In a browser via HLS

- Overview
    - Supported platforms and browsers
    - Supported codecs
    - Operation flowchart
- Quick manual on testing
    - Publishing of a video stream on the server and playing it via HLS in a browser
- Call flow
- Streams which can be played as HLS
- HLS segments automatic cut for stream published
- HLS playback authentication using REST hook
    - Custom backend application usage for HLS playback authentication
    - Unauthorized access to HLS segments prevention
- Adding cross-domain access control HTTP headers for HLS playback
    - Mask support in ACAO header
- Using nginx as reverse proxy for HLS playback
- Returning of static HTML pages on HLS port
- Preloader for HLS stream playback
    - Disabling HLS preloader
    - Custom preloader configuration
- HLS subscription management using REST API
    - REST queries and response sates
    - Parameters
    - Issues
    - LL HLS stream issues displaying
    - HLS statistics displaying
        - HLS common data
        - Audio and video profiles data
        - HLS subscribers count
        - HLS subscribers and connections displaying issues
- HLS ABR support
    - Legacy HLS ABR implementation in builds5.2.484-5.2.582
    - Actual HLS ABR implementation in build5.2.585and newer
        - Transcoder node settings
        - HLS Edgenode settings
        - Usage
        - Transcoding to higher resolutions prevention
        - Known limits
    - HLS ABR on a single node
        - Transcoding to higher resolutions prevention
    - Qualities order in HLS ABR manifest
    - Force transcoding of a maximum ABR quality only if there are B-frames in a source stream
- Maximum playlist size
- HLS segments storage
    - Using disk
    - Using memory
- Debug logs for HLS session
- Low Latency HLS support
    - Recommended settings for LL HLS playback
    - Using HTTP/2 and HTTP/1
    - LL HLS segments folder
    - Maximum LL HLS playlist size
    - LL HLS preloader
        - Custom LL HLS preloader setup
- m4s container support
- Using a common network stack for HLS and Low Latency HLS
- Manifest URL setup
- Known issues

Overview

HTTP Live Streaming (HLS) is a technology to play streaming video via the HTTP protocol developed by Apple. An HLS video stream is encoded to H.264 and AAC and is played on any compatible device, browser or player.

Web Call Server converts to HLS video received fromother supported sourcesof broadcasting such as web cameras and professional video capturing device, SIP calls and so on.

Supported platforms and browsers

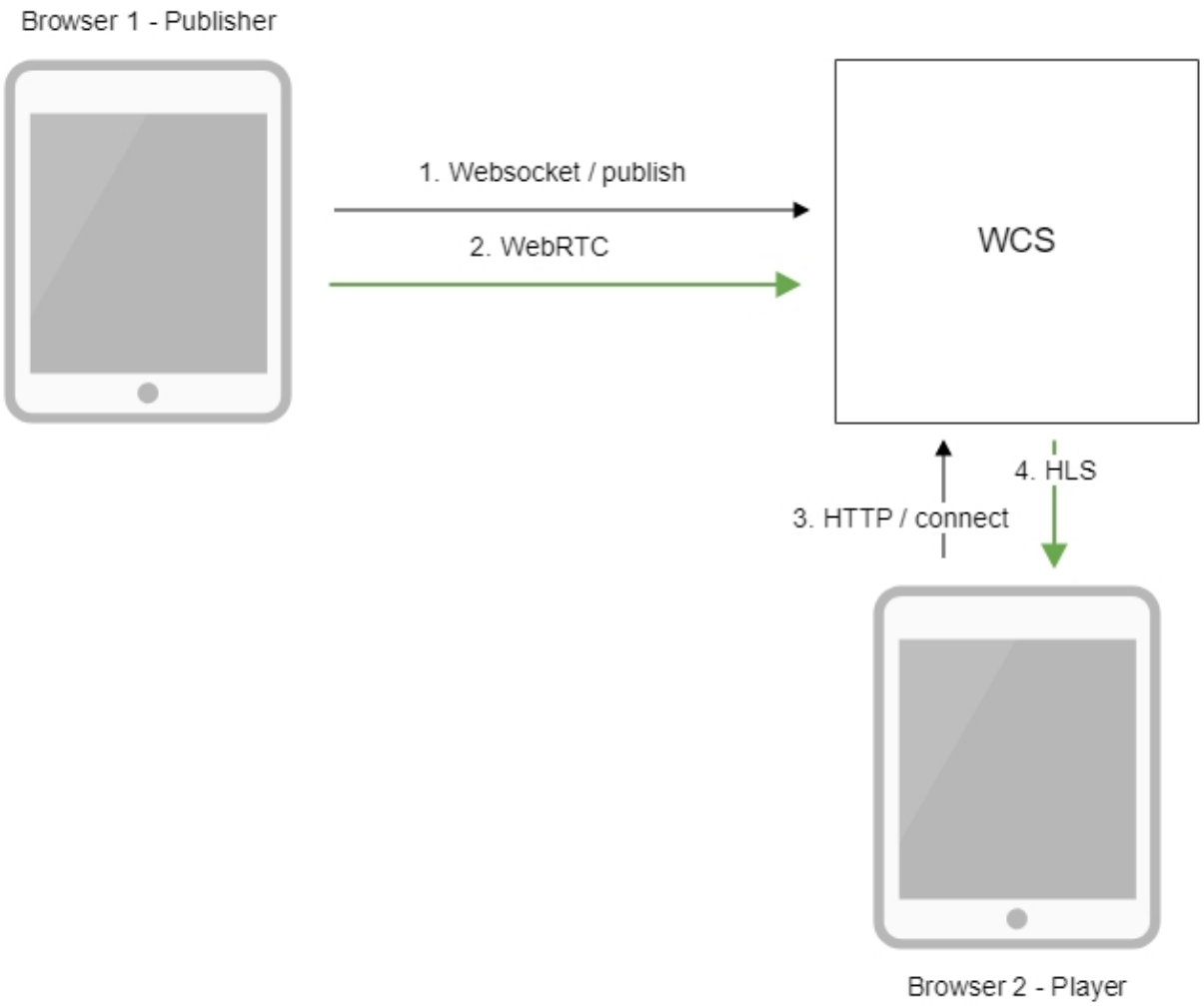
	Chrome	Firefox	Safari 11	Edge
Windows	+	+		+
Mac OS	+	+	+	

Android	+	+		
iOS	+	+	+	

Supported codecs

- Video: H.264
- Audio: AAC

Operation flowchart



1. The browser connects to the server via the Websocket protocol and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The second browser establishes a connection via HTTP.
4. The second browser receives the HLS stream and plays this stream on the page.

Quick manual on testing

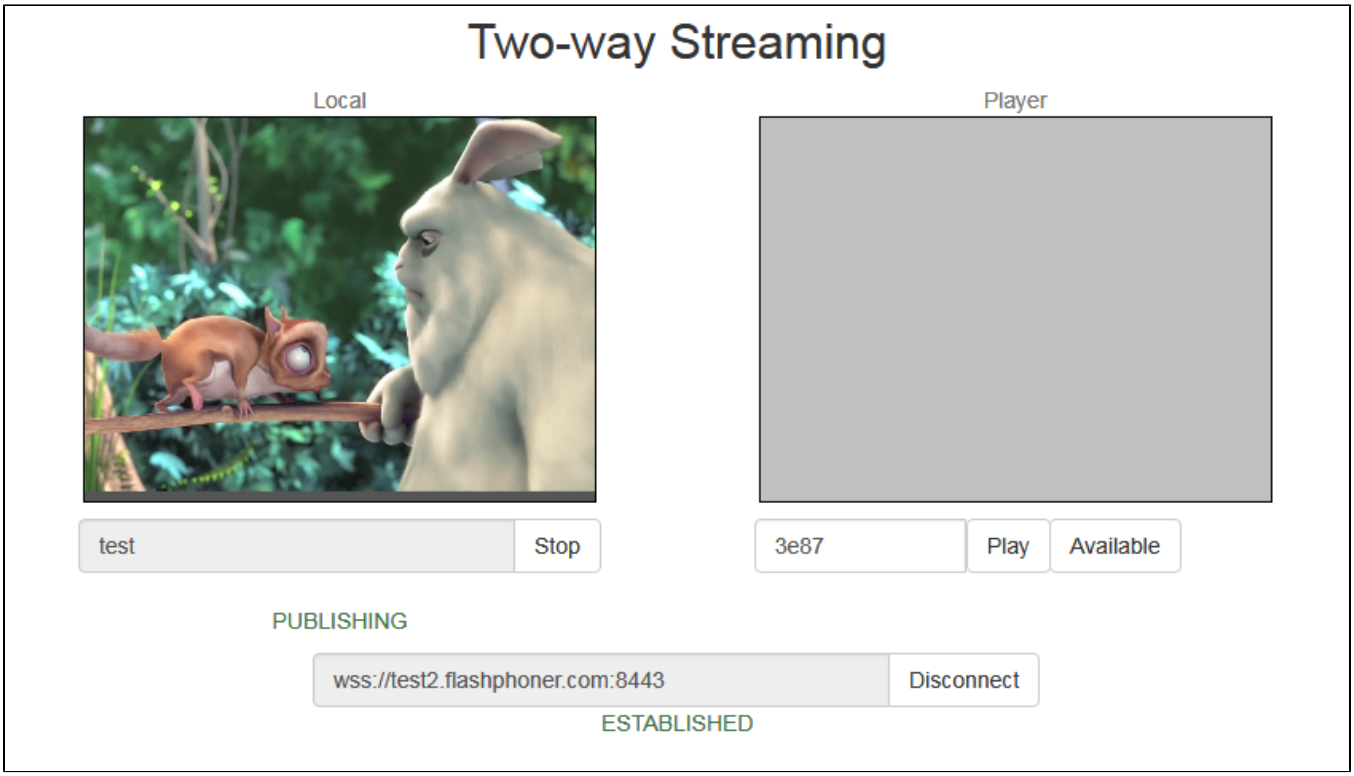
Publishing of a video stream on the server and playing it via HLS in a browser

1. For the test we use:

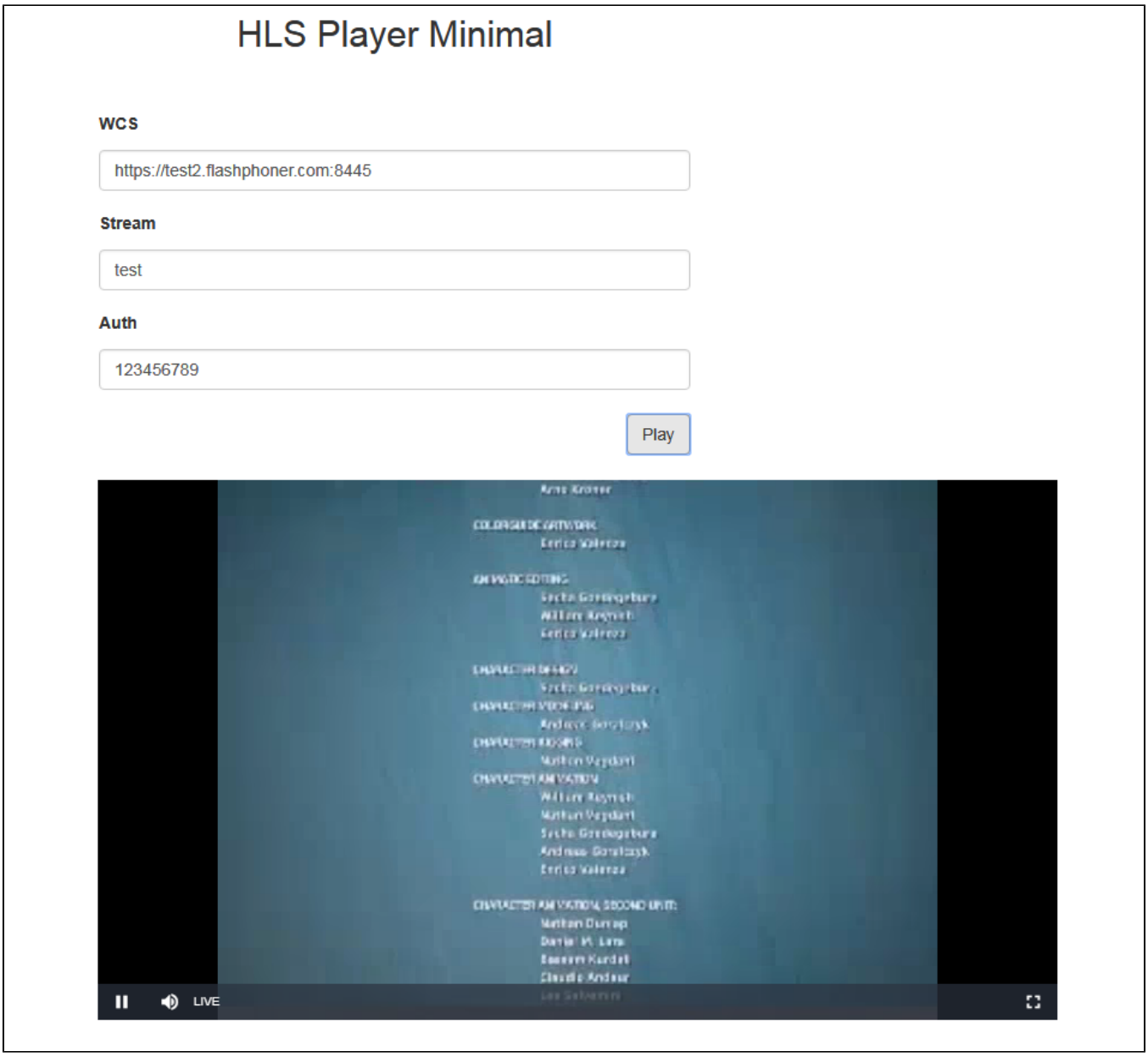
- WCS server;
- theTwo Way Streamingweb application to publish the stream

- theHLS Player Minimalweb application to publish the stream

2. Open the Two Way Streaming web application. Click Connect, then Publish. Copy the identifier of the stream:



3. open the HLS Player Minimal web application. In the Stream field paste the identifier of the stream and click Play. The stream starts playing:

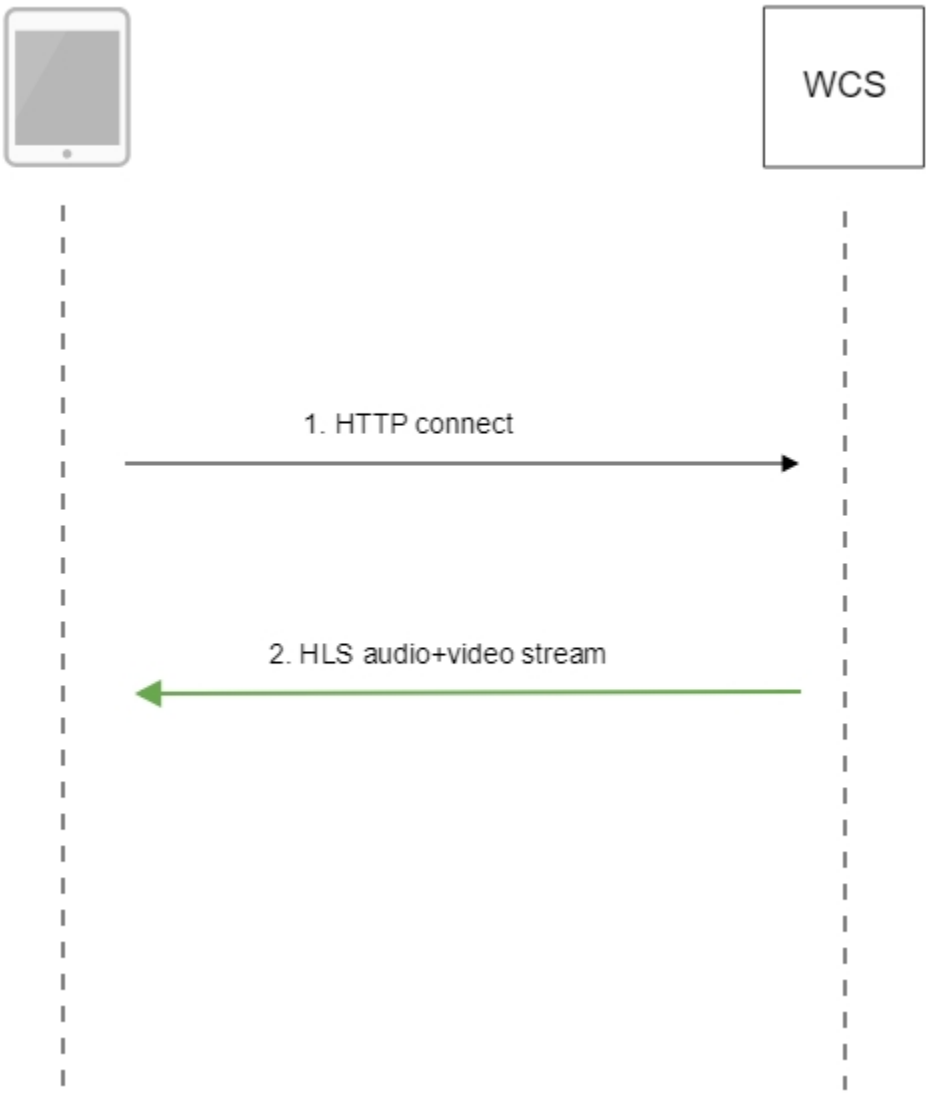


Call flow

Below is the call flow when using the HLS Player Minimal example to play a stream via HLS

hls-player.html

hls-player.js



1. Querying the server and playing.

code

```
var player = videojs('remoteVideo');
```

Configuring the HLS URL.

code

```
player.src({
    src: $("#urlServer").val() + "/" + streamName + "/" + streamName + ".m3u8",
    type: "application/vnd.apple.mpegurl"
});
```

Starting the playback.

code

```
        player.play();
```

2. Receiving the HLS stream from the server

Streams which can be played as HLS

Every stream published to WCS server with certain name, can be played as HLS

```
http://wcs:8082/streamName/streamName.m3u8
```

A stream name can be set whenpublishing stream from browseror fromRTMP кодировщика, or when capturingRTSP, RTMPorVOD stream usingREST API

Since build5.2.771, stream URI can be set for RTSP

```
http://wcs:8082/rtsp%3A%2F%2Frtspserver%2Flive.sdp/rtsp%3A%2F%2Frtspserver%2Flive.sdp.m3u8
```

RTMP source

```
http://wcs:8082/rtmp%3A%2F%2Frtmpserver%3A1935%2Flive%2Fstream/rtmp%3A%2F%2Frtmpserver%3A1935%2Flive%2Fstream.
m3u8
```

or for VOD live translation from file

```
http://wcs:8082/vod-live%3A%2F%2Ffile.mp4/vod-live%3A%2F%2Ffile.mp4.m3u8
```

In this case a stream will be captured by source URI, and after publishing to server the stream will be played as HLS. Note that URI should be encoded, all the characters except latin letters and digits must be replaced with character code.

Since build 5.2.1679 a stream may be played by source URI as HLS ABR

```
http://wcs:8082/vod-live%3A%2F%2Ffile.mp4-HLS-ABR-STREAM/vod-live%3A%2F%2Ffile.mp4-HLS-ABR-STREAM.m3u8
```

When HLS subscriber connects toCDN Edge, if a stream with certain name or URI is already published to some Origin server, the stream from CDN will be played as HLS for this subscriber. If there is no stream with such name or URI in CDN, Edge will try to capture stream locally to play as HLS.

HLS segments automatic cut for stream published

Any of streams published via WebRTC, RTMP, MPEG-TS or captured from RTSP or RTMP source using REST API may be cut automatically to HLS segments. This feature may be enabled with the following parameter

```
hls_auto_start=true
```

Since build 5.2.1895 it is possible to start HLS ABR stream automatically if HLS ABR on a single node is used. This feature may be enabled with the following parameter

```
hls_abr_auto_start=true
```

HLS playback authentication using REST hook

A client authentication for HLS playback can be setup if necessary. To do this, the following parameter should be set inflashphoner.propertiesfile

```
hls_auth_enabled=true
```

At client side, the parameter must be added to HLS URL to pass to WCS server a token obtained for example from backend server. The name of the parameter is defined with the following setting

```
client_acl_property_name=aclAuth
```

In this case, stream URL should be formed as follows

```
        var src = $("#urlServer").val() + "/" + streamName + "/" + streamName + ".m3u8";
        var token = $("#token").val();
        if (token.length > 0) {
            src += "?aclAuth=" +token;
        }
```

REST hook/playHLS must be implemented on backend server in defaultApp application. WCS server will send query to backend with token received from client

```
URL:http://localhost:8081/apps/EchoApp/playHLS
OBJECT:
{
  "nodeId" : "NTk1tLorQ00llGbPJuFexrKceubGCR0k@192.168.1.5",
  "appKey" : "defaultApp",
  "sessionId" : "/192.168.1.100:59473/192.168.1.5:8445",
  "mediaSessionId" : "60709c5b-6950-40c3-8a3d-37ea0827ae32-
727473703a2f2f73747238312e63726561636173742e636f6d2f6772616e646c696c6c6574762f6c6f77-HLS",
  "name" : "test",
  "mediaProvider" : "HLS",
  "custom" : {
    "token" : "12345789"
  }
}
```

Backend server should return 200 OK if token is checked successfully, and 403 Forbidden if token is wrong. In its turn, client will receive either HLS stream or 401 Unauthorized.

The parameter

```
hls_auth_token_cache=10
```

defines token caching interval in seconds (10 seconds by default). /playHLS queries with certain token will not be sent to backend until the token is in cache i.e. either there is stream subscriber with this token or caching interval is not expired. If caching interval parameter is set to 0

```
hls_auth_token_cache=0
```

/playHLS queries are sent to backend on every HTTP GET request from client.

HLS authentication setting can be changed without server restart. In this case `hls_auth_enabled`affects existing subscribers and `hls_auth_token_cache`affects new subscribers only.

Custom backend application usage for HLS playback authentication

Since build5.2.1008it is possible to set backend application key in HLS URL, for example

```
https://wcs:8445/streamName/streamName.m3u8?appKey=customAppKey&aclAuth=1254789
```

In this case REST hook /playHLS will be sent to backend application with defined key (customAppKey in the example above).

Unauthorized access to HLS segments prevention

To decrease server load, authentication token and stream availability in CDN are checked on HLS playlist request. The following parameter is added since build 5.2.436to protect separate HLS segments

```
hls_segment_name_suffix_randomizer_enabled=true
```

In this case, randomly generated suffix is added to every segment file name, for example

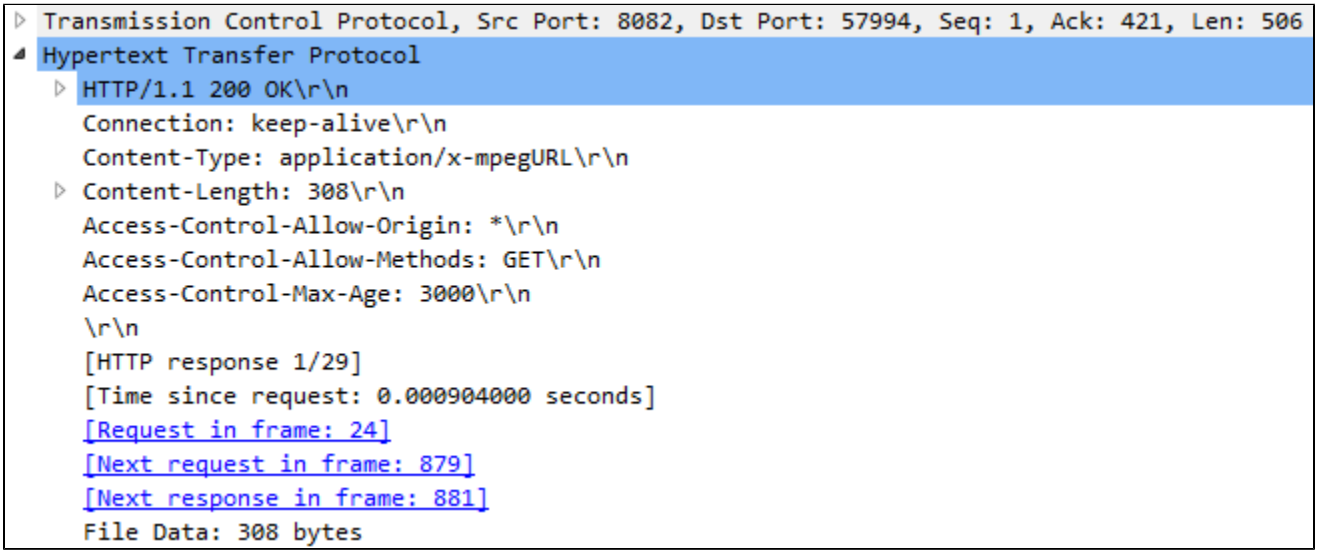
```
test16d2da4658f4374953a120f3c95bc715ea.ts
```

So, brute force iteration over segments is escaped.

Note that suffix is not added topreloader segments.

Adding cross-domain access control HTTP headers for HLS playback

By default, the followin access control headers are added to 200 OK response to HTTP GET request:

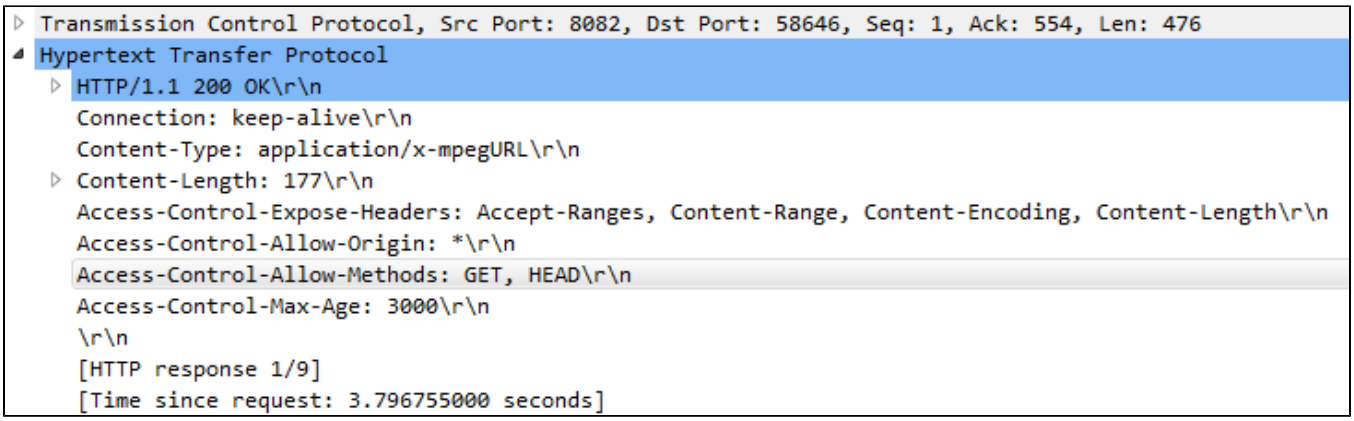



```
Access-Control-Allow-Origin: *
Access-Control-Allow-Methods: GET
Access-Control-Max-Age: 3000
```

If necessary, for example, if HLS content and HLS player page are in different domains, custom access control headers can be added using the following parameter inflashphoner.propertiesfile:

```
hls_access_control_headers=Access-Control-Allow-Origin: *;Access-Control-Allow-Methods: GET, HEAD;Access-
Control-Max-Age: 3000;Access-Control-Expose-Headers: Accept-Ranges, Content-Range, Content-Encoding, Content-
Length
```

In this case, the headers listed in the parameter will be added to 200 OK response:



Mask support in ACAO header

Sometimes, for example, when load balancer AWS LB is used, it is necessary to set request origin address including port in ACAO header sent in response to GET request, for example

```
Access-Control-Allow-Origin: http://test.flashphoner.com:8081
```

However, origin address is not always known while configuring the server. Therefore, since build5.2.755mask support in ACAO header can be enabled

```
hls_acao_header_domain_mask=true
```

This feature is enabled by default. In this case, if '*' character is set in the following parameter

```
hls_access_control_headers=Access-Control-Allow-Origin: *
```

server returns ACAO header with full request origin address in response to GET request

```
Access-Control-Allow-Origin: http://lb.yourdomain.com:8081
```

This can be disabled if necessary

```
hls_acao_header_domain_mask=false
```

Using nginx as reverse proxy for HLS playback

In some cases nginx web server can be used as reverse proxy for HLS playback from WCS server. Usually, it may require if HTTP headers adding does not help to workaround cross domain request restrictions in some browsers.

For example, if browser requires HLS player page and HLS stream to be in the same domain your.domain and on the same port 443 (HTTPS), nginx should be set up as follows:

```
    # HTTP requests are redirected from port  80 to 443
    server {
        listen 80;
        server_name docs.flashphoner.com;
            return 301 https://$server_name$request_uri;
    }

    # Server listens HTTPS port 443
    server {
                listen 443 ssl;
                ssl_certificate /etc/letsencrypt/live/your.domain/fullchain.pem;
                ssl_certificate_key /etc/letsencrypt/live/your.domain/privkey.pem;
                server_name your.domain;
                server_tokens off;
                client_max_body_size 500m;
                proxy_read_timeout 10m;

        root         /usr/share/nginx/html;


        location / {
        }

        error_page 404 /404.html;
            location = /40x.html {
        }

        error_page 500 502 503 504 /50x.html;
            location = /50x.html {
        }

        # Example web applications will be available by URL https://your.domain/client2
        location /client2/ {
            alias /usr/local/FlashphonerWebCallServer/client2/;
        }

        # HLS playlists and segments are proxying to your.domain on port 443. for example https://your.domain
/test.m3u8
                location ~* ^.+.(m3u8|ts)$ {
                    proxy_pass https://localhost:8445;
                    proxy_http_version  1.1;
                    proxy_set_header    Host $server_name:$server_port;
                    proxy_set_header    X-Forwarded-Host $http_host;
                    proxy_set_header    X-Forwarded-Proto $scheme;
                    proxy_set_header    X-Forwarded-For $remote_addr;
                    proxy_set_header    Upgrade $http_upgrade;
                    proxy_set_header    Connection "upgrade";
                }

    }
```

It also may be useful to cache HLS stream. In this case nginx should be additionally set up as follow:

1. In `http` section of /etc/nginx.conf settings file proxy cache parameters are set

```
    proxy_cache_path /var/cache/nginx/proxy levels=1:2 keys_zone=proxy_cache:1024m max_size=2048m inactive=10d;
    proxy_cache_min_uses 1;
    proxy_ignore_headers X-Accel-Expires;
    proxy_ignore_headers Expires;
    proxy_ignore_headers Cache-Control;
```

2. In `server` section of site settings file caching of HLS segments is set, playlist should not be cached:

```
        location ~* ^.+.(ts)$ {
            proxy_pass https://localhost:8445;
            proxy_http_version  1.1;
            proxy_set_header    Host $server_name:$server_port;
            proxy_set_header    X-Forwarded-Host $http_host;
            proxy_set_header    X-Forwarded-Proto $scheme;
            proxy_set_header    X-Forwarded-For $remote_addr;
            proxy_set_header    Upgrade $http_upgrade;
            proxy_set_header    Connection "upgrade";
    proxy_cache proxy_cache;
    proxy_cache_key $host$uri$is_args$args;
    proxy_cache_valid 200 2m;
       }

        location ~* ^.+.(m3u8)$ {
            proxy_pass https://localhost:8445;
            proxy_http_version  1.1;
            proxy_set_header    Host $server_name:$server_port;
            proxy_set_header    X-Forwarded-Host $http_host;
            proxy_set_header    X-Forwarded-Proto $scheme;
            proxy_set_header    X-Forwarded-For $remote_addr;
            proxy_set_header    Upgrade $http_upgrade;
            proxy_set_header    Connection "upgrade";
    proxy_cache off;
    expires -1;
       }
```

Returning of static HTML pages on HLS port

Another way to workaround cross domain requests restrictions in browser is to return a static content, player page for example, on the same port that returns HLS content. To enable this feature, the following parameter should be setinflashphoner.propertiesfile

```
hls_static_enabled=true
```

The player page should be in directory defined by the following parameter

```
hls_static_dir=client2/examples/demo/streaming/hls_static
```

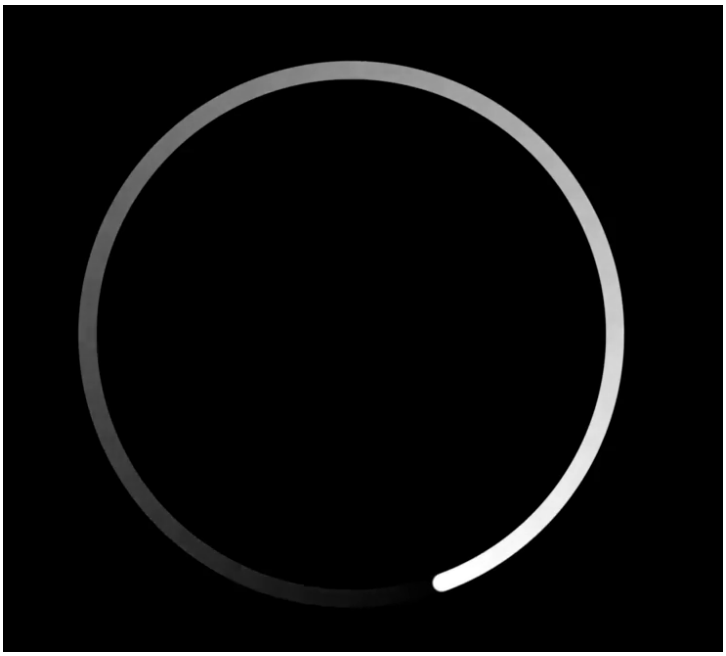
In this case (by default) the path to the player page files is set relative to WCS installation directory. A full path may also be set, for example

```
hls_static_dir=/var/www/html/hls_static
```

If static content returning is enabled, browser will display the HLS player page by URL https://host:8445/hls-player.html. If this feature is disabled, server will return 404 Not found error by such URL.

Preloader for HLS stream playback

When first HLS subscribes connects to a stream on WCS server, especilally to CDN stream, it may take a time to split a stream to HLS segmets and to form a playlist. As a result, Safari browser on iOS devices may not be able to play HLS stream on the first attempt. To successfully play HLS stream in this case, the HLS preloader was added since build5.2.371. The default preloader looks like this:



Since build5.2.408preloaders are divided by stream aspect ratio: 16:9, 4:3, 2:1

Default preloader segments are placed to`/usr/local/FlashphonerWebCallserver/hls/.preloader` folder when server is started

```
tree /usr/local/FlashphonerWebCallServer/hls/.preloader
/usr/local/FlashphonerWebCallServer/hls/.preloader
 16x9
    index0.ts
    index10.ts
    index11.ts
    index12.ts
    index13.ts
    index14.ts
    index15.ts
    index16.ts
    index17.ts
    index18.ts
    index19.ts
    index1.ts
    index2.ts
    index3.ts
    index4.ts
    index5.ts
    index6.ts
    index7.ts
    index8.ts
    index9.ts
 2x1
    index0.ts
    index10.ts
    index11.ts
    index12.ts
    index13.ts
    index14.ts
    index15.ts
    index16.ts
    index17.ts
    index18.ts
    index19.ts
    index1.ts
    index2.ts
    index3.ts
    index4.ts
    index5.ts
    index6.ts
    index7.ts
    index8.ts
    index9.ts
 4x3
     index0.ts
     index10.ts
     index11.ts
     index12.ts
     index13.ts
     index14.ts
     index15.ts
     index16.ts
     index17.ts
     index18.ts
     index19.ts
     index1.ts
     index2.ts
     index3.ts
     index4.ts
     index5.ts
     index6.ts
     index7.ts
     index8.ts
     index9.ts
```

The minimal preloader segment duration is 2 seconds by default, and can be set in milliseconds with the following parameter

```
hls_preloader_time_min=2000
```

Disabling HLS preloader

HLS preloader can be disabled if necessary, this feature is available since build5.2.396. To disable HLS preloader, the following parameter is used

```
hls_preloader_enabled=false
```

Custom preloader configuration

To replace the default preloader to thecustom one, do the following:

1. Choose video clip (logo for example) in three aspect ratios:16:9, 4:3, 2:1

2. Encode video to H264, add audio track to the clip, set GOP and remove B-frames using ffmpeg

```
ffmpeg -i clip16x9.mp4 -f lavfi -i anullsrc=channel_layout=mono:sample_rate=44100 -c:v h264 -g 30 -bf 0 -
shortest 16x9/preloader16x9.mp4
ffmpeg -i clip4x3.mp4 -f lavfi -i anullsrc=channel_layout=mono:sample_rate=44100 -c:v h264 -g 30 -bf 0 -
shortest 4x3/preloader4x3.mp4
ffmpeg -i clip2x1.mp4 -f lavfi -i anullsrc=channel_layout=mono:sample_rate=44100 -c:v h264 -g 30 -bf 0 -
shortest 2x1/preloader2x1.mp4
```

3. Download and install HLS toolsfrom Apple site

4. Prepare custom preloader HLS segments with desired duration, for example 2 seconds

```
cd 16x9
mediafilesegmenter -t 2 -B index -start-segments-with-iframe preloader16x9.mp4
tar -cvzf preloader.tar.gz index*.ts
```

This step should be repeated for all aspect ratios.

5. Make a folder for custom preloader

```
mkdir /opt/custom_preloader
mkdir /opt/custom_preloader/16x9
mkdir /opt/custom_preloader/4x3
mkdir /opt/custom_preloader/2x1
```

6. Unpack preloader segments from archive prepared on step 4

```
cd /opt/custom_preloader/16x9
tar -xvzf ~/preloader16x9.tar.gz
```

This step should also be repeated for all aspect ratios.

7. Set custom preloader folder and duration in server settings

```
hls_preloader_time_min=2000
hls_preloader_dir=/opt/custom_preloader
```

HLS subscription management using REST API

REST query should be HTTP/HTTPS POST request as follows:

- HTTP:http://test.flashphoner.com:8081/rest-api/hls/startup
- HTTPS:https://test.flashphoner.com:8444/rest-api/hls/startup

Where:

- test.flashphoner.com - WCS server address
- 8081 - WCS server standard REST / HTTP port
- 8444 - standard HTTPS port
- rest-api - mandatory URL part
- /hls/startup - REST method used

REST queries and response sates

REST query	Body example	Response example	Response state	Desctiption

/hls /startup	<pre>{
	  "name":"test"
	}</pre>
			200 - OK
			
			404 - Stream not found
			
			500 - Internal error
				Start HLS agent for the stream

/hls /find_all	<pre>{
	  "offset":0,
	  "size":10
	}</pre>
		<pre>[{
		    "id": "test",
		    "streamName": "test",
		    "status": "ACTIVE",
		    "waitingSize": 0,
		    "profiles": [
		      "a_test",
		      "v_test"
		    ],
		    "subscribers": 1,
		    "playlist": "...",
		    "createdDate": 1697605114475,
		    "logs": []
		}]</pre>
			200 – OK
			
			404 – Not found
				Find all streams having HLS agents

/hls /terminate	<pre>{
	  "name":"test"
	}</pre>
			200 – OK
			
			404 – Not found
				Stop or restart HLS agent for the stream

/hls
/profiles
	<pre>{
	  "hlsId":"test",
	  "profileName":"v_test" }</pre>
		<pre>{
		  "name": "v_test",
		  "stream": {
		    "appKey": "defaultApp",
		    "sessionId": "test-HLS",
		    "mediaSessionId": "81b8b278-612e-4b72-9153-454be9df0a34-test-HLS",
		    "name": "test",
		    "published": false,
		    "hasVideo": false,
		    "hasAudio": true,
		    "status": "PLAYING",
		    "sdp": "...",
		    "videoCodec": "H264",
		    "record": false,
		    "width": 1280,
		    "height": 720,
		    "bitrate": 0,
		    "minBitrate": 0,
		    "maxBitrate": 0,
		    "quality": 0,
		    "parentMediaSessionId": "f3401d2e-7e9a-4e18-a353-d323c947ac94",
		    "history": false,
		    "gop": 0,
		    "fps": 0,
		    "audioBitrate": 0,
		    "codecImpl": "",
		    "transport": "UDP",
		    "cvoExtension": true,
		    "createDate": 1697605114875,
		    "mediaType": "play",
		    "audioState": {
		      "muted": false
		    },
		    "videoState": {
		      "muted": false
		    },
		    "mediaProvider": "HLS"
		  },
		  "keyFrameReceived": true,
		  "videoProfile": {
		    "type": "video",
		    "width": 1280,
		    "height": 720,
		    "fps": 29,
		    "bitrate": 1720,
		    "codec": "",
		    "quality": 0,
		    "audioGroupId": "audio"
		  },
		  "metrics": {
		    "minFPS": 29.962547,
		    "avgFPS": 30.000088,
		    "maxFPS": 30.04292,
		    "countGaps": 0,
		    "resolutionChanges": 0,
		    "queueSize": 10,
		    "startPts": 375400,
		    "currentPts": 375400
		  },
		  "subscribers": 1
		}</pre>
			200 – OK
			
			400 – Bad request
			
			404 – Not found
				Get HLS profile statistics

/hls/subscribers	<pre>{
	  "hlsId":"test"
	}</pre>
		<pre>[
		  {
		  "id": "192.168.0.83-59000-Mozilla/5.0 (X11; Linux x86_64) Chrome/118.0.0.0",
		  "ip": "192.168.0.83",
		  "port": 59000,
		  "userAgent": "Mozilla/5.0 (X11; Linux x86_64) Chrome/118.0.0.0",
		  "active": true,
		  "metrics": {
		  "profileTime": {
		  "test": 71,
		  "v_test": 541353
		  },
		  "requestsNumber": 5930,
		  "requestsStatuses": {
		  "200 OK": 5930
		  },
		  "profileSwitches": 1,
		  "maxResponseTime": 29,
		  "minResponseTime": 0,
		  "avgResponseTime": 0.4436762225969646
		  }
		  }
		]</pre>
			200 – OK
			
			400 – Bad request
			
			404 – Not found
				Get HLS subscribers statistics

Parameters

Parameter name	Description	Example
name	Stream published name	test
logs	Messages about stream issues	[]

Issues

1. If HLS agent for the stream is started by REST query /hls/startup, and there are no active HLS subscribers, agent will stop after the following timeout in seconds

```
hls_manager_provider_timeout=300
```

By default, the timeout is 5 minutes. Also it concerns HLS agents which are started automatically for streams published using the following parameter

```
hls_auto_start=true
```

2. If HLS agent for the stream is stopped by REST query /hls/terminate, and there are active HLS subscribers, this agent will be restarted. In this case, active HLS subscribers must reconnect to the stream.

LL HLS stream issues displaying

Since build 5.2.1709 LL HLS stream issues are displaying in response on `/hls/find_all` REST API query:

```
{
   "test": {
    "handler": "com.flashphoner.server.client.handler.wcs4.WCS4Handler@74dbf27b",
    "state": "ACTIVE",
    "writer": "HLS-test",
    "streamStatus": "PLAYING",
    "writerStarted": "true",
    "logs": [
      "2023-07-18T10:22:52.457 WARNING: Playback speed changed to 0.779, segment 49, media type: video",
      "2023-07-18T10:22:56.614 WARNING: Gap{from=112000, to=114000, duration=2000}, media type: video",
      "2023-07-18T10:22:56.615 WARNING: Fps changed from 30 to 27, segment 50 , media type: video",
      "2023-07-18T10:22:56.624 WARNING: Segment 51.1 have no data, pts 112400, duration 400, media type: video",
      ...
    ]
  }
}
```

By default, up to 50 last issues are displayed for every stream. This value may be changed with the following parameter

```
hls_metrics_log_size=50
```

The following issues are detected:

- `Fps changed from x to y` - stream FPS leap over 10 %
- `Segment x does not start with keyframe` - stream keyframe interval is more than one segment duration
- `Playback speed changed to x` - stream playback speed has changed
- `Segment interval is too big` - an interval in milliseconds between a subsequent segments is too big
- Video resolution changed from x to y - stream resolution has changed
- `Gap{from=x, to=y, duration=z}` - stream gap detected, `EXT-X-GAP` tag is added to the playlist

Any of those issues means a source stream publishing quality degradation and may lead to freezes, out of audio and video sync and even HLS playback stopping in some browsers. In this case, a possible packets loss or bandwidth issues should be resolved, or publishing technology shuld be changed from WebRTC to a more noise-resistant, RTMP or SRT for example.

HLS statistics displaying

Since build 5.2.1777 it is possible to get HLS statistics via REST API

HLS common data

In the response to the `/hls/find_all` query HLS agents list is returned containing a common HLS data:

```
[{
    "id": "test",
    "streamName": "test",
    "status": "ACTIVE",
    "waitingSize": 0,
    "profiles": [
      "a_test",
      "v_test"
    ],
    "subscribers": 1,
    "playlist": "#EXTM3U\n#EXT-X-VERSION:9\n#EXT-X-INDEPENDENT-SEGMENTS\n#EXT-X-MEDIA:TYPE=AUDIO,URI=\"a_test
/a_test.m3u8\",GROUP-ID=\"audio\",NAME=\"none\",DEFAULT=YES,AUTOSELECT=YES,CHANNELS=\"2\"\n#EXT-X-STREAM-INF:
BANDWIDTH=1761281,CODECS=\"avc1.640028,mp4a.40.2\",RESOLUTION=1280x720,FRAME-RATE=29.0,AUDIO=\"audio\"\nv_test
/v_test.m3u8\n",
    "createdDate": 1697605114475,
    "logs": []
}]
```

Where

- id - HLS stream identifier
- streamName - a source stream name which is cut to a segments
- waitingSize - HTTP requests count waiting for response
- profiles - audio and video profiles list
- subscribers - HLS subscribers count

- playlist - HLS manifest content
- createdDate - HLS agent creation date as integer
- logs - HLS stream issues log

if there are a much HLS streams on the server, the list may be limited by the following parameters

```
{
  "offset":0,
  "size":10
}
```

Where

- offset - from which item the list should be dispalyed, 0 by default
- size - a maximum list items to display, 10 by default

Audio and video profiles data

In the response to the `/hls/profiles` query an audio or video HLS profile statistics is returned:

```
{
  "name": "v_test",
  "stream": {
    "appKey": "defaultApp",
    "sessionId": "test-HLS",
    "mediaSessionId": "81b8b278-612e-4b72-9153-454be9df0a34-test-HLS",
    "name": "test",
    "published": false,
    "hasVideo": false,
    "hasAudio": true,
    "status": "PLAYING",
    "sdp": "v=0\r\no=- 1988962254 1988962254 IN IP4 0.0.0.0\r\nc=IN IP4 0.0.0.0\r\nt=0 0\r\na=sdplang:
en\r\nm=video 0 RTP/AVP 112\r\na=rtpmap:112 H264/90000\r\na=fmtp:112 packetization-mode=1; profile-level-
id=42001f\r\na=recvonly\r\n",
    "videoCodec": "H264",
    "record": false,
    "width": 1280,
    "height": 720,
    "bitrate": 0,
    "minBitrate": 0,
    "maxBitrate": 0,
    "quality": 0,
    "parentMediaSessionId": "f3401d2e-7e9a-4e18-a353-d323c947ac94",
    "history": false,
    "gop": 0,
    "fps": 0,
    "audioBitrate": 0,
    "codecImpl": "",
    "transport": "UDP",
    "cvoExtension": true,
    "createDate": 1697605114875,
    "mediaType": "play",
    "audioState": {
      "muted": false
    },
    "videoState": {
      "muted": false
    },
    "mediaProvider": "HLS"
  },
  "keyFrameReceived": true,
  "videoProfile": {
    "type": "video",
    "width": 1280,
    "height": 720,
    "fps": 29,
    "bitrate": 1720,
    "codec": "",
    "quality": 0,
    "audioGroupId": "audio"
  },
  "metrics": {
    "minFPS": 29.962547,
    "avgFPS": 30.000088,
    "maxFPS": 30.04292,
    "countGaps": 0,
    "resolutionChanges": 0,
    "queueSize": 10,
    "startPts": 375400,
    "currentPts": 375400
  },
  "subscribers": 1
}
```


Where:

- name - profile name
- stream - profile stream data in `/stream/find` like form
- keyFrameReceived - is there any key frame received

- audioProfile, videoProfile - audio or video profile settings
    - type - profile type: video or audio
    - width - profile picture width as defined
    - height - profile picture width as defined
    - fps - profile video frame rate as defined
    - bitrate - profile bitrate as defined
    - codec - profile codec as defined
    - quality - profile quality as defined
    - audioGroupId - audio profile id used in video profile
    - rate - audio profile samplerate
    - channels - audio profile channels number
    - groupId - audio profile id to bind to video profile
- metrics - current profile metrics:
    - minFPS - minimal FPS
    - avgFPS - average FPS
    - maxFPS - maximum FPS
    - countGaps - gaps count inserted to the stream
    - resolutionChanges - video resolution changes count
    - queueSize - stream frame queue size
    - startPts - start MPEG timestamp
    - currentPts - current MPEG timestamp
- subscribers - HLS subscribers to the profile count

HLS subscribers count

In the response to the `/hls/subscribers` query an HLS stream subscribers list is returned:

```
[
  {
    "id": "192.168.0.83-59000-Mozilla/5.0 (X11; Linux x86_64) Chrome/118.0.0.0",
    "ip": "192.168.0.83",
    "port": 59000,
    "userAgent": "Mozilla/5.0 (X11; Linux x86_64) Chrome/118.0.0.0",
    "active": true,
    "metrics": {
      "profileTime": {
        "test": 71,
        "v_test": 541353
      },
      "requestsNumber": 5930,
      "requestsStatuses": {
        "200 OK": 5930
      },
      "profileSwitches": 1,
      "maxResponseTime": 29,
      "minResponseTime": 0,
      "avgResponseTime": 0.4436762225969646
    }
  }
]
```

Where:

- id - subscriber identifier
- ip - subscriber IP address
- port - subscriber source port
- userAgent - `User-Agent` header sent by subscriber
- active - subscriber is active
- metrics - current subscriber metrics:
    - profileTime - the time the subscriber requested the profile shown by profile
    - requestsNumber - subscribers requests number
    - requestStatuses - response status codes count sent to the subscriber shown by response status code
    - profileSwitches - HLS ABR profile switches count for the subscriber
    - maxResponseTime - maximum response time
    - minResponseTime - minimum response time
    - avgResponseTime - average response time

HLS subscribers and connections displaying issues

In the response to the `/hls/find_all`, `/hls/profiles`, `/hls/subscribers` queries a current HLS subscribers count and information are returned as per browser tabs. But HLS network connections count displaying at WCS statistics page

```
curl -s http://localhost:8081/?action=stat&params=connections_hls
```

may differ from HLS subscribers count. In the most cases, HLS subscribers use HTTP 2 protocol to connect and download a segments, then all the browser tabs receiving HLS streams from the same WCS server will use the same network connection.

In this case connections count displayed by `connections_hls` parameter is usually equal to HLS port connections count displayed by `netstat` command:

```
sudo netstat -np | grep ESTABLISHED | grep java | grep 8445
```

Where 8445 is HTTPS HLS port of WCS server

HLS ABR support

For a streams with video track (video only or audio+video) WCS supports HLS ABR in CDN (a qualities are encoded on a dedicated Transcoder node) and on a single node.

HLS ABR does not work for audio only streams, WCS will return `404 Not found` in response to HLS ABR manifest request for a such stream!

Legacy HLS ABR implementation in builds5.2.484-5.2.582

Since build5.2.484HLS ABR playlists support was added. This feature can be enabled with the following parameter

```
hls_master_playlist_enabled=true
```

Master playlist file name can be set using the following parameter

```
hls_manifest_file=index.m3u8
```

Browser should get master playlist by URL

```
https://wcs_address:8445/streamName/index.m3u8
```

Where

- wcs_address - WCS server address
- streamName - stream name
- index.m3u8 - master playlist file name

When master playlist is requested for the stream, server checks if streams are published according to transcoding profiles listed in cdn_profiles.yml file, for example:

```
profiles:
  -720p:
    video:
      height: 720
      bitrate: 1000
      codec: h264
  -480p:
    video:
      height: 480
      bitrate: 1000
      codec: h264
  -240p:
    video:
      height: 240
      bitrate: 400
      codec: h264
```

All the streams published by profiles on server, will be added to master playlist, for example:

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
../streamName-720p/streamName-720p.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=852x480,CODECS="avc1.42e01f,mp4a.40.2"
../streamName-480p/streamName-480p.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=400000,RESOLUTION=426x240,CODECS="avc1.42e01f,mp4a.40.2"
../streamName-240p/streamName-240p.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2500000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
../streamName/streamName.m3u8
```

Then browser switches between HLS streams listed in master playlist depending on channel bandwidth.

> When browser requests master playlist for the certain stream, transcoded stream must already be published on server and must be cut to HLS segments

To provide HLS streams by profiles, the following should be done:

1. On a standalone server:

1.1. Periodically check if streams are transcoded to parameters set by profiles, and launch transcoding if necessary usingREST API

```
        curl -s -X POST -d "{\"uri\":\"transcoder://tcode_test-240p\",\"remoteStreamName\":\"test\",\"
localStreamName\":\"test-240p\",\"encoder\":{\"width\":320,\"height\":240}}" http://localhost:8081/rest-api
/transcoder/startup
        curl -s -X POST -d "{\"uri\":\"transcoder://tcode_test-480p\",\"remoteStreamName\":\"test\",\"
localStreamName\":\"test-480p\",\"encoder\":{\"width\":640,\"height\":480}}" http://localhost:8081/rest-api
/transcoder/startup
        curl -s -X POST -d "{\"uri\":\"transcoder://tcode_test-720p\",\"remoteStreamName\":\"test\",\"
localStreamName\":\"test-720p\",\"encoder\":{\"width\":1280,\"height\":720}}" http://localhost:8081/rest-api
/transcoder/startup
```

1.2. Periodically launch HLS cut for the streams, for example

```
curl -s -X POST -d "{\"name\":\"test\"}" http://localhost:8081/rest-api/hls/startup
sleep 1
curl -s -X POST -d "{\"name\":\"test-240p\"}" http://localhost:8081/rest-api/hls/startup
sleep 1
curl -s -X POST -d "{\"name\":\"test-480p\"}" http://localhost:8081/rest-api/hls/startup
sleep 1
curl -s -X POST -d "{\"name\":\"test-720p\"}" http://localhost:8081/rest-api/hls/startup
```

2. On an Edge server in CDN periodically request HLS streams by transcoding profiles, for example

```
curl -s http://localhost:8082/test/test.m3u8
sleep 1
curl -s http://localhost:8082/test-240p/test-240p.m3u8
sleep 1
curl -s http://localhost:8082/test-480p/test-480p.m3u8
sleep 1
curl -s http://localhost:8082/test-720p/test-720p.m3u8
sleep 1
```

Actual HLS ABR implementation in build5.2.585and newer

Since build5.2.585HLS ABR implementation is significantly changed. As usual, HLS ABR can be used in CDN only, but Edge server captures all the transcoded streams for ABR manifest stream variants within the same mediasession to syhchronize the stream variants. This requires to configure Tanscoder nodes and Edge nodes simultaneously, and adds some limits. Let's explore it below.

Transcoder node settings

To synchronize all the variants of the same stream, encoding should be aligned on Transcoder node

```
transcoder_align_encoders=true
```

Also, FPS filter should be enabled

```
video_filter_enable_fps=true
video_filter_fps=25
```

All stream variants key frames (GOPs) should be synchronized. For example, we will send key frame every 2 seconds for 25 fps stream

```
video_filter_fps_gop_synchronization=50
```

HLS Edgenode settings

HLS preloader and streams resizing should be disabled on HLS Edge node

```
hls_preloader_enabled=false
hls_player_width=0
hls_player_height=0
```

The transcoding profiles should be set as follows in cdn_profiles.yml file

```
profiles:
 -240p:
  audio:
    codec : mpeg4-generic
    rate : 48000
  video:
    height : 240
    bitrate : 300
    gop : 50
    codec : h264
 -480p:
  video:
    height : 480
    bitrate : 600
    gop : 50
    codec : h264
 -720p:
  video:
    height : 720
    bitrate : 1000
    gop : 50
    codec : h264
```

Note that audio parameters can be set for the first profile only because those parameters should be identical for all the profiles, and will be applyed according to the first profile.

Then HLS ABR should be enabled

```
hls_abr_enabled=true
```

Usage

Client should request ABR manifest giving a stream name with a special suffix

```
https://server:8445/test_0-HLS-ABR-STREAM/test_0-HLS-ABR-STREAM.m3u8
```

The suffix can be set with the following parameter

```
hls_abr_stream_name_suffix=-HLS-ABR-STREAM
```

The playlist contains links to stream variants playlists, a client can switch between then

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=614400,RESOLUTION=852x480,CODECS="avc1.42e01f,mp4a.40.2"
-480p/-480p.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1024000,RESOLUTION=1278x720,CODECS="avc1.42e01f,mp4a.40.2"
-720p/-720p.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=307200,RESOLUTION=426x240,CODECS="avc1.42e01f,mp4a.40.2"
-240p/-240p.m3u8
```

If stream name contains no suffix, HLS will be played without ABR support.

Transcoding to higher resolutions prevention

If stream transcoding to higher resolutions is disabled on HLS ABR Edge server

```
cdn_strict_transcoding_boundaries=true
```

then stream variants conforming to higher resolution profiles will not be cut for the stream, and will not be vailable to a player.

Known limits

1. HLS Edge can be used for HLS streaming only, client sessions of another kinds will not work.

2. Stream recording, snapshots, mixing, stream capturing from another server and other captured stream management functions will not work.

HLS ABR on a single node

In most cases, it's more convenient to use CDN to play HLS ABR, because this is more scalable solution at server performance point. However, since build 5.2.1582 it is possible to transcode a stream to a certain HLS ABR qualities on a single server

```
hls_abr_enabled=true
hls_abr_with_cdn=false
```

In this case, HLS preloader and default transcoding must be disabled because a stream will be transcoded by defined profiles

```
hls_preloader_enabled=false
hls_player_width=0
hls_player_height=0
```

FPS equalizing should also be enabled

```
transcoder_align_encoders=true
video_filter_enable_fps=true
video_filter_fps=30
video_filter_fps_gop_synchronization=60
```

HLS ABR transcoding quality profiles should be set in `/usr/local/FlashphonerWebCallServer/conf/hls_abr_profiles.yml` file

```
profiles:
  -180p:
    audio:
      codec : mpeg4-generic
      rate : 48000
    video:
      height : 180
      bitrate : 300
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30

  -240p:
    audio:
      codec : mpeg4-generic
      rate : 48000
    video:
      height : 240
      bitrate : 500
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30

  -480p:
    audio:
      codec : mpeg4-generic
      rate : 48000
    video:
      height : 480
      bitrate : 1000
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30

  -720p:
    audio:
      codec : mpeg4-generic
      rate : 48000
    video:
      height : 720
      bitrate : 1500
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30
```

Low Latency HLS is also supported for HLS ABR

```
hls_ll_enabled=true
hls_new_http_stack=true
```

When using HLS ABR on a single server, any published stream will be transcoded to a number of qualities. This requires a lot of server CPU cores and RAM.

Transcoding to higher resolutions prevention

Since build 5.2.1611, if a stream published has a less picture height than some profiles listed in `hls_abr_profiles.yml`, then all the variants with higher resolutions will not be encoded and will not be included to manfest. For example, the manifest will look as follows when original stream has 960x540 resolution:

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=300000,RESOLUTION=320x180,CODECS="avc1.42e01f,mp4a.40.2"
180/180.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=500000,RESOLUTION=428x240,CODECS="avc1.42e01f,mp4a.40.2"
240/240.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=848x480,CODECS="avc1.42e01f,mp4a.40.2"
480/480.m3u8
```

because there will not be upscale to 1280x720.

If the stream resolution is lower than a minimal profile, this stream will be transcoded to the minimal available profile, and the only variant will be included to manifest

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=300000,RESOLUTION=320x180,CODECS="avc1.42e01f,mp4a.40.2"
180/180.m3u8
```

If there is a profile without both width and height parameters, the stream will be transcoded to its original resolution with GOP and FPS defined in the profile, and this variant will always be included to manifest:

```
profiles:
  original:
    video:
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30
```

Qualities order in HLS ABR manifest

Before build 5.2.1606, HLS ABR manifest qualities are sorted in profile names alphabetical order, for example, such `cdn_profiles.yml` or `hls_abr_profiles.yml`

```
profiles:
  360:
    video:
      height : 360
      bitrate : 1000
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30

  720:
    video:
      height : 720
      bitrate : 2000
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30

  1080:
    video:
      height : 1080
      bitrate : 2500
      codec : h264
      codecImpl : OPENH264
      gop : 60
      fps : 30
```

give the following manifest

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=2500000,RESOLUTION=1920x1080,CODECS="avc1.42e01f,mp4a.40.2"
1080/1080.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=640x360,CODECS="avc1.42e01f,mp4a.40.2"
360/360.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2000000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
720/720.m3u8
```

Since build 5.2.1606, manifest qualities order equals to profiles order in `cdn_profiles.yml` or `hls_abr_profiles.yml`

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=640x360,CODECS="avc1.42e01f,mp4a.40.2"
360/360.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2000000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
720/720.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2500000,RESOLUTION=1920x1080,CODECS="avc1.42e01f,mp4a.40.2"
1080/1080.m3u8
```

If there are two profiles with the same name in the setup, server will use only the last profile with the same name.

Force transcoding of a maximum ABR quality only if there are B-frames in a source stream

To reduce a server load while video encoding, since WCS build 5.2.1840 it is possible to transcode a maximum ABR quality (which is usually the original stream resolution and bitrate) only if there are B-frames in a source stream. The feature may be enabled by the following parameter

```
h264_b_frames_force_transcoding=true
```

In this case the server will detect B-frames in a stream analizing a certain frames count (10 by default)

```
frame_cnt_to_determine_their_type=10
```

If there are B-frames in the stream, the maximum ABR quality will be transcoded and will be available for playback in the HLS manifest

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=640x360,CODECS="avc1.42e01f,mp4a.40.2"
360/360.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2000000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
720/720.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2500000,RESOLUTION=1920x1080,CODECS="avc1.42e01f,mp4a.40.2"
1080/1080.m3u8
```

If there are no B-frames in the stream, the maximum ABR quality will not be transcoded

```
#EXTM3U
#EXT-X-STREAM-INF:BANDWIDTH=1000000,RESOLUTION=640x360,CODECS="avc1.42e01f,mp4a.40.2"
360/360.m3u8
#EXT-X-STREAM-INF:BANDWIDTH=2000000,RESOLUTION=1280x720,CODECS="avc1.42e01f,mp4a.40.2"
720/720.m3u8
```

The original quality should be requested separately from a playing client.

Since build 5.2.1916, the feature is available for HLS ABR in CDN. To activate it, all the CDN servers should be updated to build 5.2.1916 or newer, and the following parameters should be set on Edge server

```
cdn_strict_transcoding_boundaries=true
h264_b_frames_force_transcoding=true
```

Maximum playlist size

Maximum playlist size in segments is defined by the following parameter

```
hls_list_size=8
```

By default, HLS playlist size is 8 segments. Note that HLS cut is just started, a segments quantity in the first playlists will be less than value set.

HLS segments storage

Using disk

In builds before 5.2.1713 HLS segments are written to server hard drive to the `/usr/local/FlashphonerWebCallServer/hls` folder by default. Starting from build 5.2.687, the folder for saving segments can be changed using the following parameter

```
hls_dir=/usr/local/FlashphonerWebCallServer/hls
```

(Preloader folder is configured separately with parameter hls_preloader_dir.)

The number of stored segments corresponds to the specified playlist size. The less segments number, the less playback latency. However subscribers with poor channel could request HLS segments which are already gone from playlist and disk if playlist is short. To fix it, since build5.2.581the certain number of segments can be stored on disk after they gone from playlist. This feature can be enabled by the following parameter

```
hls_hold_segments_before_delete=true
```

By default, 5 last segments will be stored

```
hls_hold_segments_size=5
```

For example, if playlist contains 3 segments

```
#EXTM3U
#EXT-X-VERSION:8
#EXT-X-TARGETDURATION:11
#EXT-X-MEDIA-SEQUENCE:15
#EXT-X-DISCONTINUITY-SEQUENCE:1
#EXTINF:3.415,
test_017.ts
#EXTINF:10.417,
test_018.ts
#EXTINF:9.084,
test_019.ts
```

3 current segments and 5 previous segments will be stored on disk

```
test_012.ts
test_013.ts
test_014.ts
test_015.ts
test_016.ts
test_017.ts
test_018.ts
test_019.ts
```

Using memory

Under high load, for example on HLS streaming dedicated server, segments reading from hard drive to send them to subscribers can increase the latency. In this case, HLS segments storing in memory should be enabled

```
hls_store_segment_in_memory=true
```

Segments will be read from memory to send them to subscribers. Note this will require more Java heap memory.

Since build5.2.1713HLS segments are stored in memory by default.

Debug logs for HLS session

For an error report, debug logging can be enabled for HLS sessions using CLI

```
update node-setting --value true hls_enable_session_debug
```

Note that config flashphoner.properties will be overwritten after the command.

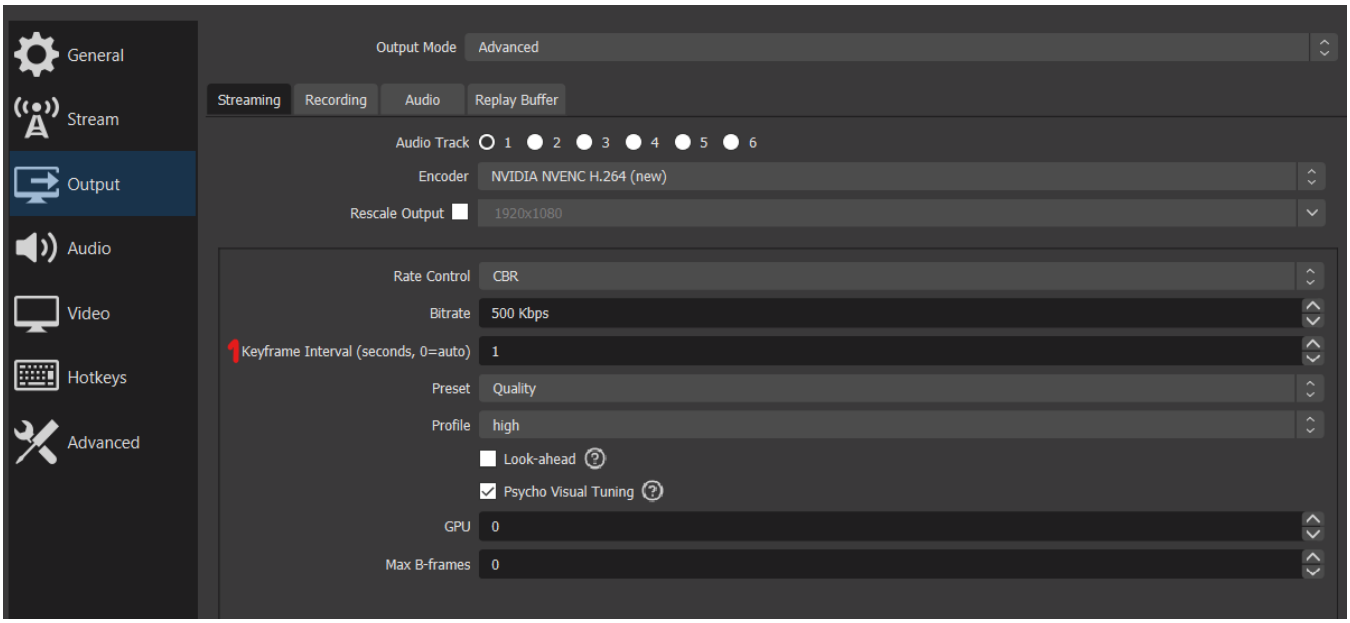
Low Latency HLS support

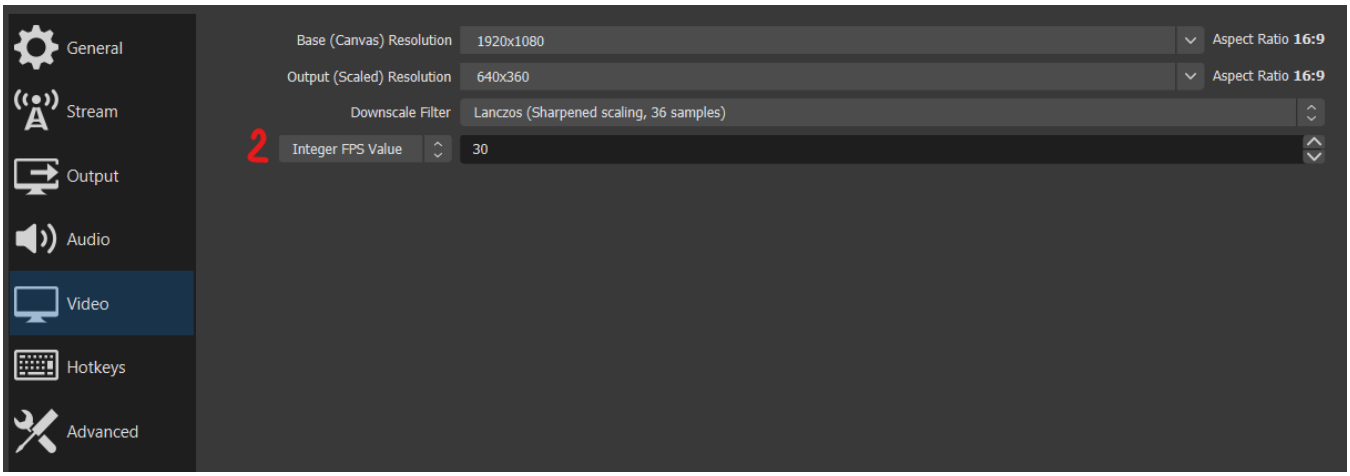
Since WCS build 5.2.1181, Low Latency HLS (LL HLS) is supported. The feature may be enabled with the following parameters

```
hls_ll_enabled=true
hls_new_http_stack=true
```

In this case, players supporting LL HLS (HLS.JS for example) will play additional HLS segments reducing playback latency comparing with players which do not play those segments (VideoJS for example).

It is necessary to provide a stable FPS and keyframe interval for a published stream to play it as LL HLS correctly. Therefore, it is recommended to publish the source strem as RTMP with the following example parameters





Recommended settings for LL HLS playback

Since build 5.2.1345, the recommended settings to play Low latency HLS are follow:

```
hls_ll_enabled=true
hls_auto_start=true
hls_preloader_enabled=false
hls_player_width=848
hls_player_height=480
video_filter_enable_fps=true
video_filter_fps=30
video_encoder_h264_gop=60
```

Using HTTP/2 and HTTP/1

According to specification, LL HLS must be played using HTTP/2, i.e. via secure connection

```
https://wsc:8445/test/test.m3u8
```

It is also possible to use HTTP/1 via unsecure connection with WCS

```
http://wsc:8082/test/test.m3u8
```

Note that LL HLS via HTTP/1 works in main browsers except Safari, and this feature is not recommended to use in production.

LL HLS segments folder

By default, LL HLS segments are written to streams subdolder to the folder

```
ll_hls_dir=/usr/local/FlashphonerWebCallServer/ll-hls
```

If folder has changed

```
ll_hls_dir=/opt/ll-hls
```

it is necessary to set file access permissions using the command

```
/usr/local/FlashphonerWebCallServer/bin/webcallserver set-permissions
```

and restart WCS to apply the changes.

Maximum LL HLS playlist size

Maximum LL HLS playlist size in segments is defined by the following parameter

```
ll_hls_list_size=12
```

By default, LL HLS playlist size is 12 full segments. Note that LL HLS cut is just started, a full segments quantity in the first playlists will be less than value set.

LL HLS preloader

Since build 5.2.1729, aspecial preloader can be used for LL HLS as like as usual HLS

```
hls_preloader_enabled=true
```

LL HLS preloader files are placed by default to the folder

```
ll_hls_preloader_dir=/usr/local/FlashphonerWebCallServer/ll-hls/.preloader
```

The folder can be changed, for example

```
ll_hls_preloader_dir=/opt/preloader
```

Default LL HLS preloader consists of the following files, one per each standard video streams aspect ratio

```
16x9.mp4
2x1.mp4
4x3.mp4
```

If stream aspect ration is unknown, 16:9 preloader file will be used. If there are no preloader files at all, LL HLS stream cut will start without a preloader like

```
hls_preloader_enabled=false
```

Custom LL HLS preloader setup

A custom LL HLS preloader can be set up if necessary. The media files in three standard aspects 16:9, 4:3 и 2:1 should be prepared according to the following requirements:

- MP4 container, video codec H264, audio codec AAC
- the files should allow instant playback (MP4 `moov` atom must precede `mdat`one)
- the files should not containt B frames
- the file duration should be near 1 minute
- the file should have a smooth FPS
- keyframe interval should be near 2 seconds

Suppose a source files are prepared in a proper aspects in OBS Studio or in a video editor. Use the following command example to convert a preloader files to conform the requirements above:

```
ffmpeg -i 16x9-source.mp4 -bf 0 -acodec aac -vcodec h264 -preset ultrafast -g 60 -strict -2 -r 30 -ar 48000 -
movflags faststart -ss 00:00:00 -t 00:01:00 16x9.mp4
```

Then the default preloader files should be replaced by custom preloader files, and WCS should be restarted.

To restore default preloader it is enough to remove custom preloader files and restart WCS.

m4s container support

Since build 5.2.1626 m4s container is supported for HLS segments cut, and since build 5.2.1632 the container is enabled by default for HLS ABR too

```
ll_hls_fragmented_mp4=true
```

Since build 5.2.1724, m4s container is supported for HLS ABR in CDN.

You can switch back to the ts container if necessary

```
ll_hls_fragmented_mp4=false
```

Using a common network stack for HLS and Low Latency HLS

Since build 5.2.1749 the parameter allowing an unified network stack usage both for HLS and Low latency HLS is added. The parameter is enabled by default:

```
use_new_hls=true
```

In this case:

- m4s container is used by default to record an HLS segments
- parameters with `hls` prefix are applied both to HLS and LL HLS
- parameters with `ll_hls` prefox are applied to LL HLS and to m4s container

> Since build 5.2.1793, the parameter is removed. The unified network stack is always used to deliver both HLS and LL HLS segments.

Manifest URL setup

Since build 5.2.1852, an URL templates to request a stream main playlist (manifest) may be customized. By default, the following templates are used:

```
hls_path_template={streamName}/{streamName}.m3u8
hls_abr_path_template={streamName}{abrSuffix}/{streamName}{abrSuffix}.m3u8
```

Where:

- streamName - stream published name
- abrSuffix - HLS ABR stream suffix set by `hls_abr_stream_name_suffix` parameter

In this case, the following URLs should be used to get HLS manifest

```
https://wcs:8445/stream/stream.m3u8
```

and to get HLS ABR manifest

```
https://wcs:8445/stream-HLS-ABR-STREAM/stream-HLS-ABR-STREAM.m3u8
```

For example, if a fixed manifest name different for HLS ABR and non-ABR streams is preferred to use, then the following templates should be set

```
hls_path_template={streamName}/playlist.m3u8
hls_abr_path_template={streamName}/playlist{abrSuffix}.m3u8
```

In this case, the following URLs should be used to get HLS manifest

```
https://wcs:8445/stream/playlist.m3u8
```

and to get HLS ABR manifest

```
https://wcs:8445/stream/playlist-HLS-ABR-STREAM.m3u8
```

Known issues

1. Non-recoverable freeze of HLS stream played in iOS Safari through a CDN

Symptoms: one minute after publishing start image stops, sound continues to play

Solution:

b) enable transcoding on server using the following option in flashphoner.propertiesfile

```
disable_streaming_proxy=true
```

b) if transcoding is undesirable, set the followingoption inflashphoner.propertiesfile

```
hls_discontinuity_enabled=true
```

In this case, clicks are possible in audio, but video will not stop.

2. HLS segments writing stops when playing stream published in Firefox browser.

Symptoms: a few minutes after playback start HLS segments stop writing, in that case the stream directory in hls directory is not deleted, and messages in server log continue to appear

```
INFO HLSStreamManager - HLSStreamProviderKeepaliveThread-80 Remove hls channel
```

Publisher must publish stream again to recover.

Solution: use another browser to publish the stream which supposed to be played via HLS.

3. No video for the first subscriber when playing HLS in Safari on iOS 12.4

Symptoms: no video for the first subscriber when playing a stream as HLS in Safari on iOS 12.4, however video is played normally if there are other HLS-subscribers to the stream

Solution: set the minimum HLS playlist segments count not less than 2

```
hls_min_list_size=2
```

4. No video for any subscriber when playing RTMP stream as HLS in Safari on iOS 12.4 if the following parameter is set to

```
hls_auto_start=true
```

Symptoms: no video for any subscriber when playing RTMP stream as HLS in Safari on iOS 12.4

Soluition: use mono sound when a file with stereo sound track is published, for example, set the followint command line options for ffmpeg

```
-acodec aac -ac 1
```

5. If stream transcoded by CDN is played as HLS, and if stream aspect ratio is changed during transcoding, HLS preloader is shown by original stream aspect ratio

Symptoms: when stream transcoded is requested by setting profile name, test-640x480p for example, 16x9 preloader is shown if original stream is 1280x720

Solution: turn on aspect ratio preserving on Transcoder CDN nodes

```
video_transcoder_preserve_aspect_ratio=true
```

6. If the source stream contains B-fames, the picture can twitch in some players

Symptoms: a strong picture twitching while playing a stream via HLS, this may lokk like low FPS

Solution: update WCS to build 5.2.863, where the problem is solved

7. Audio may be missed on the first connection to the stream when playing native LL HLS in Safari browser

Symptoms: stream is playing as LL HLS but there is no sound

Solution: update WCS to build 5.2.1345 or newer where the issue is solved

8. Chrome browser on Ubuntu 22.04 may raise CORS error while downloading HLS playlists via HTTPS

Symptoms: Chrome browser on Ubuntu 22.04 plays HLS via HTTPS normally, then CORS error occurs while downloading another playlist

Solution: do not send HTTP requests from Chrome to the same site HLS via HTTPS is playing from

9. LL HLS ABR stream may be played with a big delay in iOS Safari 16.

Symptoms: all the subscribers using iOS Safari 16, play LL HLS ABR stream with a big delay (more than 20 seconds) from published stream

Solution: update WCS to build5.2.1677 to use m4s container by default for LL HLS and wait for a posssible fix in iOS Safari 17

10. HLS ABR may not be played if m4s container is used

Symptoms: HLS ABR segments are not cut with server log message

```
02:18:01,957 ERROR HlsAbrStreamProvider - HLS-HTTPS-pool-5-thread-1 Failed to check stream null
java.lang.NullPointerException
```

Solution: update WCS to build5.2.1677 where the issue is resolved

11. VLC requires LL HLS manifest to include at least 4-6 segments for the first subscriber

Symptoms: audio and video are out of sync when VLC plays LL HLS in m4s container, or playback is freezing while switching a quality in LL HLS ABR

Solution: update WCS to build5.2.1677 and increase a minimal manifest size

```
hls_min_list_size=6
```

12. Audio only HLS stream in ts container is playing with a notable clicks in Safari browser

Symptoms: Audio only HLS stream clicks while playing in native HTML5 player in Safari browser

Solution: since build 5.2.1690, use m4s container for audio only streams

13. Encoder resources leak may appear under a high load when using HLS ABR

Symptoms: when HLS ABR is used and server CPU load is high (for instance, an encoding profiles number for all the streams published exceeds CPU capabilities), encoding resources may not be freed after publishing is stopped, and this may be visible on server statistics page, for example

```
streams_hls=0
...
native_resources.video_encoders=5
```

Solution: update WCS to build 5.2.1947 and set the following parameter

```
handler_async_disconnect=false
```