

# 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](#)
- [HLS playback authentication using REST hook](#)
- [Adding cross-domain access control HTTP headers for HLS playback](#)
- [Using nginx as reverse proxy for HLS playback](#)
- [Returning of static HTML pages on HLS port](#)
- [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 from [other supported sources](#) of broadcasting such as web cameras and professional video capturing device, SIP calls and so on.

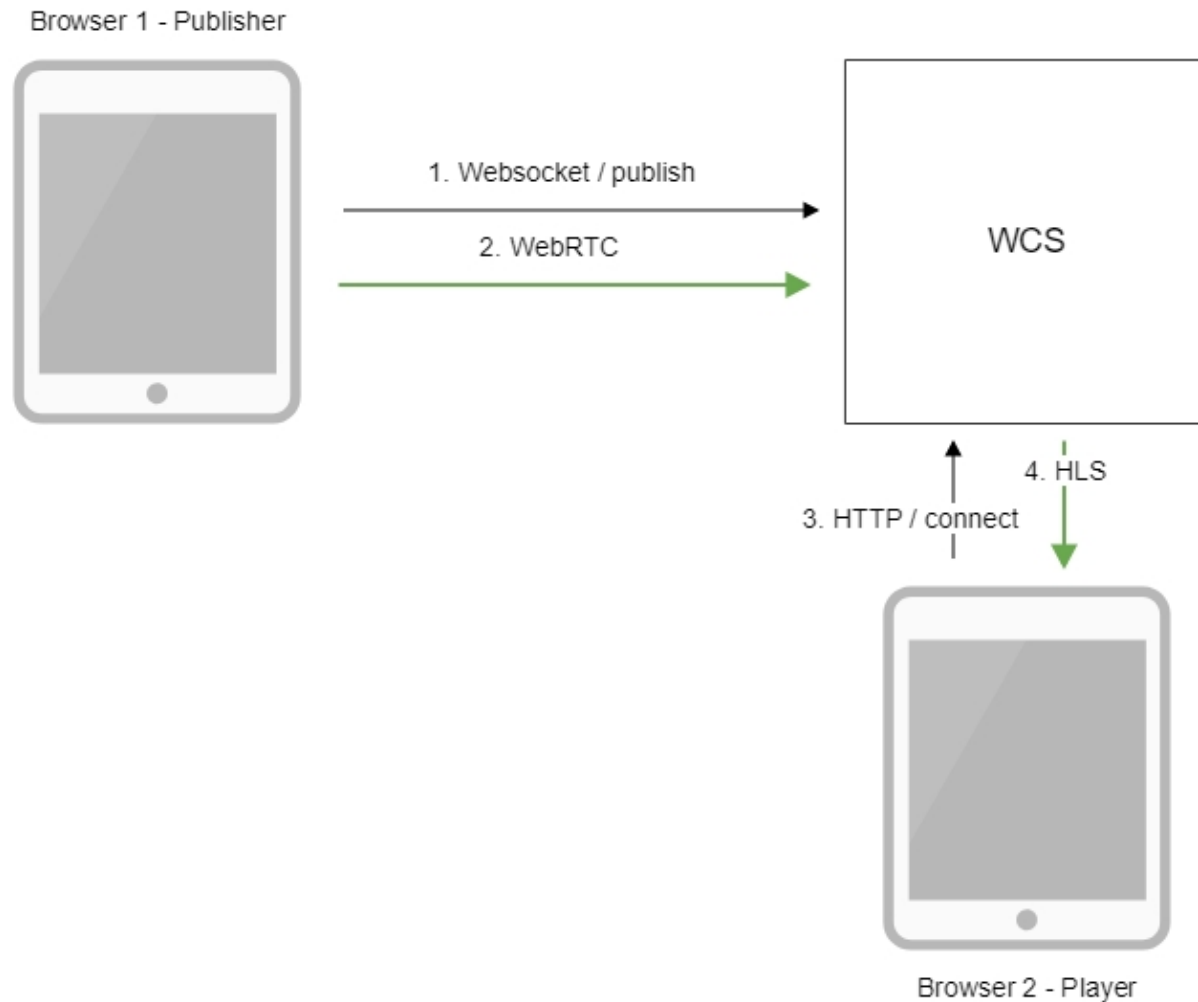
## Supported platforms and browsers

|         | 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;
- the [Two Way Streaming](#) web application to publish the stream
- the [HLS Player Minimal](#) web application to publish the stream

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

## Two-way Streaming

Local



test Stop

Player



3e87 Play Available

PUBLISHING

wss://test2.flashphoner.com:8443 Disconnect

ESTABLISHED

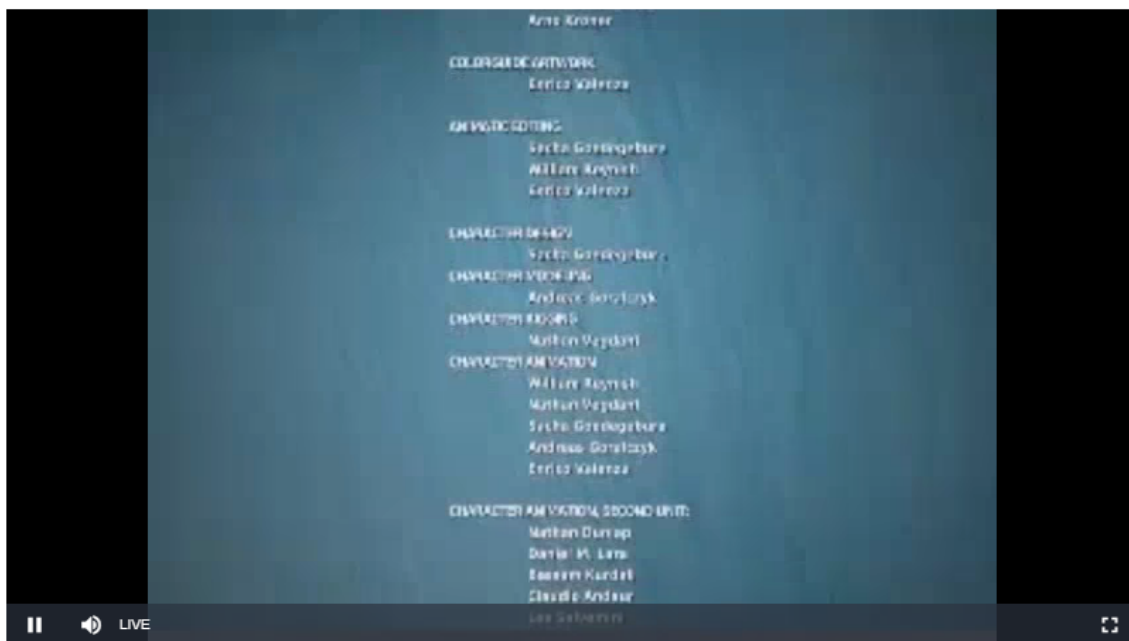
3. open the HLS Player Minimal web application. In the Stream field paste the identifier of the stream and click Play. The stream starts playing:

# HLS Player Minimal

## WCS

## Stream

## Auth

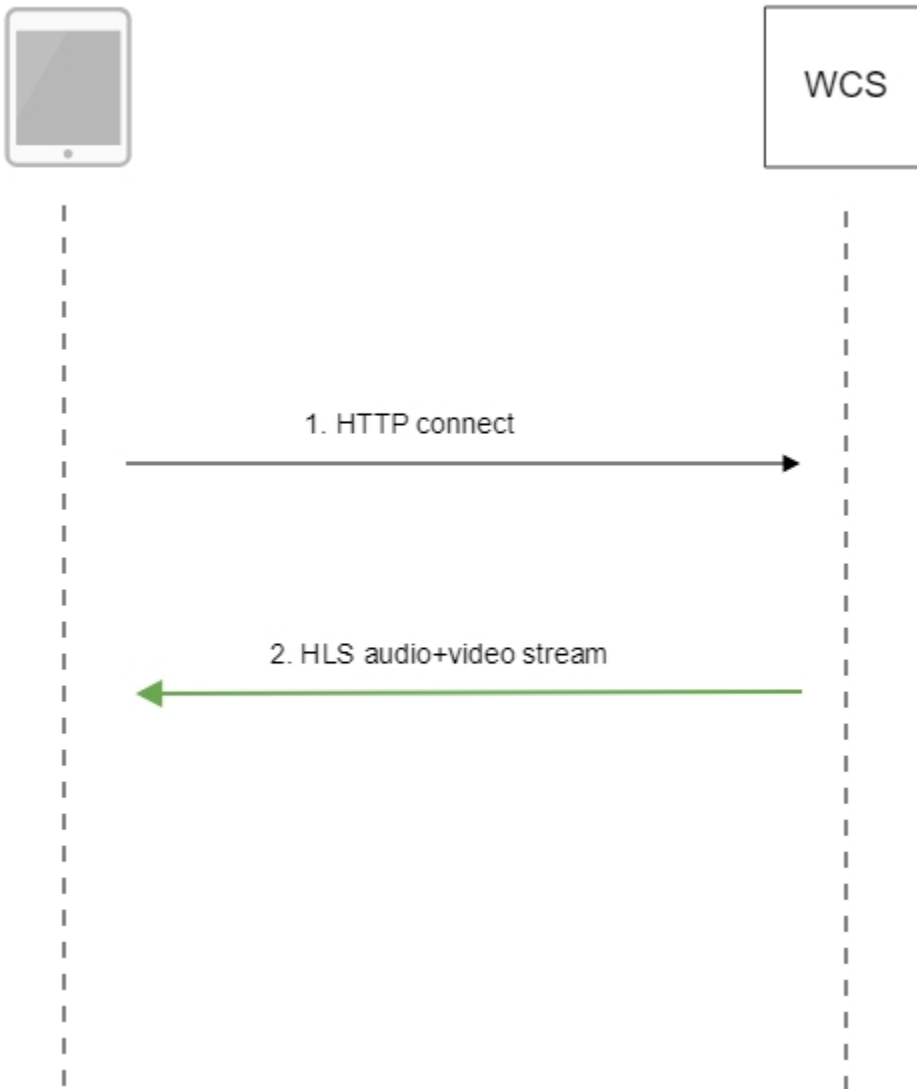


## Call flow

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

## 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 in [flashphoner.properties](#) file

```
hls_auth_enabled=true
```

At client side, token parameter must be added to HLS URL to pass to WCS server a token obtained for example from backend server:

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

[REST hook](#)/playHLS must be implemented on backend server. WCS server will send query to backend with token received from client

```
URL:http://localhost:8081/apps/EchoApp/playHLS
OBJECT:
{
  "nodeId" : "NTk1tLorQ00l1GbPJufexrKceubGCR0k@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.

## Adding cross-domain access control HTTP headers for HLS playback

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

```

> Transmission Control Protocol, Src Port: 8082, Dst Port: 57994, Seq: 1, Ack: 421, Len: 506
  ▾ Hypertext Transfer Protocol
    > HTTP/1.1 200 OK\r\n
      Connection: keep-alive\r\n
      Content-Type: application/x-mpegURL\r\n
    > Content-Length: 308\r\n
      Access-Control-Allow-Origin: *\r\n
      Access-Control-Allow-Methods: GET\r\n
      Access-Control-Max-Age: 3000\r\n
      \r\n
      [HTTP response 1/29]
      [Time since request: 0.000904000 seconds]
      [Request in frame: 24]
      [Next request in frame: 879]
      [Next response in frame: 881]
      File Data: 308 bytes

```

```

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 in [flashphoner.properties](#) file:

```

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:

```

> Transmission Control Protocol, Src Port: 8082, Dst Port: 58646, Seq: 1, Ack: 554, Len: 476
  ▾ Hypertext Transfer Protocol
    > HTTP/1.1 200 OK\r\n
      Connection: keep-alive\r\n
      Content-Type: application/x-mpegURL\r\n
    > Content-Length: 177\r\n
      Access-Control-Expose-Headers: Accept-Ranges, Content-Range, Content-Encoding, Content-Length\r\n
      Access-Control-Allow-Origin: *\r\n
      Access-Control-Allow-Methods: GET, HEAD\r\n
      Access-Control-Max-Age: 3000\r\n
      \r\n
      [HTTP response 1/9]
      [Time since request: 3.796755000 seconds]

```

## 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 set in [flashphoner.properties](#) file

```
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.

## 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.properties](#) file

```
disable_streaming_proxy=true
```

b) if transcoding is undesirable, set the following option in [flashphoner.properties](#) file

```
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.