. RTSP provides feature-tags to deal with such functionality additions. A device that supports bi-directional connections (e.g audio or metadata connections) shall support the introduced RTSP extensions.

9.1 RTSP Require- Tag

The RTSP standard [RFC 2326] can be extended by using additional headers objects. For that purpose a Require tag is introduced to handle special functionality additions (see [RFC 2326], 1.5 Extending Rtp and 12.32 Require).

The Require-tag is used to determine the support of this feature. This header shall be included in any request where the server is required to understand that feature to correctly perform the request.

A device that supports backchannel and signals Audio output support via the AudioOutputs capability shall understand the backchannel tag:

www.onvif.org/ver20/backchannel

An RTSP client that wants to built up an RTSP connection with a data backchannel shall include the Require header in its requests.

9.2 Connection setup for a bi- directional connection

A client shall include the feature tag in it's DESCRIBE request to indicate that a bidirectional data connection shall be established.

A server that understands this Require tag shall include an additional media stream in its SDP file as configured in its Media Profile.

An RTSP server that does not understand the backchannel feature tag or does not support bidirectional data connections shall respond with an error code 551 Option not supported according to the RTSP standard. The client can then try to establish an RTSP connection without backchannel.

A SDP file is used to describe the session. To indicated the direction of the media data the server shall include the a=sendonly in each media section representing media being sent from the client to the server and a=recvonly attributes in each media section representing media being sent from the server to the client.

The server shall list all supported decoding codecs as own media section and

the client chooses which one is used. The payload type and the encoded bitstream shall be matched with one of the `a=rtpmap` fields provided by the server so that the server can properly determine the audio decoder.

Example 1: Server without backchannel support:

```
Client - Server:    DESCRIBE rtsp://192.168.0.1 RTSP/1.0
                   Cseq: 1
                   User-Agent: ONVIF Rtsp client
                   Accept: application/sdp
                   Require: www.onvif.org/ver20/backchannel

Server - Client:    RTSP/1.0 551 Option not supported
                   Cseq: 1
                   Unsupported: www.onvif.org/ver20/backchannel
```

Example 2: Server with Onvif backchannel support:

```
Client - Server:    DESCRIBE rtsp://192.168.0.1 RTSP/1.0
                   Cseq: 1
                   User-Agent: ONVIF Rtsp client
                   Accept: application/sdp
                   Require: www.onvif.org/ver20/backchannel

Server - Client:    RTSP/1.0 200 OK
                   Cseq: 1
                   Content-Type: application/sdp
                   Content-Length: xxx

                   v=0
                   o= 2890842807 IN IP4 192.168.0.1
                   s=RTSP Session with audiobackchannel
                   m=video 0 RTP/AVP 26
                   a=control:rtsp://192.168.0.1/video
                   a=recvonly
                   m=audio 0 RTP/AVP 0
                   a=control:rtsp://192.168.0.1/audio
                   a=recvonly
                   m=audio 0 RTP/AVP 0
                   a=control:rtsp://192.168.0.1/audioback
                   a=rtpmap:0 PCMU/8000
                   a=sendonly
```

This SDP file completely describes the RTSP session. The Server gives the client its control URLs to setup the streams.

In the next step the client can setup the sessions:

```
Client - Server:      SETUP rtsp://192.168.0.1/video RTSP/1.0
                      Cseq: 2
                      Transport: RTP/AVP;unicast;client_port=4588-4589

Server - Client:      RTSP/1.0 200 OK
                      Cseq: 2
                      Session: 123124;timeout=60
                      Transport:RTP/AVP;unicast;client_port=4588-4589;
                      server_port=6256-6257

Client - Server:      SETUP rtsp://192.168.0.1/audio RTSP/1.0
                      Cseq: 3
                      Session: 123124
                      Transport: RTP/AVP;unicast;client_port=4578-4579

Server - Client:      RTSP/1.0 200 OK
                      Cseq: 3
                      Session: 123124;timeout=60
                      Transport:RTP/AVP;unicast;client_port=4578-4579;
                      server_port=6276-6277

Client - Server:      SETUP rtsp://192.168.0.1/audioback RTSP/1.0
                      Cseq: 4
                      Session: 123124
                      Transport: RTP/AVP;unicast;client_port=6296-6297
                      Require: www.onvif.org/ver20/backchannel

Server - Client:      RTSP/1.0 200 OK
                      Cseq: 4
                      Session: 123124;timeout=60
                      Transport:RTP/AVP;unicast;client_port=6296-6297;
                      server_port=2346-2347
```

The third setup request establishes the audio backchannel connection.
In the next step the client starts the session by sending a PLAY request.

```
Client - Server:      PLAY rtsp://192.168.0.1 RTSP/1.0
                      Cseq: 5
                      Session: 123124
                      Require: www.onvif.org/ver20/backchannel

Server - Client:      RTSP/1.0 200 OK
                      Cseq: 5
                      Session: 123124;timeout=60
```

After receiving the OK response to the PLAY request the client MAY start sending audio data to the server. It shall not start sending data to the server before it has received the response.

The Require-header indicates that a special interpretation of the PLAY command is necessary. The command covers both starting of the video and audio stream from NVT to the client and starting the audio connection from client to server.

To terminate the session the client sends a TEARDOWN request.

```
Client - NVT:      TEARDOWN rtsp://192.168.0.1 RTSP/1.0
                  Cseq: 6
                  Session: 123124
                  Require: www.onvif.org/ver20/backchannel

NVT - Client:      RTSP/1.0 200 OK
                  Cseq: 6
                  Session: 123124
```

9.3 Example

Server with Onvif backchannel support (with multiple decoding capability)

If a device supports multiple audio decoders as backchannel, it can signal such capability by listing multiple a=rtpmap fields illustrated as follows.

```
Client - Server:    DESCRIBE rtsp://192.168.0.1 RTSP/1.0
                  Cseq: 1
                  User-Agent: ONVIF Rtsp client
                  Accept: application/sdp
                  Require: www.onvif.org/ver20/backchannel

Server - Client:    RTSP/1.0 200 OK
                  Cseq: 1
                  Content-Type: application/sdp
                  Content-Length: xxx

                  v=0
                  o= 2890842807 IN IP4 192.168.0.1
                  s=RTSP Session with audiobackchannel
                  m=video 0 RTP/AVP 26
                  a=control:rtsp://192.168.0.1/video
                  a=recvonly
                  m=audio 0 RTP/AVP 0
                  a=control:rtsp://192.168.0.1/audio
                  a=recvonly
                  m=audio 0 RTP/AVP 0 97 98 99 100
                  a=control:rtsp://192.168.0.1/audioback
                  a=rtpmap:0 PCMU/8000
                  a=rtpmap:97 G726-16/8000
                  a=rtpmap:98 G726-24/8000
                  a=rtpmap:99 G726-32/8000
                  a=rtpmap:100 G726-40/8000
                  a=sendonly
```

Chapter 10 Run media server

The server is a console application.

Windows: to run the server, simply type "mediaserver".

Linux: to run the server, type "./start.sh", on linux platform, media server run as daemon by default.

media server supports the following command line options:

`-c config` specify the configuration file

`-c` option specifies the configuration file, if not specified, the default configuration mediaserver.cfg is used.

`-l [device|videodevice|audiodevice|window]`

`-l device` list available video and audio capture device

`-l videodevice` list available video capture device

`-l audiodevice` list available audio capture device

`-l window` list available application window

Below is sample output of `-l device`:

mediaserver -l device

Available video capture device :

index : 0, name : FaceTime HD Camera (Built-in)

Available audio capture device :

index : 0, name : Headset Microphone (Apple Audio Device)

index : 1, name : Internal Digital Microphone (Apple Audio Device)

Note : The demo version has the following limitations:

Maximum support four concurrent sessions.

Chapter 11 Multiple capture devices support

If your system have multiple audio capture device, you can use

`rtsp://[serverip]:[serverport]/[application-name]/audiodeviceN`

`rtmp://[serverip]:[serverport]/[application-name]/audiodeviceN`

`srt://[serverip]:[serverport]?streamid=[application-name]/audiodeviceN`

`http://[serverip]:[serverport]/[application-name]/audiodeviceN`

The N to specify the audio capture device index, start from 0, such as:

`rtsp://192.168.0.100/myapp/audiodevice` ; stream audio from the first audio device

`rtsp://192.168.0.100/myapp/audiodevice1` ; stream audio from the second audio device

If your system have multiple video capture device, you can use

`rtsp://[serverip]:[serverport]/[application-name]/videodeviceN`

`rtmp://[serverip]:[serverport]/[application-name]/videodeviceN`

`srt://[serverip]:[serverport]?streamid=[application-name]/videodeviceN`

`http://[serverip]:[serverport]/[application-name]/videodeviceN`

The N to specify the video capture device index, start from 0, such as:

`rtsp://192.168.0.100/myapp/videodevice` ; stream video from the first video device

`rtsp://192.168.0.100/myapp/videodevice1` ; stream video from the second video device

If your system have multiple monitors, you can use

`rtsp://[serverip]:[serverport]/[application-name]/screenliveN`

`rtmp://[serverip]:[serverport]/[application-name]/screenliveN`

`srt://[serverip]:[serverport]?streamid=[application-name]/screenliveN`

`http://[serverip]:[serverport]/[application-name]/screenliveN`

The N to specify the monitor index, start from 0, such as:

`rtsp://192.168.0.100/myapp/screenlive` ; stream living screen from the first monitor

`rtsp://192.168.0.100/myapp/screenlive1` ; stream living screen the second monitor

The audio index or video index represents which device can run ***mediaserver -l device*** to view.

videodevice or audiodevice can also specify the device name, such as:
rtsp://[serverip]:[serverport]/[application-name]/videodevice=testvideo

Run the *mediaserver -l device* command to get the device name.

Note that there can be no spaces in the device name, if the device name contains spaces, you need to use %20 instead of spaces.

If the device name is “FaceTime HD Camera (Built-in)”, the rtsp stream address is:

rtsp://[serverip]:[serverport]/[application-name]/videodevice=FaceTime%20HD
%20Camera%20(Built-in)

Chapter 12 Capture application window

The media server supports capturing application windows, you can use the following command to list valid application windows:

```
mediaserver -l window
```

Below is a sample output of the command

Available window name :

```
C:\Windows\system32\cmd.exe - MediaServer.exe -l window
user manual.doc - WPS Office
Rtmp-server Project - Source Insight - [Main.cpp]
RtmpServer
```

You can use the following url to capture the specified application window:

```
rtsp://[serverip]:[serverport]/[application-name]/window=[window title]
```

Note : window title case insensitive

Such as :

```
rtsp://[serverip]:[serverport]/[application-name]/window=mediaserver
```

Note that there can be no spaces in the window title, if the window title contains spaces, you need to use %20 instead of spaces. Such as:

```
rtsp://[serverip]:[serverport]/[application-name]/window=user%20manual.doc%
20-%20WPS%20office
```

Chapter 13 Support URL parameters

Media server supports URL parameters, with the following format:

Taking playing the test.mp4 file as an example:

rtsp://[serverip]:[serverport]/[application-name]/test.mp4?param1=value1¶m2=value2

Param1 and param2 represent URL parameters, while value1 and value2 represent the values of param1 and param2.

Note : The parameter value specified through the URL has a higher priority than the parameters configured in the configuration file.

The media server supports the following parameters:

Note : Unless otherwise specified, the parameters are valid for all streams.

srtp : Whether to enable SRTP, valid for rtsp stream.

Possible values

0 - disable srtp

1 - enable srtp

t : Specify the transmission method, valid for rtsp stream.

Possible values:

unicast

multicast

p : Specify transmission protocol, valid for rtsp stream.

Possible values:

udp : rtp over udp

tcp : rtp over tcp

ve : Specify video encoding

Possible values:

H264

H265

JPEG

MP4

fps : Specify video frame rate

w : Specify video width

h : Specify video height

vb : Specify video bitrate

ae : Specify audio encoding

Possible values:

PCMU : g711 ulaw

PCMA: g711 alaw

G726

G722

OPUS

AAC

sr : Specify audio samplerate

ch : Specify the number of audio channels

ab : Specify audio bitrate

Example:

Using unicast RTP over UDP mode, output video encoding as H264, resolution as 1920 * 1080, frame rate as 30, bit rate as 4000K, audio as AAC, sampling rate as 44100, stereo RTSP stream:

**rtsp://[serverip]:[serverport]/[application-name]/test.mp4?t=unicast&p=udp
&ve=h264&w=1920&h=1080&fps=30&vb=4000&ae=aac&sr=44100&ch=2**

Chapter 14 Support SRTP

Happytiem media server supports Secure Real-time Transport Protocol (secure RTP or SRTP) for rtsp publishing and playback. The encryption keys can be passed in the SDP data. The open-source tools FFmpeg and FFplay can be used for SRTP testing.

14.1 SRTP for rtsp publishing

Happytime RTSP Pusher supports SRTP publishing. Modify the RTSP Pusher configuration file to set <SRTP> under the <Pusher> tag to 1, and RTSP Pusher will use SRTP for data publishing:

RTSP Pusher uses SRTP to publish configuration examples:

```
<pusher>
  <src>test.mp4</src>
  <transfer>
    <mode>RTSP</mode>
    <rtspurl>rtsp://192.168.3.36/myapp/live</rtspurl>
    <user>admin</user>
    <pass>admin</pass>
  </transfer>
  <video>
    .....
  </video>
  <audio>
    .....
  </audio>
  <metadata>0</metadata>
  <srtp>1</srtp>
</pusher>
```

You will see SDP information in the ANNOUNCE request message of rsp pusher. It should look something like this:

```
v=0
o=- 0 0 IN IP4 192.168.3.36
c=IN IP4 192.168.3.36
s=session
t=0 0
```

```
a=control:*
m=video 0 RTP/SAVP 96
a=rtpmap:96 H264/90000
a=fmtp:96
packetization-mode=1;profile-level-id=42801F;sprop-parameter-sets=ZOKAH5ZSAKALdJQEBAUAAAMAAQ
AAAwAyhA==,aMuNSA==
a=control:realvideo
a=crypto:1 AES_CM_128_HMAC_SHA1_80 inline:KSO+hOFslq5SkEnx8bvp670m2zyHDD6ZJF4NHAA3
m=audio 0 RTP/SAVP 97
a=rtpmap:97 MPEG4-GENERIC/44100/2
a=fmtp:97
streamtype=5;profile-level-id=1;mode=AAC-hbr;sizelength=13;indexlength=3;indexdeltalength=3;
config=121056E500
a=control:realaudio
a=crypto:1 AES_CM_128_HMAC_SHA1_80 inline:R96zEk3IQ7uLph8DWn0JOCUfXdtL/Jb1RTsTDYkK
```

14.2 SRTP for rtsp playback

The media server supports the URL parameter SRTP to specify whether rtsp SRTP playback is enabled or not.

Example:

Enable SRTP to play test.mp4 files:

```
rtsp://[serverip]:[serverport]/[application-name]/test.mp4?srtsp=1
```

Use FFplay or Happytime rtsp client to test playback.