In a browser via MSE

Overview

Media Source Extensions (MSE) is a browser API that allows playing audio and video using the corresponding HTML5 tags `audio` and `video`. While WebRTC is intended for both playing and publishing streams in real time, MSE is for playing only. Therefore, the MSE technology can be used when you only need to play a stream on the page and there are no strict requirements to latency.

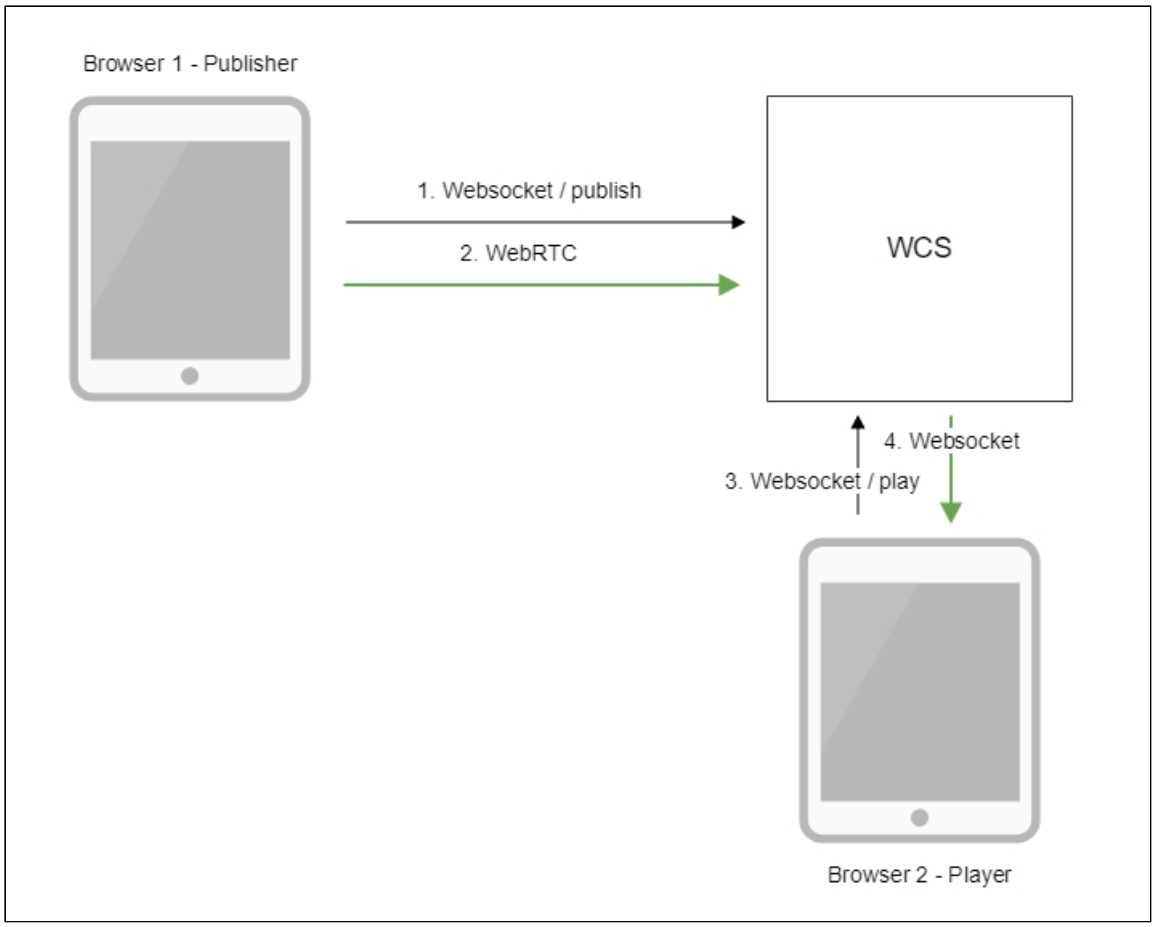
Supported platforms and browsers

	Chrome	**Firefox**	**Safari**	**Edge**
Windows	✅	✅	❌	✅
Mac OS	✅	✅	✅	✅
Android	✅	✅	❌	✅
iOS	❌	❌	❌	❌

Supported codecs

- Video: H.264
- Audio: AAC

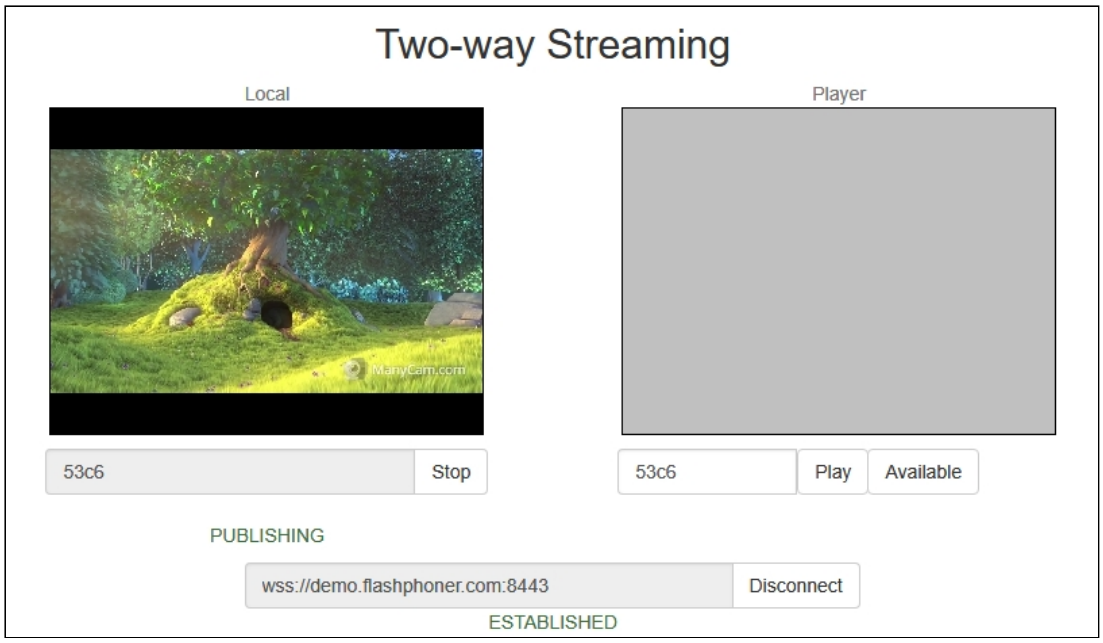
Operation flowchart



1. The browser connects to the server via the Websocket protocol and sends the `publishStream` 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 Websockets and sends the `playStream` command.
4. The second browser receives the H.264 + AAC stream via Websocket and plays this stream on the page using MSE.

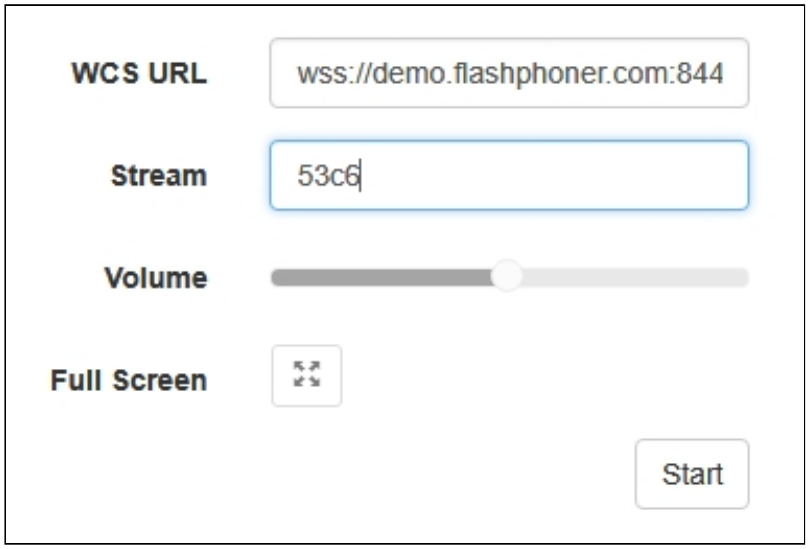
Quick manual on testing

1. For this test we use:
2. the demo server at `demo.flashphoner.com`;
3. the Two Way Streaming web application for publishing the stream
4. the Player web application to play the stream via MSE
5. Open the Two Way Streaming application. Click `Connect`, then `Publish`. Copy the identifier of the stream:

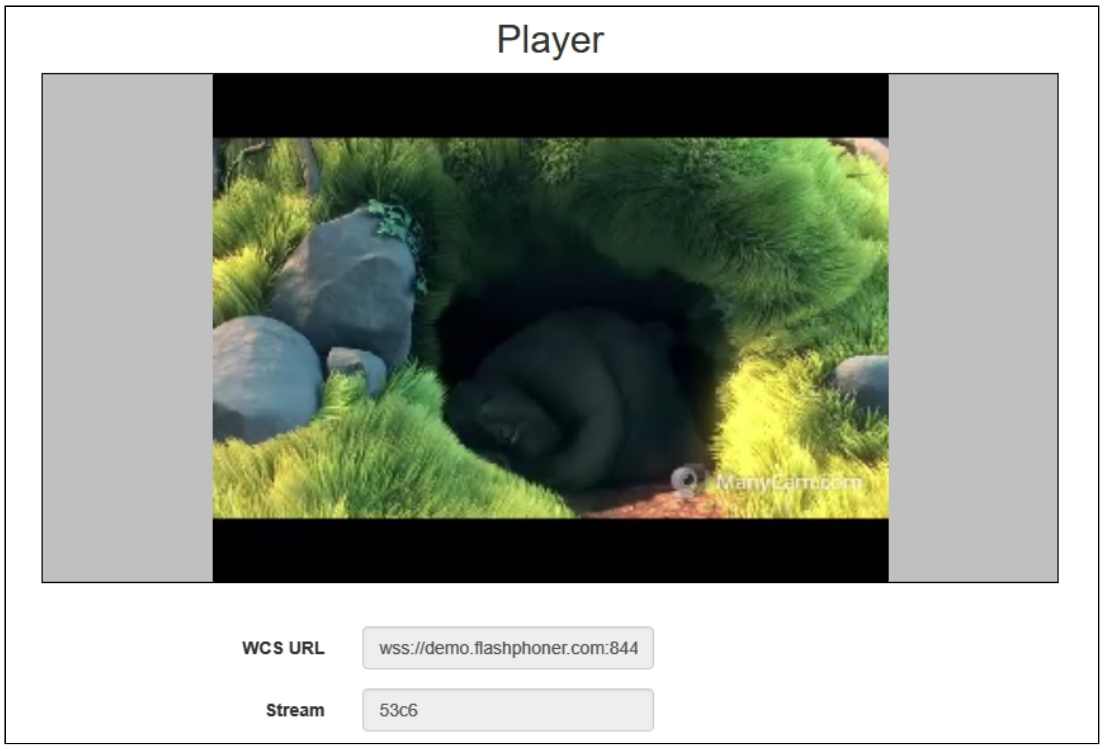


6. Open the Player web application and specify MSE in the parameters of the URL `https://demo.flashphoner.com/client2/examples/demo/streaming/player/player.html?mediaProvider=MSE`

7. Set the identifier of the stream in the `Stream` field:



8. Click the `Start` button. The stream starts playing:

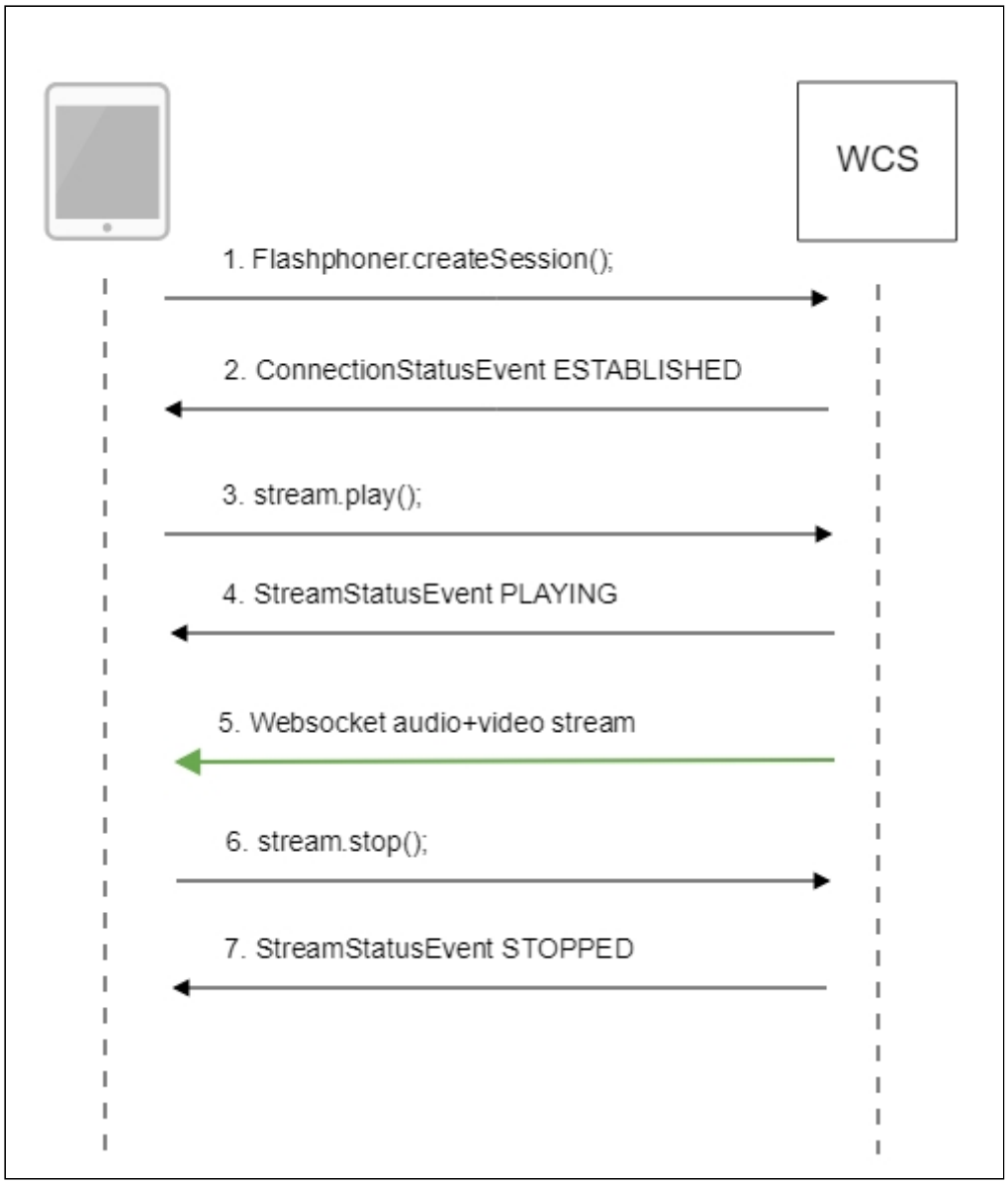


Call flow

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

player.html

player.js



1. Establishing a connection to the server

   `Flashphoner.createSession()` code

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function(session){
  setStatus(session.status());
  //session connected, start playback
  playStream(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
  setStatus(SESSION_STATUS.DISCONNECTED);
  onStopped();
}).on(SESSION_STATUS.FAILED, function(){
  setStatus(SESSION_STATUS.FAILED);
  onStopped();
});
```

2. Receiving from the server an event confirming successful connection `SESSION_STATUS.ESTABLISHED` code

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED,
function(session){
  setStatus(session.status());
  //session connected, start playback
  playStream(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
  ...
}).on(SESSION_STATUS.FAILED, function(){
  ...
});
```

3. Playing the stream `Stream.play()` code

```
if (Flashphoner.getMediaProviders()[0] === "MSE" && mseCutByIFrameOnly) {
  options.mediaConnectionConstraints = {
      cutByIFrameOnly: mseCutByIFrameOnly
  }
}
...
stream = session.createStream(options).on(STREAM_STATUS.PENDING,
function(stream) {
  ...
});
stream.play();
```

4. Receiving from the server an event confirming successful playing of the stream `STREAM_STATUS.PLAYING` code

```
stream = session.createStream(options).on(STREAM_STATUS.PENDING,
function(stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function(stream) {
  $("#preloader").show();
  setStatus(stream.status());
  onStarted(stream);
}).on(STREAM_STATUS.STOPPED, function() {
  ...
}).on(STREAM_STATUS.FAILED, function(stream) {
  ...
}).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
  ...
});
stream.play();
```

5. Receiving the audio-video stream via Websocket and playing via MSE

6. Stopping the playback of the stream `Stream.stop()` code

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

7. Receiving from the server and event confirming the playback of the stream is stopped `STREAM_STATUS.STOPPED` code

```
stream = session.createStream(options).on(STREAM_STATUS.PENDING,
function(stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function(stream) {
  ...
}).on(STREAM_STATUS.STOPPED, function() {
  setStatus(STREAM_STATUS.STOPPED);
  onStopped();
}).on(STREAM_STATUS.FAILED, function(stream) {
  ...
}).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
  ...
});
stream.play();
```

MSE buffering

With a large number of subscribers playing MSE streams, server CPU load average increases. To prevent this, MSE frames buffering added since build 5.2.360. A frames count to be sent in one packet is defined with the following parameter in flashphoner.properties file

```
avcc_buffer_wait_frames_count=5
```

By default, 5 frames are sent in one packet.

A buffer size for the frames to be sent can be set with the following parameter

```
avcc_send_buffer_size=500000
```

By default, buffer size is 500 kbytes. If packet does not fit to the buffer, server will try to send it directly to the subscriber with the following message in server log

```
12:00:50,555 ERROR       AvccSendBuffer - VideoProcessor-db2da9a0-ddb6-11e9-
9fc2-cf9284f3bdd0 Failed to buffer frame
```

It is recommended to set more frames count per one packet and more buffer size to decrease CPU load average.

> **Attention**
>
> More buffering adds more playback delay

Buffering can be disabled if necessary by changing the parameter `msePacketizationVersion` in WebSDK source code

```
wsConnection.onopen = function () {
   onSessionStatusChange(SESSION_STATUS.CONNECTED);
   cConfig = {
       appKey: appKey,
       mediaProviders: Object.keys(MediaProvider),
       keepAlive: keepAlive,
       authToken:authToken,
       clientVersion: "0.5.28",
       clientOSVersion: window.navigator.appVersion,
       clientBrowserVersion: window.navigator.userAgent,
       msePacketizationVersion: 2,
       custom: options.custom
   };
   ...
}
```

to

```
   cConfig = {
       ...
       msePacketizationVersion: 1,
       ...
   }
```

In this case buffering settings are not applied, frames will be sent directly to MSE subscribers.

Known issues

1. Video freezes are possible when playing a low FPS RTMP stream with `mseCutByIFrameOnly=true` and transcoding

When RTMP stream is published with low framerate and played via MSE in MS Edge and Internet Explorer 11 browsers with `mseCutByIFrameOnly=true` setting and transcoding enabled, video freezes are possible.

Symptoms

When stream is published from Flash client and played in Player web application with `mseCutByIFrameOnly=true` setting enabled and resolution explicitly set, for example `https://server:8888/client2/examples/demo/streaming/player/player.html?resolution=320x240&mediaProvider=MSE&mseCutByIFrameOnly=true`, freezes often occur in MS Edge or IE 11 browsers.

Solution

a) FPS must not be lower then 25 when stream is published from Flash client, transcoding has also to be escaped;

b) If FPS cannot be higher or transcoding is necessary, the following parameter in flashphoner.properties file should be reduced, for example

```
video_encoder_h264_gop=30
```

2. MSE is not supported in iOS Safari on iPhone devices

Symptoms

Stream playback by MSE on iPhone device with iOS 12 and later is not started, in this case `None of preferred media providers available` message is displayed in Embed Player example

Solution

Use WebRTC or HLS (in native player) on iPhone device with iOS 12 and later

3. Two streams cannot be played simultaneously by MSE using the same Websocket connection on the same page

Symptoms

Two streams cannot be played in 2Players example using main browsers (Chrome, Firefox, Safari) while connecting to WCS server via HTTP

Solution

Use a separate Websocket connection for each stream on the same page while playing them by MSE