In a browser via WebRTC

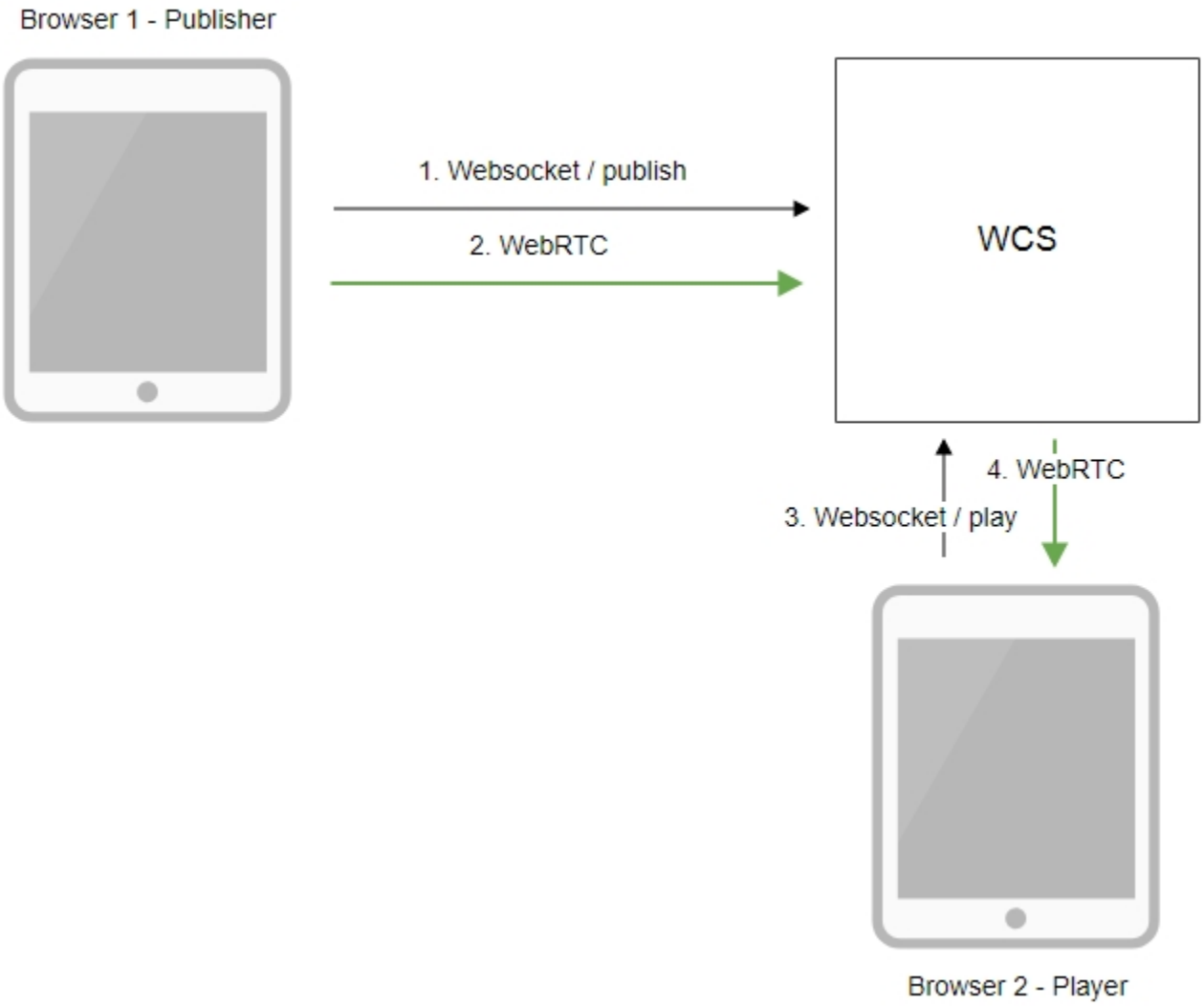
- Overview
    - Supported platforms and browsers
    - Operation flowchart
- Quick manual on testing
    - Publishing a video stream on the server and playing it via WebRTC in a browser
- Call flow
- Playing two or more streams on the same page
- WebRTC stream playback in custom player
    - Testing
    - Custom player page code sample
- Automatic stream playback
    - Autoplay issues in different browsers
        - Chrome
        - iOS Safari
- Audio playback tuning in iOS Safari
- Known issues

Overview

Supported platforms and browsers

	Chrome	Firefox	Safari 11	Edge
Windows	+	+		+
Mac OS	+	+	+	
Android	+	+		
iOS	-	-	+	

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 also via Websocket and sends the play command.
4. The second browser receives the WebRTC stream and plays this stream on the page.

Quick manual on testing

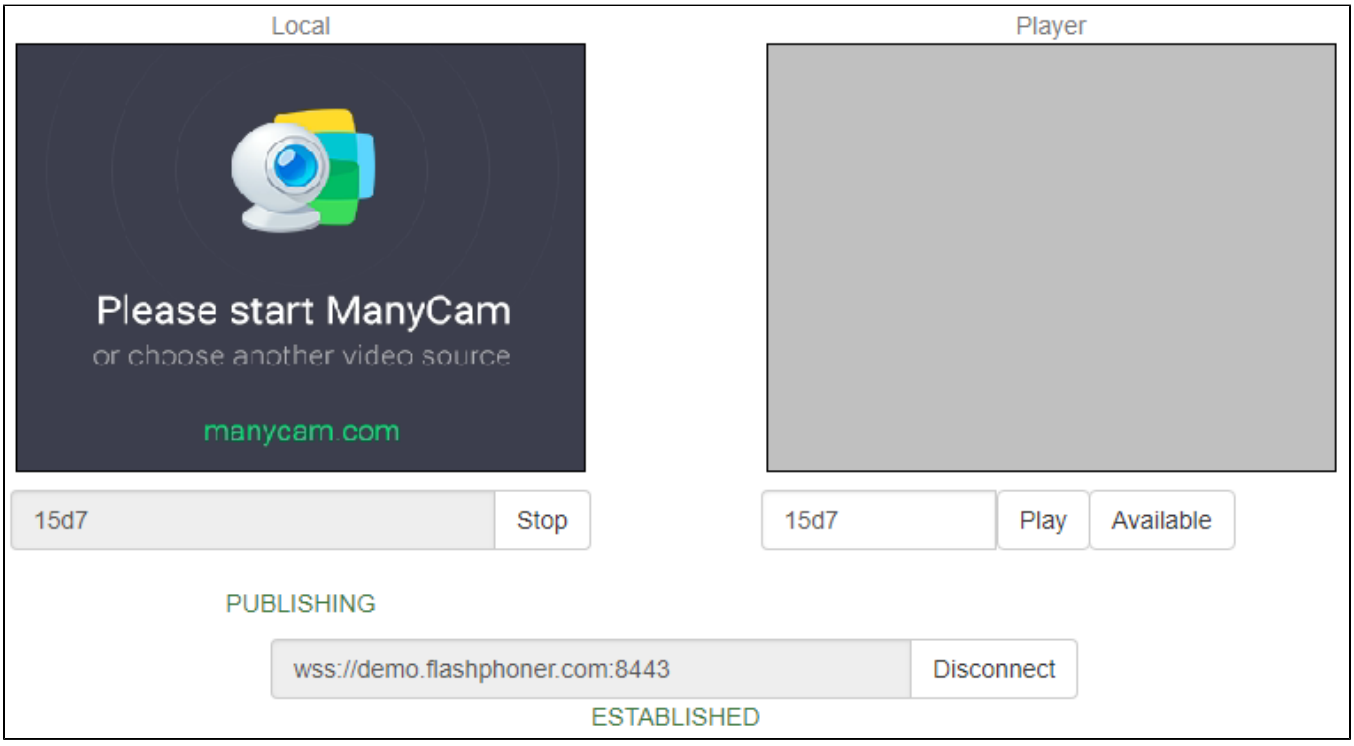
Publishing a video stream on the server and playing it via WebRTC in a browser

1. For this test we use the demo server at demo.flashphoner.com and the Two Way Streaming web application

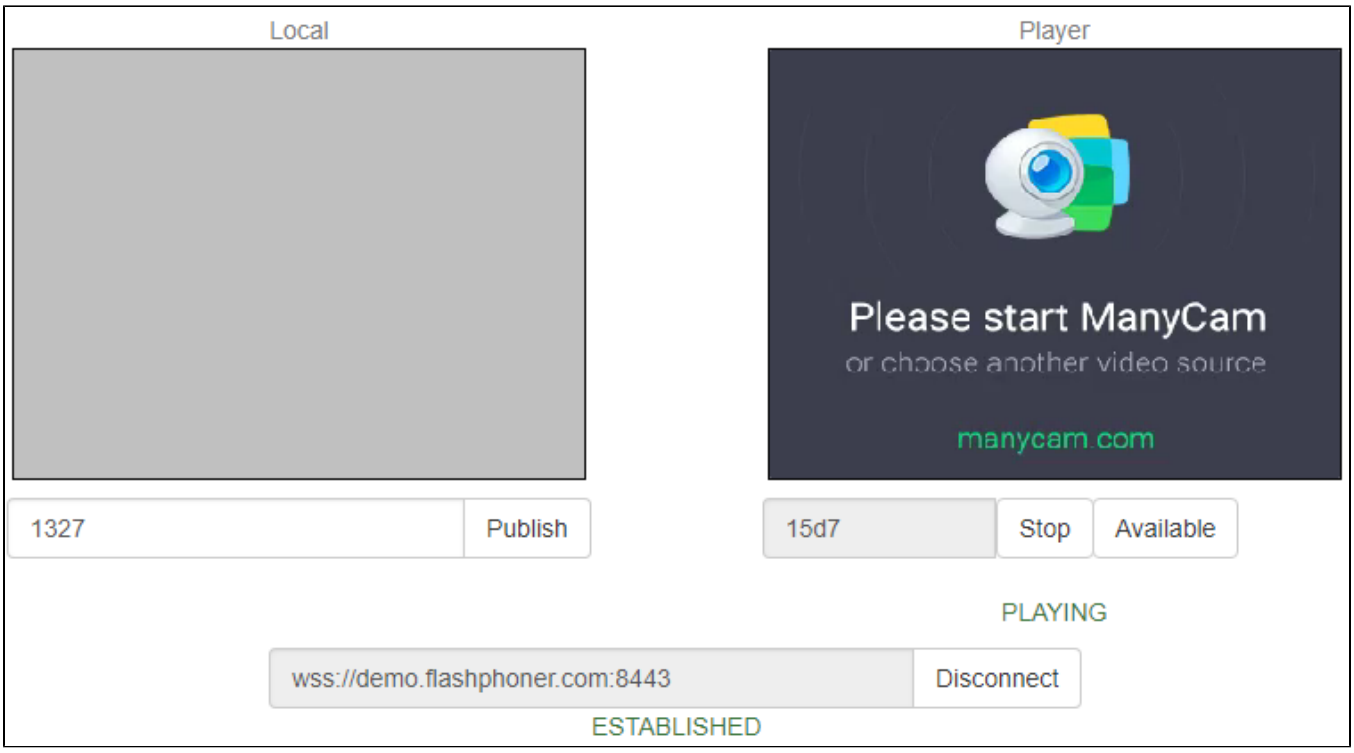
https://demo.flashphoner.com/client2/examples/demo/streaming/two_way_streaming/two_way_streaming.html

2. Establish a connection to the server using the Connect button

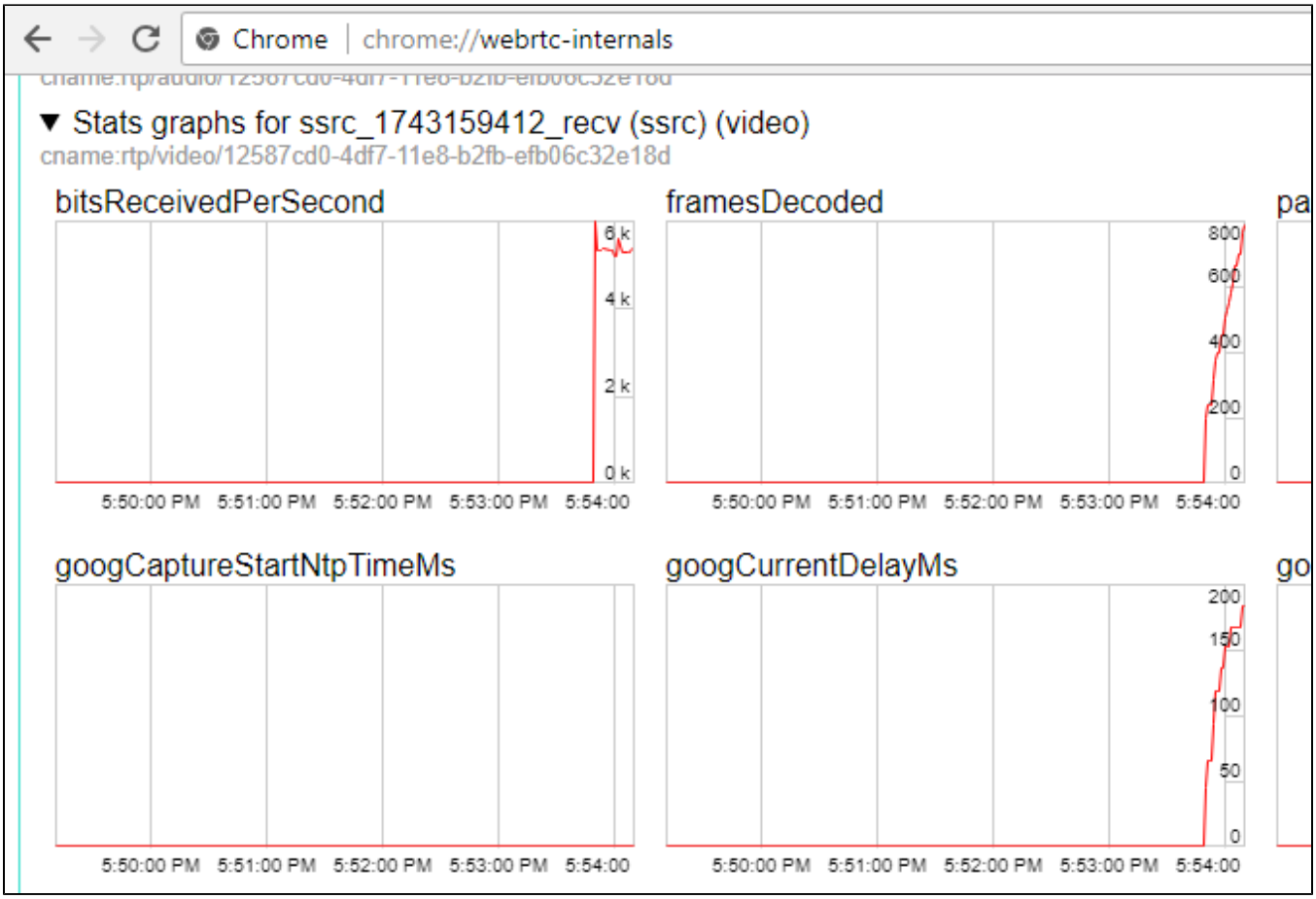
3. Click Publish. The browser captures the camera and sends the stream to the server.



4. Open Two Way Streaming in a separate window, click Connect and provide the identifier of the stream, then click Play.



5. Playback diagrams in chrome://webrtc-internals

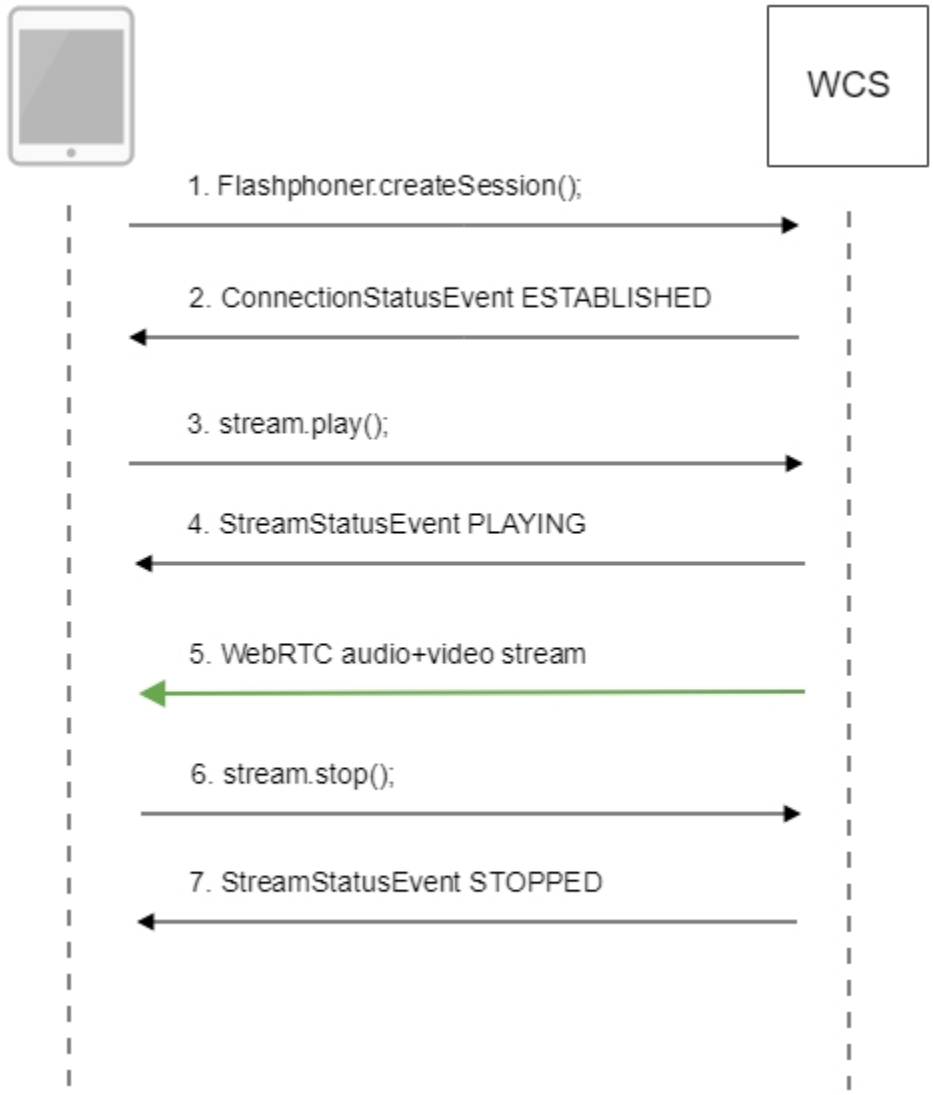


Call flow

Below is the call flow when using the Two Way Streaming example to play the stream

two_way_streaming.html

two_way_streaming.js



1. Establishing a 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 an event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

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

3. Playing the stream.

stream.play();code

```
    session.createStream({
        name: streamName,
        display: remoteVideo
        ...
    }).play();
```

4. Receiving from the server an event confirming successful playing of the stream.

StreamStatusEvent, status PLAYINGcode

```
    session.createStream({
        name: streamName,
        display: remoteVideo
    }).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function (stream) {
        setStatus("#playStatus", stream.status());
        onPlaying(stream);
    }).on(STREAM_STATUS.STOPPED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).play();
```

5. Receiving the audio and video stream via WebRTC

6. Stopping playing the stream.

stream.stop();code

```
function onPlaying(stream) {
    $("#playBtn").text("Stop").off('click').click(function () {
        $(this).prop('disabled', true);
        stream.stop();
    }).prop('disabled', false);
    $("#playInfo").text("");
}
```

7. Receiving from the server an event confirming the playback is stopped.

StreamStatusEvent, status STOPPEDcode

```
session.createStream({
    name: streamName,
    display: remoteVideo
}).on(STREAM_STATUS.PENDING, function(stream) {
    ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
    ...
}).on(STREAM_STATUS.STOPPED, function () {
    setStatus("#playStatus", STREAM_STATUS.STOPPED);
    onStopped();
}).on(STREAM_STATUS.FAILED, function (stream) {
    ...
}).play();
```

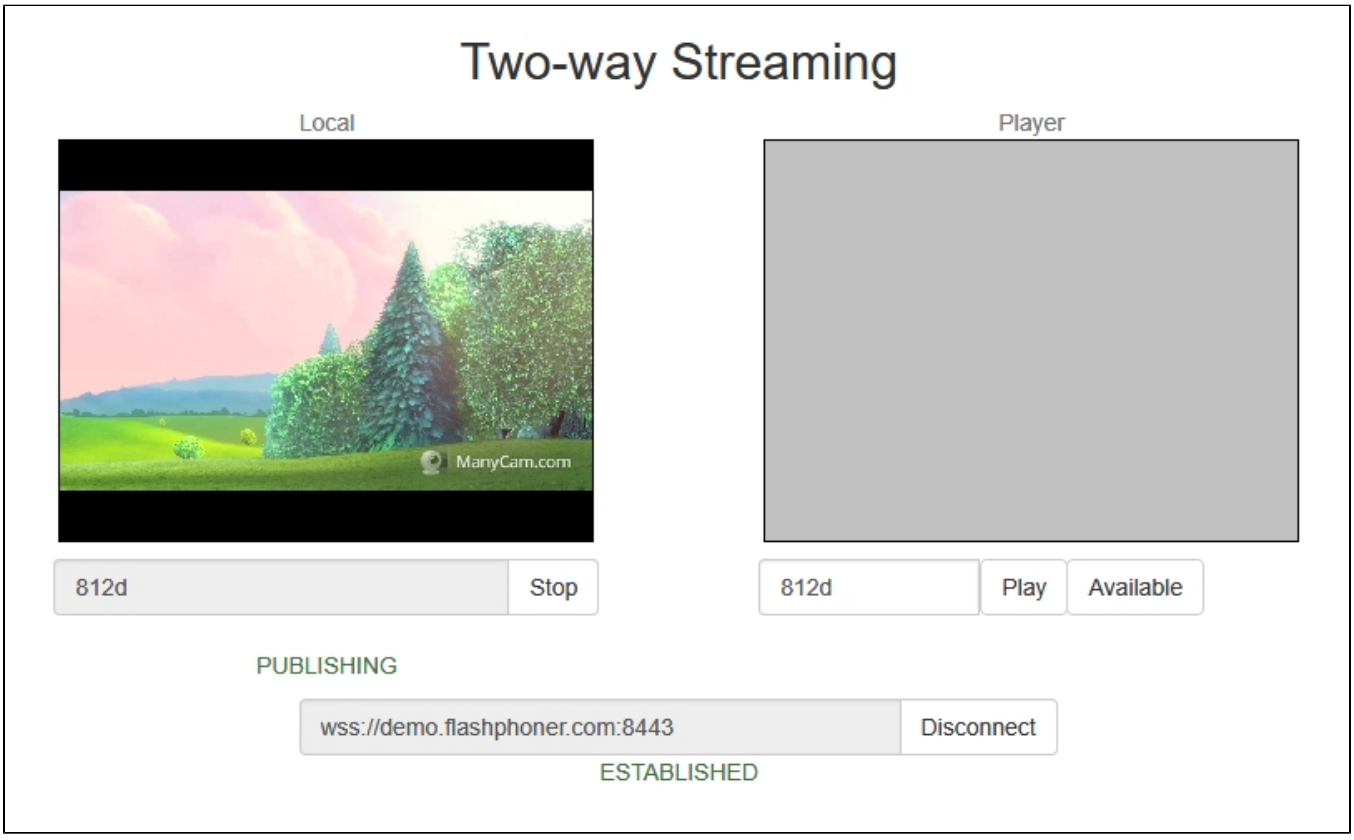
Playing two or more streams on the same page

WCS provides possibility to play two or more streams on the same page. In the context offlowchartandcall flowplaying multiple streams is no different from playing just one.

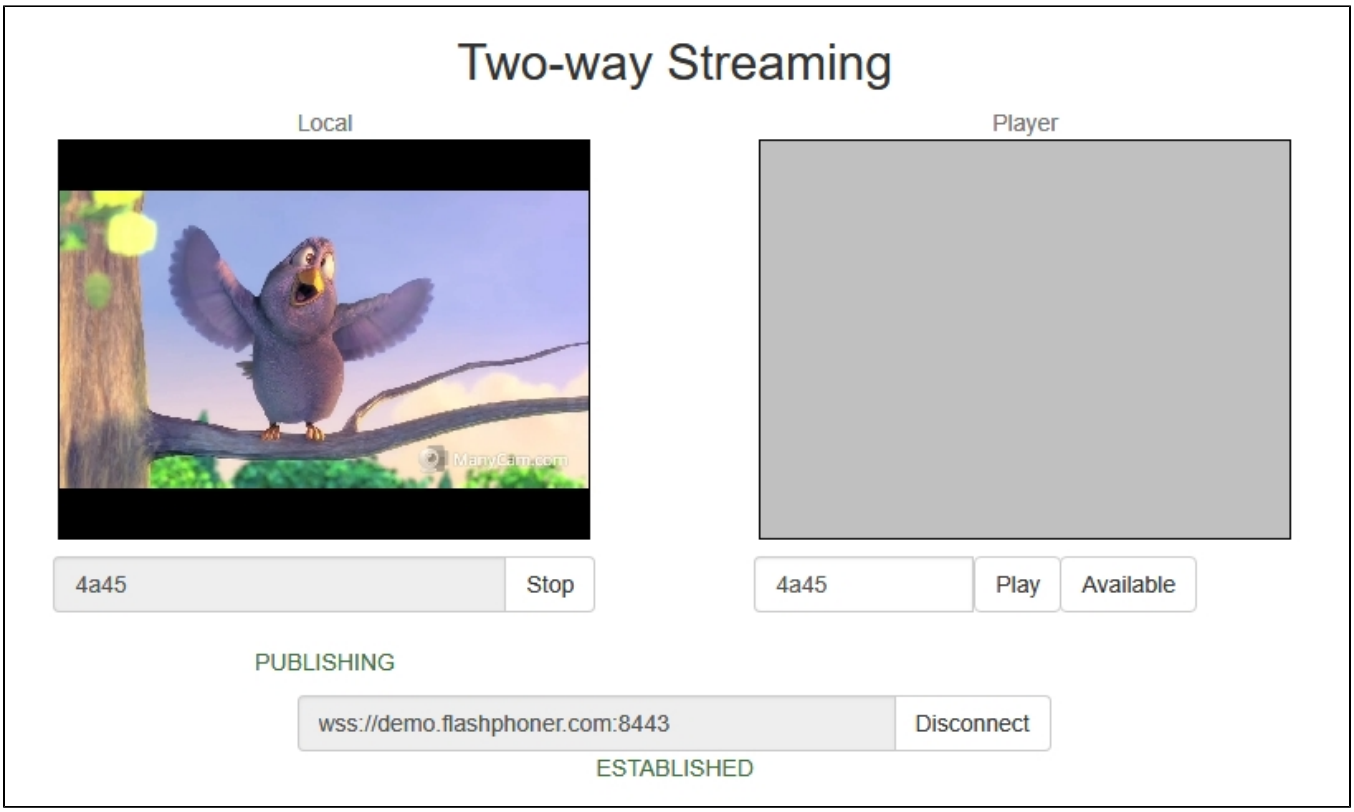
1. For the test we use:

- the demo server at demo.flashphoner.com;
- theTwo Way Streamingweb application to publish streams
- the2 Playersweb application to play streams

2. Open the Two Way Streaming web application, click Connect, then Publish. Copy the identifier of the first stream from the Play window:



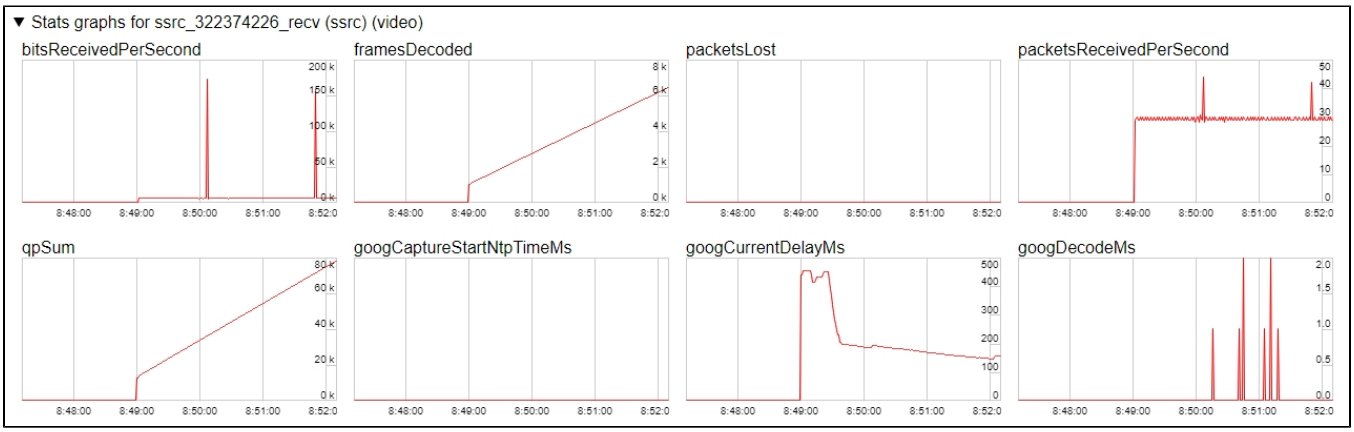
3. In another tab, open the Two Way Streaming web application, click Connect, then Publish. Copy the identifier of the second stream from the Play window:



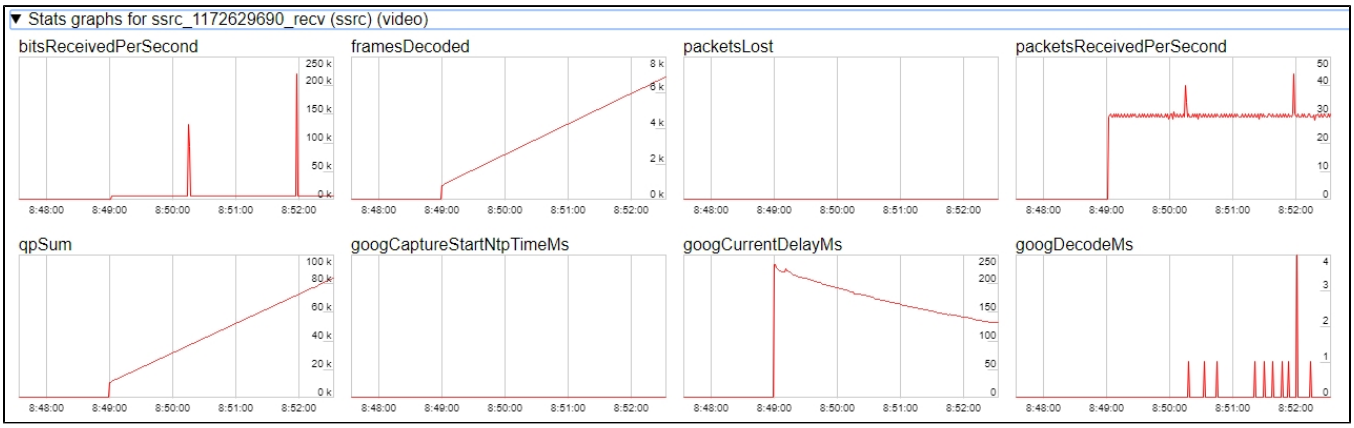
4. Open the 2 Players web application and specify identifiers of the first (left) and the second (right) streams:

5. Click Play below right and left players:

6. Diagrams in chrome://webrtc-internals for the first stream:



7. Diagrams in chrome://webrtc-internals for the second stream:



WebRTC stream playback in custom player

A stream published on WCS server can be played via WebRTC in custom player, for example, in a VR player. To fo this, video page element to play stream should be passed as`remoteVideo`parameter tosession.createStream() WebSDK function

session.ceateStream() code

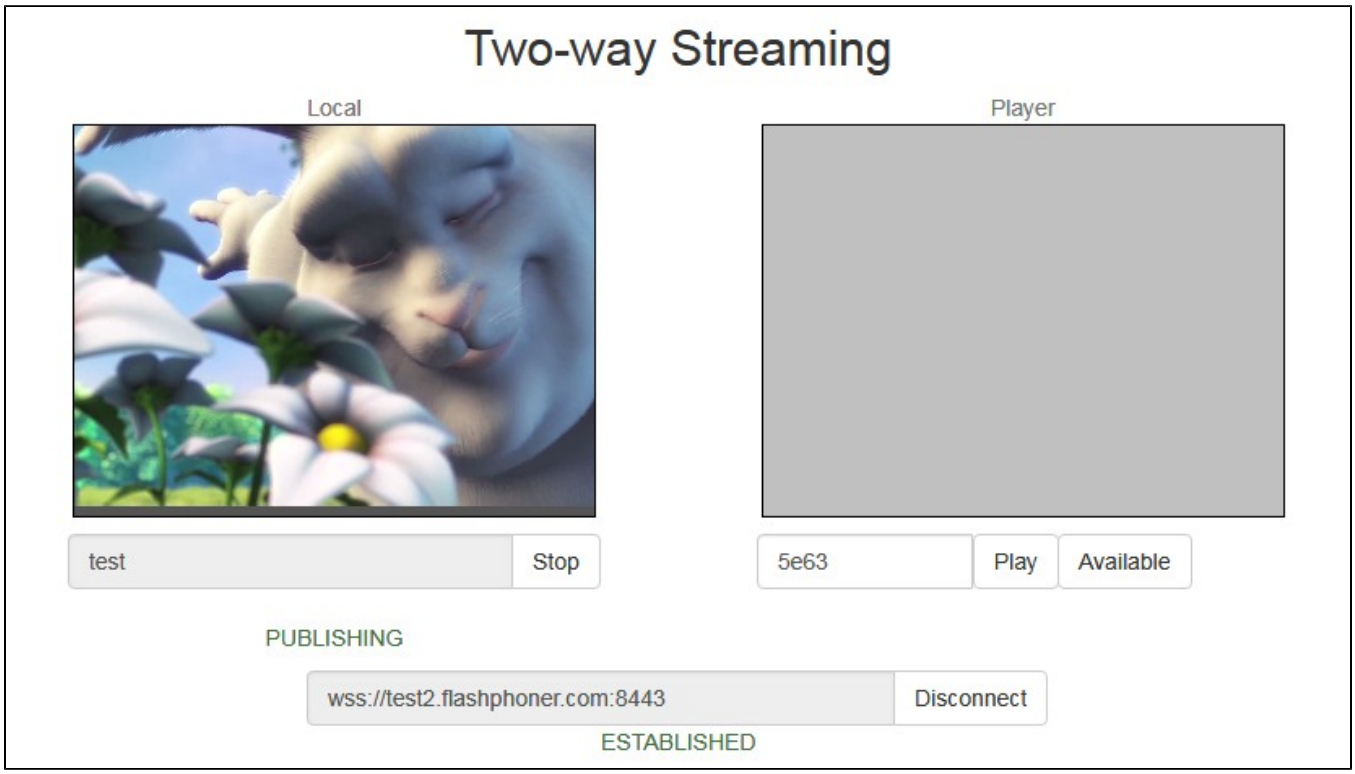
```
                        session.createStream({
                                name: document.getElementById('playStream').value,
                                display: display,
                                remoteVideo: video
                        })
                        ...
```

Testing

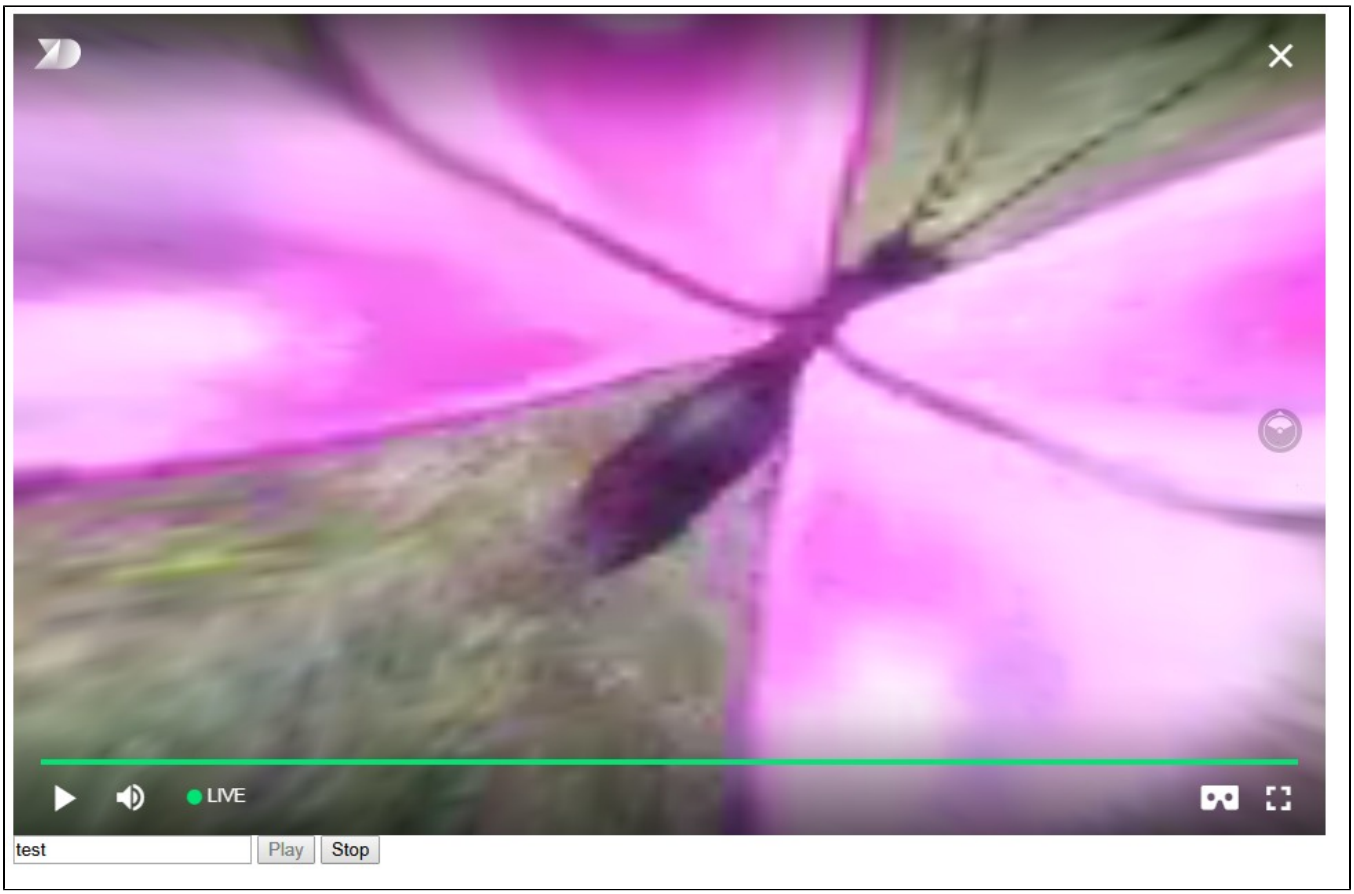
1. For test we use:

- WCS server
- Two Way Streamingweb application to publish a stream
- DelightVR player to play a stream

2. Publish stream on WCS server



3. Play stream in VR player



Custom player page code sample

1. Video page element, stream name input field and buttons to start and stop playback declaration

```
<div style="width: 50%;" id="display">
    <dl8-live-video id="remoteVideo" format="STEREO_TERPON">
        <source>
    </dl8-live-video>
</div>
        <input class="form-control" type="text" id="playStream" placeholder="Stream Name">
        <button id="playBtn" type="button" class="btn btn-default" disabled>Play</button>
        <button id="stopBtn" type="button" class="btn btn-default" disabled>Stop</button>
```

2. Player ready to playback event handling

```
    document.addEventListener('x-dl8-evt-ready', function () {
                        dl8video = document.getElementById('remoteVideo');
                        $('#playBtn').prop('disabled', false).click(function() {
                                playStream();
                        });
    });
```

3. Connection to WCS server establishing 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: document.getElementById('playStream').value,
                                display: display,
                                remoteVideo: video
                        }).on(STREAM_STATUS.PLAYING, function (stream) {
                ...
                                }).play();
                                })
```

4. Start playback in VR player and stop button click handling

```
                        ...
                        }).on(STREAM_STATUS.PLAYING, function (stream) {
                                dl8video.start();
                                $('#stopBtn').prop('disabled', false).click(function() {
                                        $('#playBtn').prop('disabled', false);
                                        $('#stopBtn').prop('disabled', true);
                                        stream.stop();
                                        dl8video.exit();
                                });
                        }).play();
                        })
```

Full custom player page code sample

Code

```
<!DOCTYPE html>
<html>
    <head>
        <title>WebRTC Delight</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
                  <script type="text/javascript" src="../../../../flashphoner.js"></script>
                  <script type="text/javascript" src="../../dependencies/jquery/jquery-1.12.0.js"></script>
                  <script type="text/javascript" src="../../dependencies/js/utils.js"></script>
                  <script src="dl8-66b250447635476d123a44a391c80b09887e831e.js" async></script>
        <meta name="dl8-custom-format" content='{"name": "STEREO_TERPON","base":"STEREO_MESH","params":{"uri":
"03198702.json"}}'>
    </head>
    <body>
        <div style="width: 50%;" id="display">
            <dl8-live-video id="remoteVideo" format="STEREO_TERPON">
                <source>
            </dl8-live-video>
        </div>
                <input class="form-control" type="text" id="playStream" placeholder="Stream Name">
                <button id="playBtn" type="button" class="btn btn-default" disabled>Play</button>
                <button id="stopBtn" type="button" class="btn btn-default" disabled>Stop</button>
        <script>
                        Flashphoner.init({flashMediaProviderSwfLocation: '../../../../media-provider.swf'});
                        var SESSION_STATUS = Flashphoner.constants.SESSION_STATUS;
                        var STREAM_STATUS = Flashphoner.constants.STREAM_STATUS;
                        var STREAM_STATUS_INFO = Flashphoner.constants.STREAM_STATUS_INFO;
                        var playBtn = document.getElementById('playBtn');
                        var display = document.getElementById('display');
                        var dl8video = null;
                        var url = setURL();
            document.addEventListener('x-dl8-evt-ready', function () {
                                dl8video = document.getElementById('remoteVideo');
                                $('#playBtn').prop('disabled', false).click(function() {
                                        playStream();
                                });
            });
                        function playStream() {
                        $('#playBtn').prop('disabled', true);
                        $('#stopBtn').prop('disabled', false);
                        var video = dl8video.contentElement;
                        Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function
(session) {
                        var session = Flashphoner.getSessions()[0];
                        session.createStream({
                                name: document.getElementById('playStream').value,
                                display: display,
                                remoteVideo: video
                        }).on(STREAM_STATUS.PLAYING, function (stream) {
                                dl8video.start();
                                $('#stopBtn').prop('disabled', false).click(function() {
                                        $('#playBtn').prop('disabled', false);
                                        $('#stopBtn').prop('disabled', true);
                                        stream.stop();
                                        dl8video.exit();
                                });
                        }).play();
                        })
                        }
        </script>
    </body>
</html>
```

Automatic stream playback

PlayerandEmbed Playerexamples support automatic stream playback with the following URL parameter

```
autoplay=true
```

for example

```
https://hostname:8888/embed_player?urlServer=wss://hostname:
8443&streamName=stream1&autoplay=true&mediaProviders=WebRTC
```

Where

- hostname is WCS server hostname
- stream1 is a stream name

Autoplay issues in different browsers

Chrome

In latest Chrome versions (71 and higher) content autoplay policy was changed. Now, user has to do something to start video playback on web page, to press a key for example.

The policy change affects also audiocontext creation that is needed to play a sound. Accrding to new policy, audiocontext may only be created as response to some user action.

Therefore, in Chrome 71 and in another Chromium based browsers that support new autoplay policy, video automatic playback starts with no sound. To enable sound user has to move volume control in Embed Player window.

Ther is a way to workaround this limit with no users action. To do this, this code should be added to player page

```
<iframe src="silence.mp3" allow="autoplay" id="audio" style="display:none"></iframe>
```

and silence.mp3 file containing 0.25 seconds of silence should be placed in page folder.

In this case, the silence will be played on page load, then audiocontext can be created and video with sound can be played.

iOS Safari

Autoplay works since iOS 12.2. Note that autoplay policy as well as in Chrome browser, requires user to move volume cotrol to start sound playback.

In iOS 12.2-12.3 sound playback may not be started even after moving volume control. In this case, video playback should be restarted without reloading the page.

Autoplay does not work in iOS Safari when Low Power Mode is enabled.

Audio playback tuning in iOS Safari

If one video stream is playing and then another video stream is publishing on the same page (videochat case for example) in iOS Safari, the sound level may change for stream played. This can be escaped by the following ways:

1. Query media devices access on session creation before playing a stream

```
    Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function (session) {
        ...
        if (Browser.isSafariWebRTC() && Browser.isiOS() && Flashphoner.getMediaProviders()[0] === "WebRTC") {
            Flashphoner.playFirstVideo(localVideo, true, PRELOADER_URL).then(function () {
                Flashphoner.getMediaAccess(null, localVideo).then(function (disp) {
                });
            });
        }
        ...
    });
```

2. 1-1,5 seconds after PLAYING stream status receiving, mute and unmute video and/or sound

```
    session.createStream({
        name: streamName,
        display: remoteVideo
    }).on(STREAM_STATUS.PENDING, function (stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function (stream) {
        setStatus("#playStatus", stream.status());
        onPlaying(stream);
        if (Browser.isSafariWebRTC() && Browser.isiOS() && Flashphoner.getMediaProviders()[0] === "WebRTC") {
            setTimeout(function () {
                muteVideo();
                unmuteVideo();
            }, 1500);
        }
        ...
    }).play();
```

Known issues

1. Possible bug in the Safari browser on iOS leads to freezes while playing via WebRTC

Symptoms: video playback stops, while the audio track may continue playing. Recovery needs reloading the page or restarting the browser.

Solution:

a) enable the transcoder on the server by setting the following parameter in flashphoner.properties

```
disable_streaming_proxy=true
```

b) when playing the stream from iOS Safari explicitly specify width and height, for example:

```
session.createStream({constraints:{audio:true,video:{width:320,height:240}}}).play();
```

2. Audiocodec PMCU ia used instead of Opus when sttream is published via RTMP and is played via WebRTC

Symptoms: PMCU codec is shown in chrome://webrtc-internals

Solution: switch Avoid Transcoding Alhorithm off using the following parameter inflashphoner.properties

```
disable_rtc_avoid_transcoding_alg=true
```

3. When RTMP stream is published with Flash Streaming, then it is played in iOS Safari browser via WebRTC, and another stream is published form iOS Safari via WebRTC, sound stops playing in RTMP stream.

Symptoms:

a) The stream1 stream is published from Flash Streaming web application in Chrome browser on Windows
b) The stream1 stream is played in Two Way Streaming web application in iOS Safari browser. Sound and video play normally.
c) The stream2 stream is published from Two Way Streaming web application in iOS Safari browser.Sound stops playing.
d) Stop publishing stream in iOS Safari. Sound of stream1 plays again.

Solution: switch Avoid Transcoding Alhorithm off on the server using the following parameter inflashphoner.propertiesfile

```
disable_rtc_avoid_transcoding_alg=true
```

4. While publishing RTMP stream with Keep Alive disabled for all protocols, this stream playback via WebRTC in browser stops when WebSocket timeout expires

Symptoms: playback of stream published with RTMP encoder stops in browser with no error message

Solution: if Keep Alive is disabled for all protocols with the following parameterinflashphoner.propertiesfile

```
keep_alive.algorithm=NONE
```

it is necessary to switch off WebSocket read timeout with the following parameter

```
ws_read_socket_timeout=false
```

5. G722 codec does not work in Edge browser

Symptoms: WebRTC stream with G722 audio does not play in Edge browser or play without sound and with freezes

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, for example Opera, Yandex, 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 clent side

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

Note thatstream transcodingon server is enabled when stream published as H264 is played as VP8 and vice versa.

7. If Flash is enabled in site settings, an error can occur in Chrome 71 and later browser console "Cross-origin content must have visible size large than 400 x 300 pixels, or it will be blocked" while playing WebRTC stream.

Symptoms: "Cross-origin content must have visible size large than 400 x 300 pixels, or it will be blocked" message in browser console while playing WebRTC stream, playback works normally

Solution: use WebSDK without Flash support

```
flashphoner-no-flash.js
```