

From a web camera in a browser via WebRTC

Overview

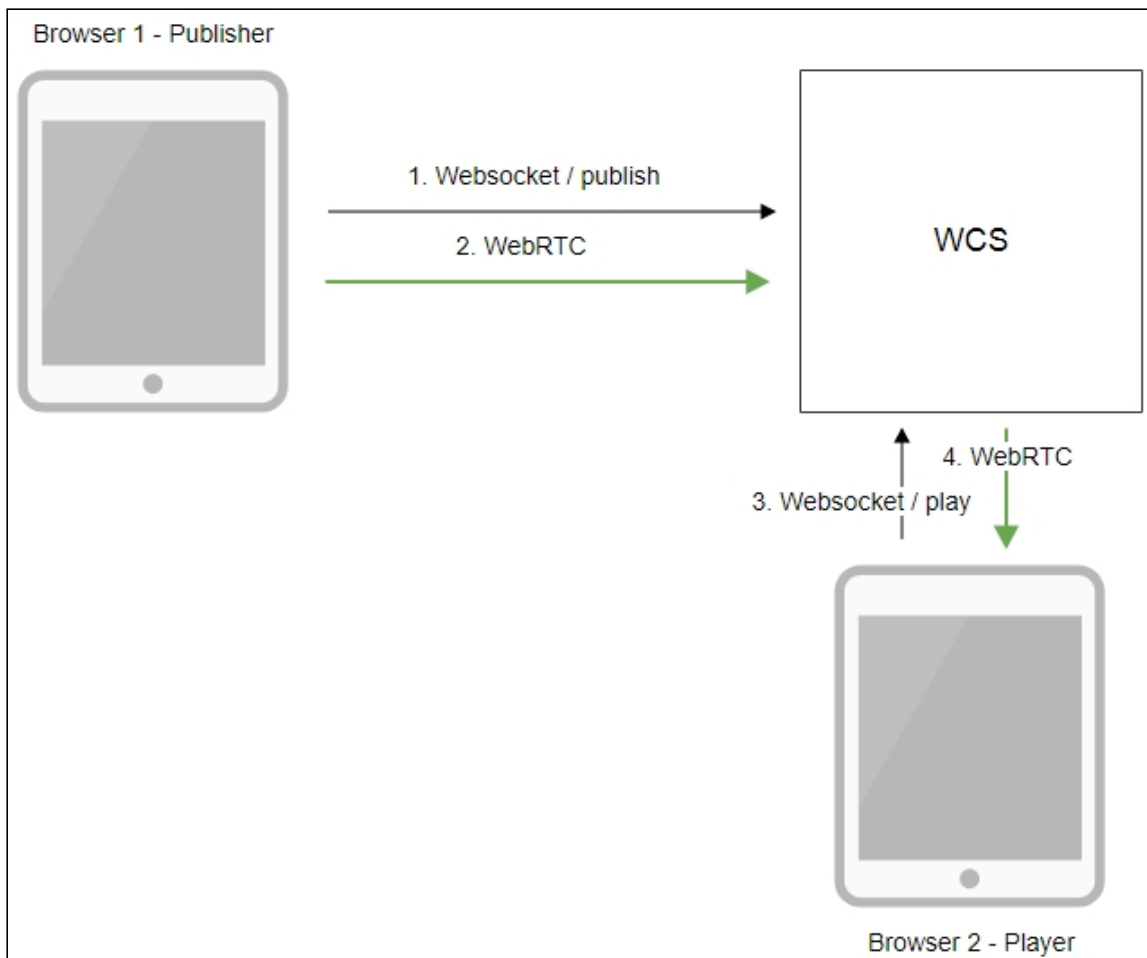
Supported platforms and browsers

	Chrome	Firefox	Safari	Edge
Windows	✓	✓	✗	✓
Mac OS	✓	✓	✓	✓
Android	✓	✓	✗	✓
iOS	✓	✓	✓	✓

Supported codecs

- Video: H.264, VP8
- Audio: Opus, PCMA, PCMU, G722, G729

Operation flowchart

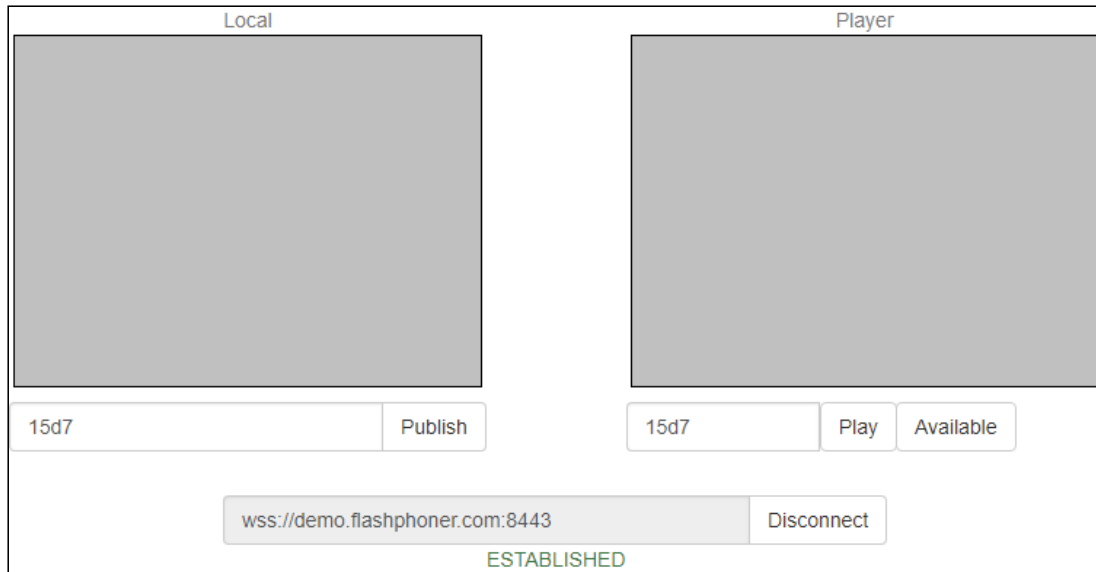


1. The browser connects to the server via the Websocket protocol and sends the `publishStream` command.
2. The browser captures the microphone and the camera and sends a WebRTC stream to the server.
3. The second browser establishes a connection also via Websocket and sends the `playStream` command.
4. The second browser receives the WebRTC stream and plays that stream on the page.

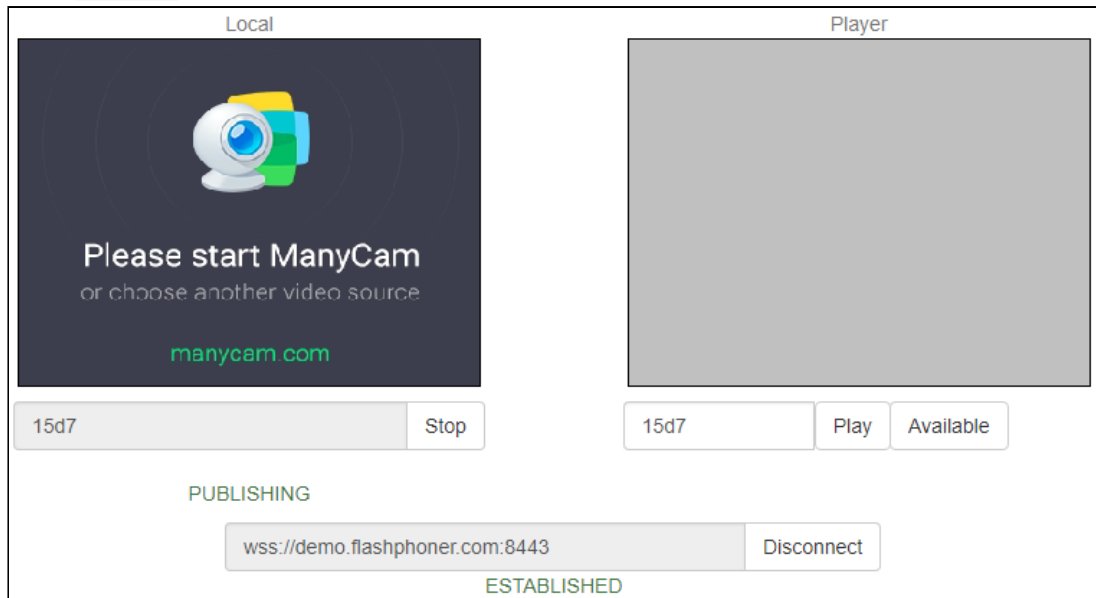
Quick manual on testing

1. For this test we use the demo server at `demo.flashphoner.com` and the Two Way Streaming web application `https://demo.flashphoner.com/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html`

2. Establish a connection with the server by clicking the **Connect** button

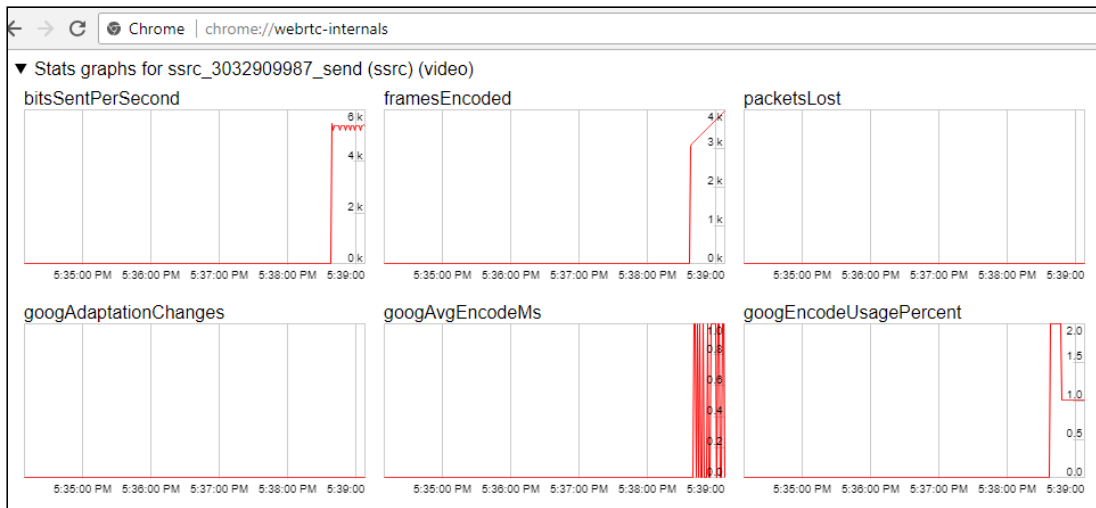


3. Click **Publish**. The browser captures the camera and sends the stream to the server



4. Make sure the stream is sent to the server and the system operates normally by opening

chrome://webrtc-internals



5. Open the Two Way Streaming app in a new window, click **Connect** and specify the stream ID, then click **Play**

The screenshot shows the Two Way Streaming app interface with the following components:

- Local View:** A grey rectangular area representing the local video feed.
- Player View:** A dark area with a ManyCam logo and the text "Please start ManyCam or choose another video source" and "manycam.com".
- Stream ID:** A text input field containing "1327" and a "Publish" button.
- Player Controls:** A text input field containing "15d7" and buttons for "Stop" and "Available".
- Status:** The word "PLAYING" is displayed in green.
- Connection:** A text input field containing "wss://demo.flashphoner.com:8443" and a "Disconnect" button.
- Connection State:** The word "ESTABLISHED" is displayed in green at the bottom.

6. Playback diagrams in `chrome://webrtc-internals`

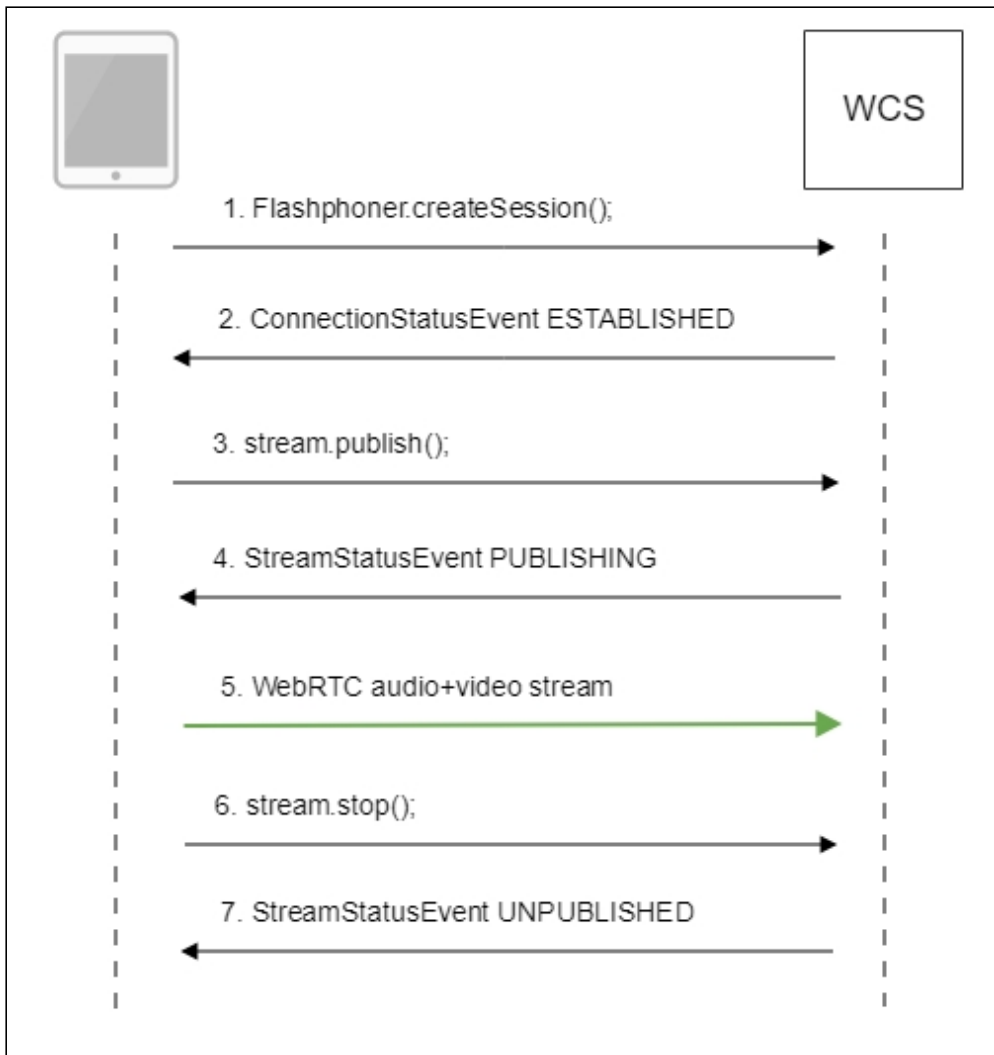


Call flow

Below is the call flow based on the Two Way Streaming example

[two_way_streaming.html](#)

[two_way_streaming.js](#)



1. Establishing connection to the server

`Flashphoner.createSession()` [code](#)

```

Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function (session) {
    setStatus("#connectStatus", session.status());
    onConnected(session);
}).on(SESSION_STATUS.DISCONNECTED, function () {
    setStatus("#connectStatus", SESSION_STATUS.DISCONNECTED);
    onDisconnected();
}).on(SESSION_STATUS.FAILED, function () {
    setStatus("#connectStatus", SESSION_STATUS.FAILED);
    onDisconnected();
});
  
```

2. Receiving from the server the successful connection status

`SESSION_STATUS.ESTABLISHED` [code](#)

```

Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function (session) {
    setStatus("#connectStatus", session.status());
  }
  
```

```

    onConnected(session);
  }).on(SESSION_STATUS.DISCONNECTED, function () {
    setStatus("#connectStatus", SESSION_STATUS.DISCONNECTED);
    onDisconnected();
  }).on(SESSION_STATUS.FAILED, function () {
    setStatus("#connectStatus", SESSION_STATUS.FAILED);
    onDisconnected();
  });

```

3. Publishing the stream

`Stream.publish()` [code](#)

```

session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  receiveVideo: false,
  receiveAudio: false
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
  setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
  onPublishing(stream);
}).on(STREAM_STATUS.UNPUBLISHED, function () {
  setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
  onUnpublished();
}).on(STREAM_STATUS.FAILED, function () {
  setStatus("#publishStatus", STREAM_STATUS.FAILED);
  onUnpublished();
}).publish();

```

4. Receiving from the server the successful publishing status

`STREAM_STATUS.PUBLISHING` [code](#)

```

session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  receiveVideo: false,
  receiveAudio: false
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
  setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
  onPublishing(stream);
  ...
}).publish();

```

5. Sending audio-video stream via WebRTC

6. Stopping publishing the stream

`Stream.stop()` [code](#)

```

function onPublishing(stream) {
  $("#publishBtn").text("Stop").off('click').click(function () {
    $(this).prop('disabled', true);
    stream.stop();
  }).prop('disabled', false);
}

```

```
$("#publishInfo").text("");  
}
```

7. Receiving from the server an even confirming successful unpublishing

`STREAM_STATUS.UNPUBLISHED` [code](#)

```
session.createStream({  
  name: streamName,  
  display: localVideo,  
  cacheLocalResources: true,  
  receiveVideo: false,  
  receiveAudio: false  
  ...  
}).on(STREAM_STATUS.UNPUBLISHED, function () {  
  setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);  
  onUnpublished();  
  ...  
}).publish();
```

Stream publishing with local video playback in VR Player

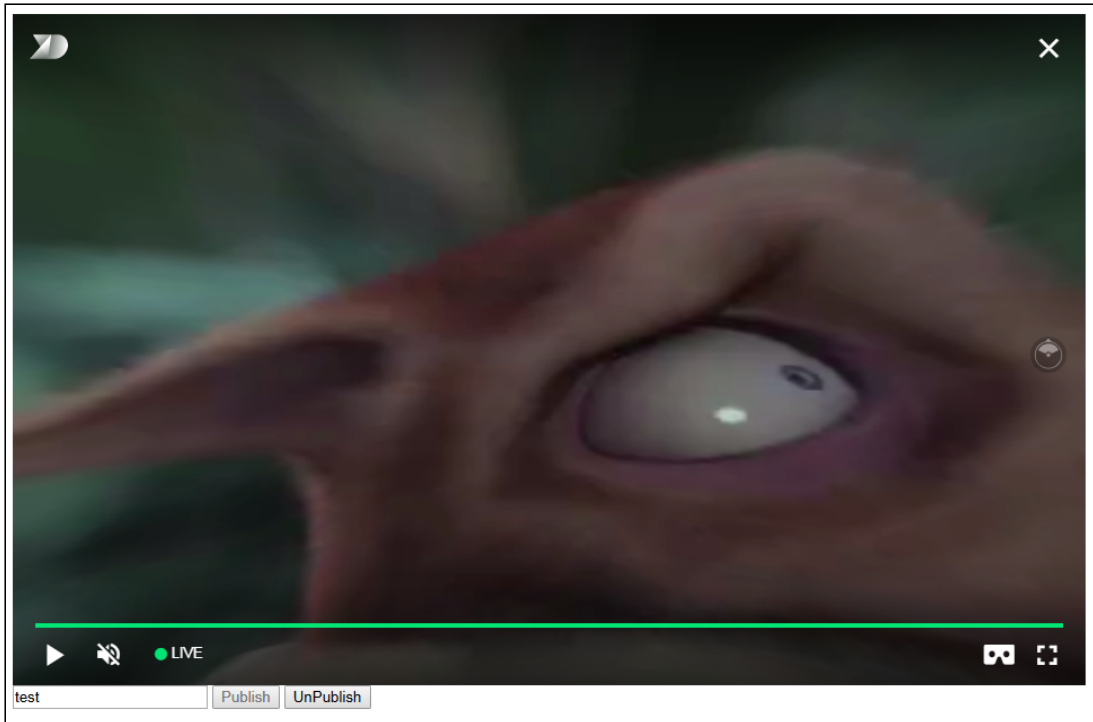
When stream is captured from webcam it is possible to play a local video in a browser-based VR player, [Delight Player](#) for example. This way stream can be played in virtual and mixed reality devices if one of browsers supports work on this device. [JavaScript and HTML5 features](#) are used to integrate a custom player.

Testing

1. For test we use:

- WCS server
- test page with [Delight](#) VR player to play stream while publishing

2. Set stream name `test` and press `Publish`. Stream published is played in Delight player



Player page example code

1. Declaration of video element to play the stream, stream name input field and `Publish/Unpublish` buttons

```
<div style="width: 50%;">
  <dl8-live-video id="remoteVideo" format="STEREO_TERPON" muted="true">
    <source>
  </dl8-live-video>
</div>
<input class="form-control" type="text" id="streamName"
placeholder="Stream Name">
<button id="publishBtn" type="button" class="btn btn-default"
disabled>Publish</button>
<button id="unpublishBtn" type="button" class="btn btn-default"
disabled>UnPublish</button>
```

2. Player readiness event handling

```
document.addEventListener('x-dl8-evt-ready', function () {
  dl8video = $('#remoteVideo').get(0);
  $('#publishBtn').prop('disabled', false).click(function() {
    publishStream();
  });
});
```

3. Creating mock elements to play a stream

```
var mockLocalDisplay = $('<div></div>');
var mockLocalVideo = $('<video></video>', {id: 'mock-LOCAL_CACHED_VIDEO'});
mockLocalDisplay.append(mockLocalVideo);
```

4. Establishing connection to the server and stream creation

```
var video = dl8video.contentElement;
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function (session) {
    var session = Flashphoner.getSessions()[0];
    session.createStream({
        name: $('#streamName').val(),
        display: mockLocalDisplay.get(0)
    }).on(STREAM_STATUS.PUBLISHING, function (stream) {
        ...
    }).publish();
})
```

5. Publishing stream, playback start in VR player and **Unpublish** button handling

```
session.createStream({
    ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
    var srcObject = mockLocalVideo.get(0).srcObject;
    video.srcObject = srcObject;
    dl8video.start();
    mockLocalVideo.get(0).pause();
    mockLocalVideo.get(0).srcObject = null;
    $('#unpublishBtn').prop('disabled', false).click(function() {
        stream.stop();
        $('#publishBtn').prop('disabled', false);
        $('#unpublishBtn').prop('disabled', true);
        dl8video.exit();
    });
}).publish();
```

 [Full source code of the sample VR player page](#)



Audio and video tracks activity checking

Before build [5.2.533](#) media traffic presence in a stream can be controlled by the following parameter in [flashphoner.properties](#)

```
rtp_activity_detecting=true,60
```

By default, RTP activity checking is enabled. If no media packets from browser in 60 seconds, publisher connection will be closed with the message to log **Failed by RTP activity**.

Since build [5.2.533](#), video and audio RTP activity checking settings are split

```
rtp_activity_audio=true  
rtp_activity_video=true
```

Activity timeout is set by the following parameter

```
rtp_activity_timeout=60
```

Therefore, if video only streams are published to the server, audio RTP activity checking should be disabled

```
rtp_activity_audio=false
```

If audio only streams are published to the server, video RTP activity checking should be disabled

```
rtp_activity_video=false
```

RTP activity can be checked for publishing streams only, not for playing streams.

Disable tracks activity checking by stream name

Since build [5.2.1784](#) it is possible to disable video and audio tracks activity checking for the streams with names matching a regular expression

```
rtp_activity_audio_exclude=stream1  
rtp_activity_video_exclude=stream1
```

The feature may be useful for streams in which a media traffic can stop for a long time, for example, screen sharing streams from an application window

```
rtp_activity_audio_exclude=.*-screen$  
rtp_activity_video_exclude=.*-screen$
```

In this case tracks activity checking will not be applied to the tracks named like `conference-123-user-456-screen`

If Chrome browser sends empty video due to web camera conflict

Some Chrome versions does not return an error if web camera is busy, but publish a stream with empty video (black screen). In this case, stream publishing can be stopped by two ways: using JavaScript and HTML5 on client, or using server settings.

Stopping a stream with empty video on client side

Videotrack that Chrome browsers creates for busy web camera, stops after no more than one second publishing, then stream is send without a videotrack. In this case videotrack state (`readyState` variable) changes to `ended`, and corresponding `onended` event is generated that can be caught by web application. To use this event:

1. Add the registration function for `onended` event handler to web application script, in which stream publishing is stopped with `Stream.stop()`

```
function addVideoTrackEndedListener(localVideo, stream) {
  var videoTrack = extractVideoTrack(localVideo);
  if (videoTrack && videoTrack.readyState == 'ended') {
    console.error("Video source error. Disconnect...");
    stream.stop();
  } else if (videoTrack) {
    videoTrack.onended = function (event) {
      console.error("Video source error. Disconnect...");
      stream.stop();
    };
  }
}
```

2. Add function to remove event handler when stream is stopped

```
function removeVideoTrackEndedListener(localVideo) {
  var videoTrack = extractVideoTrack(localVideo);
  if(videoTrack) {
    videoTrack.onended = null;
  }
}
```

3. Add function to extract videotrack

```
function extractVideoTrack(localVideo) {
  return localVideo.firstChild.srcObject.getVideoTracks()[0];
}
```

4. Register event handler when publishing a stream

```
session.createStream({
  name: streamName,
  display: localVideo,
  ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
  addVideoTrackEndedListener(localVideo, stream);
  setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
  onPublishing(stream);
  ...
}).publish();
```


5. Remove event handler when stopping a stream

```
function onPublishing(stream) {
  $("#publishBtn").text("Stop").off('click').click(function () {
    $(this).prop('disabled', true);
    removeVideoTrackEndedListener(localVideo);
    stream.stop();
  }).prop('disabled', false);
  $("#publishInfo").text("");
}
```

Videotrack activity checking on server side

Videotrack activity checking for streams published on server is enabled by default with the following parameter in [flashphoner.properties](#) file

```
rtp_activity_video=true
```

In this case, if there is no video in stream, its publishing will be stopped after 60 seconds.

Video only stream publishing with constraints

In some cases, video only stream should be published while microphone is busy, for example video is published while voice phone call. To prevent browser access request to microphone, set the constraints for video only publishing:

```
session.createStream({
  name: streamName,
  display: localVideo,
  constraints: {video: true, audio: false}
  ...
}).publish();
```

Audio only stream publishing

In most cases, it is enough to set the constraints to publish audio only stream:

```
session.createStream({
  name: streamName,
  display: localVideo,
  constraints: {video: false, audio: true}
  ...
}).publish();
```

Audio only stream publishing in Safari browser

When audio only stream is published from iOS Safari browser with constraints, browser does not send audio packets. To workaround this, a stream should be published with video, then video should be muted:

```
session.createStream({
  name: streamName,
  display: localVideo,
  constraints: {video: true, audio: true}
  ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
  stream.muteVideo();
  ...
}).publish();
```

In this case, iOS Safari browser will send empty video packets (blank screen) and audio packets.

Disable resolution constraints normalization in Safari browser

By default, WebSDK normalizes stream publishing resolution constraints set in Safari browser. In this case, if width or height is not set, or equal to 0, then the picture resolution is forced to 320x240 or 640x480. Since WebSDK build [0.5.28.2753.109](#), it is possible to disable normalization and pass resolution constraints to the browser as is, for example:

```
publishStream = session.createStream({
  ...
  disableConstraintsNormalization: true,
  constraints {
    video: {
      width: {ideal: 1024},
      height: {ideal: 768}
    },
    audio: true
  }
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

H264 encoding profiles exclusion

Since build [5.2.620](#) some H264 encoding profiles can be excluded from remote SDP which is sent by server to browser. The profiles to exclude should be listed in the following parameter

```
webrtc_sdp_h264_exclude_profiles=4d,64
```

In this case, Main (4d) and High (64) profiles will be excluded, but Baseline (42) still remains:

```
a=rtptime:102 H264/90000
a=fmtp:102 level-asymmetry-allowed=1;packetization-mode=1;profile-level-id=42001f
a=rtptime:125 H264/90000
a=fmtp:125 level-asymmetry-allowed=1;packetization-mode=1;profile-level-id=42e01f
a=rtptime:127 H264/90000
a=fmtp:127 level-asymmetry-allowed=1;packetization-mode=0;profile-level-id=42001f
a=rtptime:108 H264/90000
a=fmtp:108 level-asymmetry-allowed=1;packetization-mode=0;profile-level-id=42e01f
```

This setting may be useful if some browser encodes B-frames for example using high profiles while hardware acceleration is enabled.

By default, no profiles will be excluded from SDP if they are supported by browser

```
webrtc_sdp_h264_exclude_profiles=
```

Content type management while publishing from Chromium based browser

In some cases, most browsers based on Chromium 91 aggressively assess a publishing channel quality and drop publishing resolution lower than set in constraints, even if channel is enough to publish 720p or 1080p stream. To workaround this behaviour, since WebSDK build [2.0.180](#) `videoContentHint` option was added:

```
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  receiveVideo: false,
  receiveAudio: false,
  videoContentHint: "detail"
  ...
}).publish();
```

In WebSDK builds before [2.0.242](#) this option is set to `detail` by default and forces browsers to keep the publishing resolution as set in constraints. However, browser can drop FPS in this case when publishing stream from some USB web cameras. If FPS should be kept no matter to resolution, the option should be set to `motion`

```
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  receiveVideo: false,
  receiveAudio: false,
```

```
videoContentHint: "motion"  
...  
}).publish();
```

Since WebSDK build 2.0.242, `videoContentHint` is set to `motion` by default. The `detail` or `text` values should be set only for screen sharing streaming in browser.

Since WebSDK build 2.0.204 `videoContentHint` selection is available in Media Device example

Send Video

Cam Logitech HD Webcam C61 ▼
Switch

Screen share off

Size 640 480

FPS 30

Bitrate min max
0 0

COD

CVO

Content Hint motion ▼

FPS management in Firefox browser

By default, Firefox is publishing video with maximum FPS shown by web camera driver for requested resolution. This value is 30 FPS for most of modern web cameras. Publishing FPS can be defined more strictly if necessary. To do this, disable constraints normalization:

```
session.createStream({  
  ...  
  disableConstraintsNormalization: true,  
  constraints: {  
    video: {  
      width: 640,  
      ...  
    }  
  }  
});
```

```

        height: 360,
        frameRate: { max: 15 }
    },
    audio: true
}
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
    ...
}).publish();

```

Note that Firefox can exclude the camera from the list while requesting camera and microphone access if camera driver does not provide a required combination of resolution and FPS. Also, Firefox can change a publishing resolution if there is only one resolution with required FPS in camera driver response.

Stereo audio publishing in browser

Audio bitrate should be more than 60000 bps to publish stereo in Opus codec from browser. This can be done by setting Opus codec parameters on client side

```

session.createStream({
  name: streamName,
  display: remoteVideo,
  constraints: {
    audio: {
      bitrate: 64000
    },
    ...
  }
  ...
}).publish();

```

or on server side

```

opus_formats = maxaveragebitrate=64000;stereo=1;sprop-stereo=1;

```

In this case, Firefox browser publishes stereo audio without additional setup.

Stereo audio publishing in Chrome based browsers

A certain client setup is required to publish stereo audio from Chrome. There are two ways to set this up depending on client implementation

Using Web SDK

If [Web SDK](#) is used in project, it is necessary to set the following constraint option:

```

session.createStream({
  name: streamName,
  display: localVideo,

```

```
constraints: {
  audio: {
    stereo: true
  },
  ...
}
...
}).publish();
```

Using Websocket API

If only [Raw Websocket API](#) is used in project, it is necessary to disable echo cancellation

```
var constraints = {
  audio: {
    echoCancellation: false,
    googEchoCancellation: false
  },
  ...
};
...
navigator.getUserMedia(constraints, function (stream) {
  ...
}, reject);
```

If echo cancellation is enabled, Chrome will publish mono audio even if stereo is set in Opus codec options.

How to bypass an encrypted UDP traffic blocking

Sometimes an encrypted UDP mediatraffic may be blocked by ISP. In this case, WebRTC stream publishing over UDP will fail with `Failed by RTP activity` error or with `Failed by ICE timeout`. To bypass this, it is recommended to use [TCP transport](#) at client side

```
session.createStream({
  name: streamName,
  display: localVideo,
  transport: "TCP"
  ...
}).publish();
```

Another option is to use [external or internal TURN server](#) or publish a stream via [RTMP](#) or [RTSP](#).

Known issues

1. If the web app is inside an iframe element, publishing of the video stream may fail.

Symptoms

IceServer errors in the browser console

Solution

Put the app out of iframe to an individual page

2. Bitrate problems are possible when publishing a stream under Windows 10 or Windows 8 with hardware acceleration enabled in a browser

Symptoms

Low quality of the video, muddy picture, bitrate shown in `chrome://webrtc-internals` is less than 100 kbps.

Solution

Turn off hardware acceleration in the browser or use the VP8 codec to publish

3. Stream publishing with local video playback in VR Player does not work in legacy MS Edge

Symptoms

When stream is published in legacy MS Edge, local video playback does not start in VR Player

Solution

Use another browser to publish a stream or update legacy MS Edge to Chromium Edge

4. In some cases microphone does not work in Chrome browser while publishing WebRTC stream

Symptoms

Microphone does not work while publishing WebRTC stream, including example web applications out of the box

Solution

Turn off gain node creation in Chrome browser using WebSDK initialization parameter `createMicGainNode: false`

```
Flashphoner.init({
  flashMediaProviderSwfLocation: '../../../media-provider.swf',
  createMicGainNode: false
});
```

Note that [microphone gain setting](#) will not work in this case.

5. G722 codec does not work in Edge browser

Symptoms

WebRTC stream with G722 audio does not publish in Edge browser

Solution

Use another codec or another browser. If Edge browser must be used, exclude G722 with the following parameter

```
codecs_exclude_streaming=g722,telephone-event
```

6. Some Chromium based browsers do not support H264 codec depending on browser and OS version

Symptoms

Stream publishing does not work, stream playback works partly (audio only) or does not work at all

✓ Solution

Enable VP8 on server side

```
codecs=opus,...,h264,vp8,...
```

exclude H264 for publishing or playing on client side

```
publishStream = session.createStream({
  ...
  stripCodecs: "h264,H264"
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

Note that [stream transcoding](#) is enabled on server when stream published as H264 is played as VP8 and vice versa.

7. iOS Safari 12.1 does not send video frames when picture with certain resolution is published

🚩 Symptoms

When H264 stream is published from iOS Safari 12.1, subscriber receives audio packets only, publishers WebRTC statistics also shows audio frames only

✓ Solution

Enable VP8 on server side

```
codecs=opus,...,h264,vp8,...
```

exclude H264 for publishing or playing on client side

```
publishStream = session.createStream({
  ...
  stripCodecs: "h264,H264"
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

Note that [stream transcoding](#) is enabled on server when stream published as H264 is played as VP8 and vice versa.

8. Stream from built-in camera cannot be published in iOS Safari 12 and MacOS Safari 12 in some resolutions

Symptoms

Stream publishing from browser fails with error in console

```
Overconstrained error: width
```

Solution

- use only resolutions which passes [WebRTC Camera Resolution](#) test
- use external camera supporting resolutions as needed in MacOS Safari
- disable resolution constraints normalization and set `width` and `height` as `ideal`, see [example above](#).

9. Non-latin characters in stream name should be encoded

Symptoms

Non-latin characters in stream name are replaced to questionmarks on server side

Solution

Use JavaScript function `encodeURIComponent()` while publishing stream

```
var streamName = encodeURIComponent($('#publishStream').val());
session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  receiveVideo: false,
  receiveAudio: false
  ...
}).publish();
```

10. In some cases, server can not parse H264 stream encoded with CABAC

Symptoms

WebRTC H264 stream publishing does not work

Solution

- a) use lower encoding profile
- b) Enable VP8 on server side

```
codecs=opus, ..., h264, vp8, ...
```

exclude H264 for publishing or playing on client side

```
publishStream = session.createStream({
  ...
  stripCodecs: "h264,H264"
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

Note that [stream transcoding](#) is enabled on server when stream published as H264 is played as VP8 and vice versa.

11. WebRTC playback may not work in Firefox on MacOS Catalina

Symptoms

System warning `"libgmpopenh264.dylib" can't be opened because it is from an identified developer` is displayed, H264 WebRTC stream is not playing

Solution

Firefox uses a third-party library unsigned by the developer to work with H264. In accordance with macOS Catalina security policies, this is prohibited. To add an exception, go to `System Preferences - Security & Privacy - General - Allow apps downloaded from - App Store and identified developers`, find `"libgmpopenh264.dylib" was blocked from opening because it is not from an identified developer` and click `Open Anyway`

12. `ice_keep_alive_enabled=true` parameter is not used in latest WCS builds

Since build [5.2.672](#), the parameter

```
ice_keep_alive_enabled=true
```

is not used. ICE keep alive timeout is started automatically if WCS starts sending STUN keep alives first, for example while incoming SIP call or while pushing WebRTC stream to another server

13. MacOS Safari 14.0.2 (MacOS 11) does not publish stream from MacBook FaceTimeHD camera with aspect ratio 4:3

Symptoms

After stream publishing starts in the examples Two Way Streaming, Stream Recording etc, a browser stops sending video packets in 10 seconds, the black screen is shown on player side, publishing fails due to video traffic absence

Solution

a) Publish stream with aspect ratio 16:9 (for example, 320x180, 640x360 etc)

```
publishStream = session.createStream({
  ...
  constraints: {
    video: {
      width: 640,
      height: 360
    },
    audio: true
  }
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

b) Update Web SDK to build [0.5.28.2753.153](#) or newer, where default resolution for Safari browser is adopted to 16:9

c) Update MacOS to build 11.3.1, Safari to build 14.1 (16611.1.21.161.6)

14. Streams published on Origin server in [CDN](#) may not be played from Edge if some H264 encoding profiles are excluded

Symptoms

WebRTC H264 stream published on Origin is playing as audio only from Edge with VP8 codec shown in stream metrics

Solution

If some H264 encoding profiles are excluded on Origin

```
webrtc_sdp_h264_exclude_profiles=4d,64
```

the allowed H264 profiles should be explicitly set on Edge with the following parameter

```
profiles=42e01f
```

15. MacOS Safari 14.0.* stops sending video packets after video in muted and then unmuted due to [Webkit bug](#)

Symptoms

When `muteVideo()` then `unmuteVideo()` are applied, the publishing stops after a minute with `Failed by Video RTP activity` error

Solution

Update MacOS to build 11.3.1, Safari to build 14.1 (16611.1.21.161.6), in this build the Webkit bug seems to be fixed, the problem is not reproducing

16. When publishing from Google Pixel 3/3XL, the picture is strongly distorted in some resolutions

Symptoms

A local video is displaying normally, but is playing with a strong distortion (transverse stripes)

✓ Solution

Avoid the following resolutions while publishing stream from Google Pixel 3/3XL:

- 160x120
- 1920x1080

17. iOS Safari 15.1 requires from another side to enable image orientation extension when publishing H264 stream

🚩 Symptoms

Webpage crashes in iOS Safari 15.1 when stream publishing is started (Webkit bugs https://bugs.webkit.org/show_bug.cgi?id=232381 and https://bugs.webkit.org/show_bug.cgi?id=231505)

✓ Solution

a) enable image orientation extension support on client side in iOS Safari

```
session.createStream({
  name: streamName,
  ...
  cvoExtension: true
}).publish();
```

and in WCS builds before [5.2.1074](#) disable RTP bundle support

```
rtp_bundle=false
```

Since WCS build [5.2.1074](#) RTP bundle may not be disabled

b) use VP8 to publish a stream

```
session.createStream({
  name: streamName,
  ...
  stripCodecs: "H264"
}).publish();
```