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.

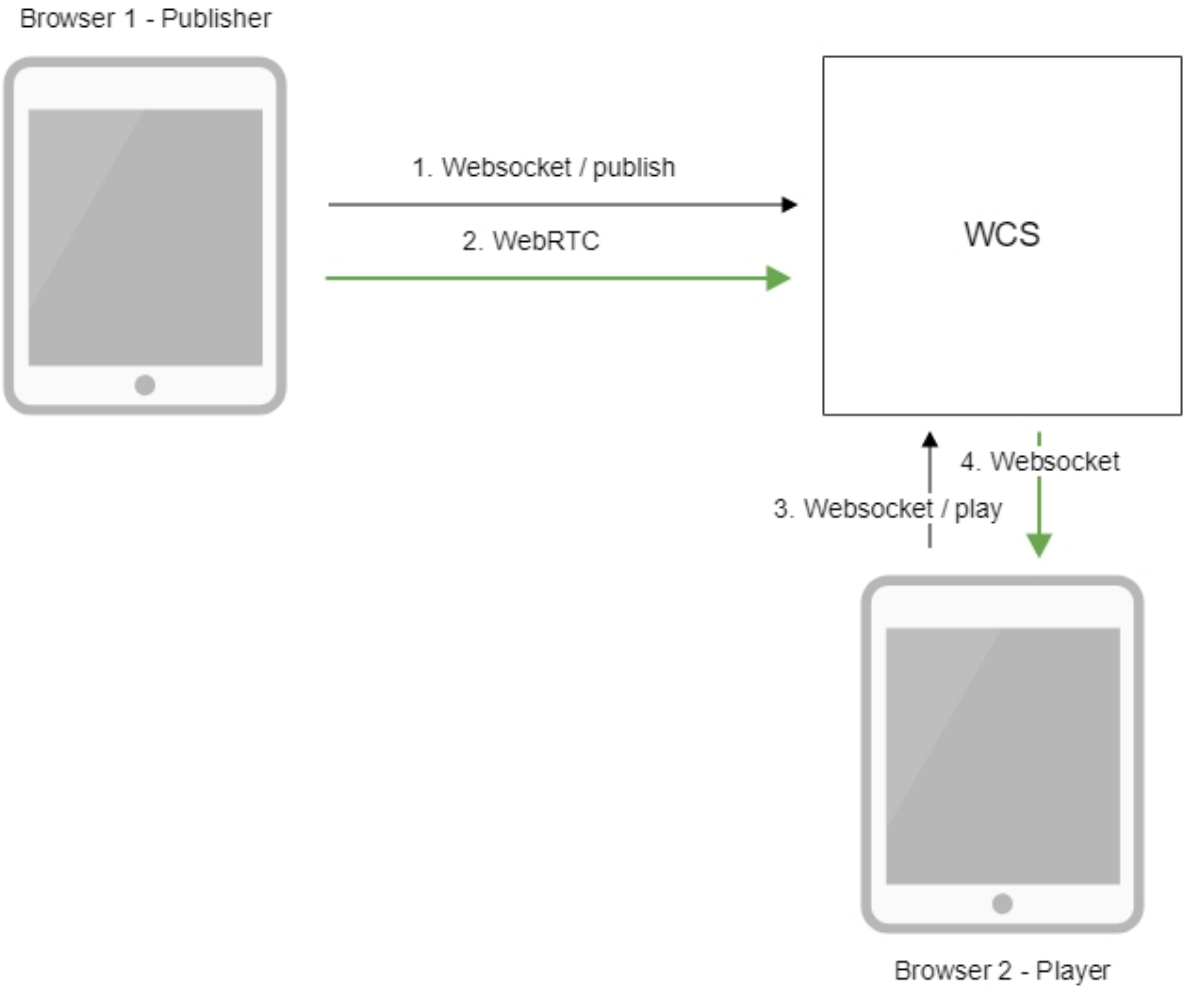
Supported platforms and browsers

	Chrome	Firefox	Safari 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 also via Websockets and sends the play command.
4. The second browser receives the H.264 + AAC stream via Websocket and plays this stream on the page using MSE.

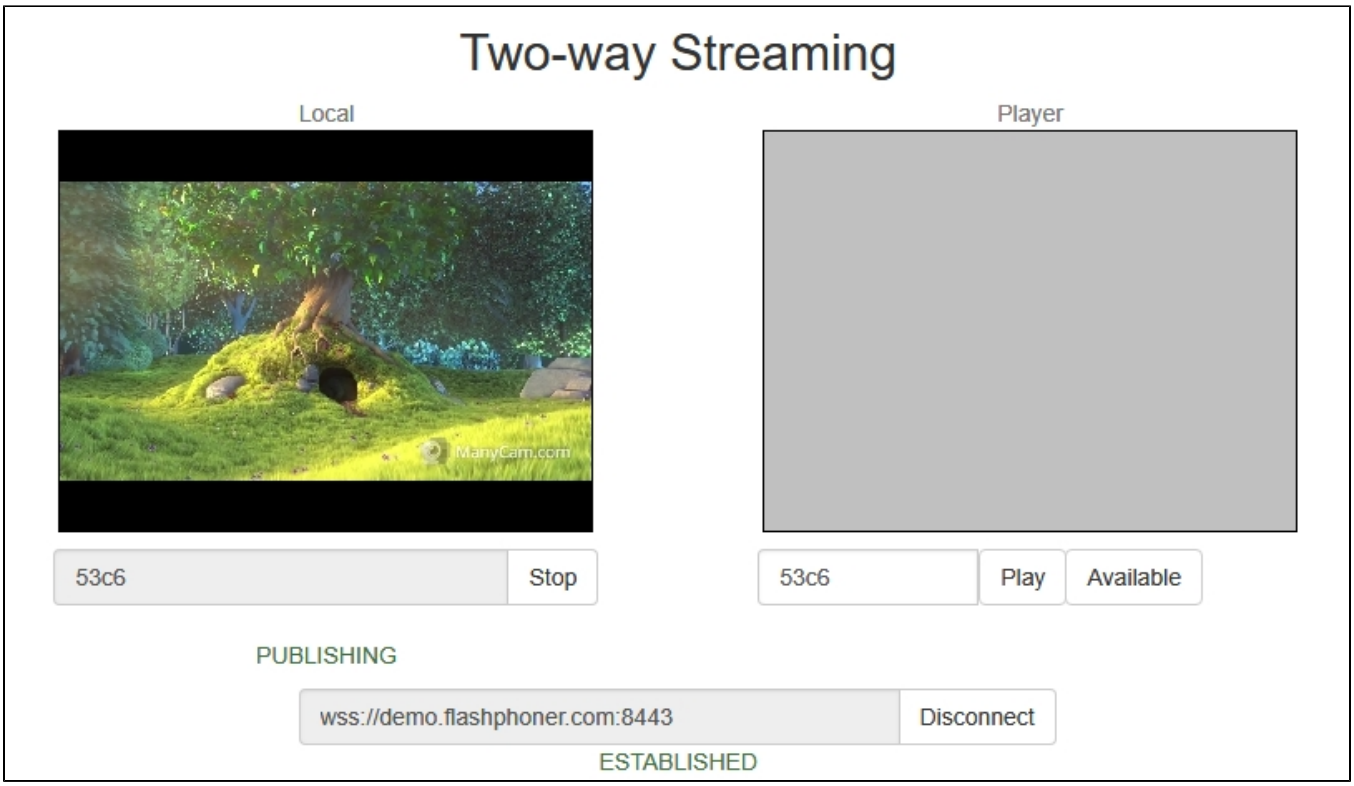
Quick manual on testing

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

1. For this test we use:

- the demo server at demo.flashphoner.com;
- theTwo Way Streamingweb application for publishing the stream
- thePlayerweb application to play the stream via MSE

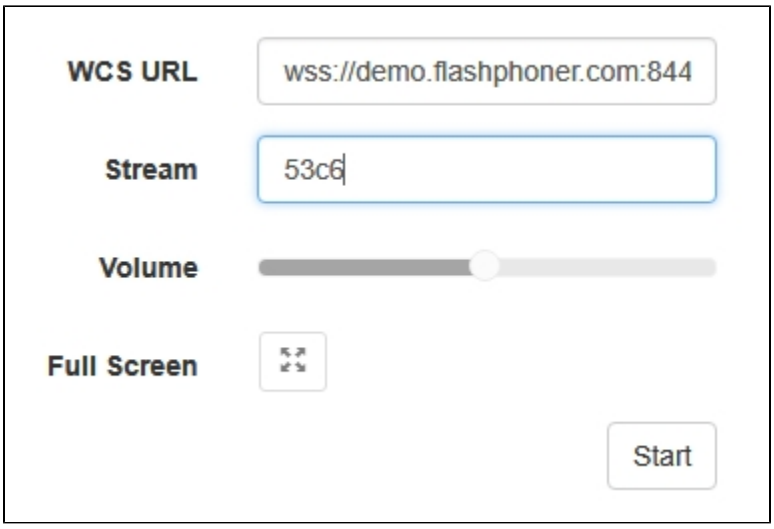
2. Open the Two Way Streaming application. Click Connect, then Publish. Copy the identifier of the stream:



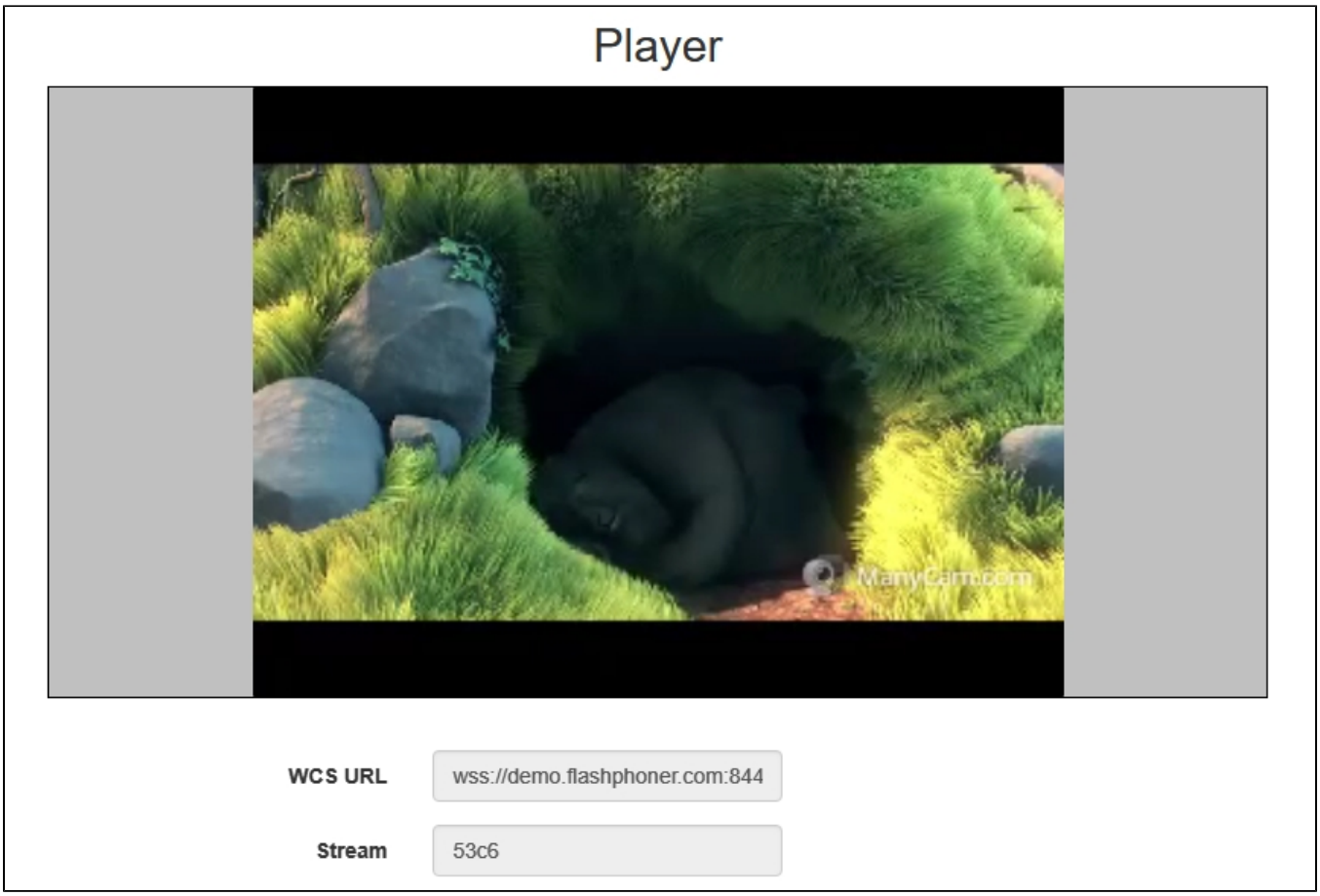
3. Open the Player web application and specify MSE in the parameters of the URL

https://demo.flashphoner.com/client2/examples/demo/streaming/player/player.html?mediaProvider=MSE

4. Set the identifier of the stream in the Stream field:



5. Click the Start button. The stream starts playing:

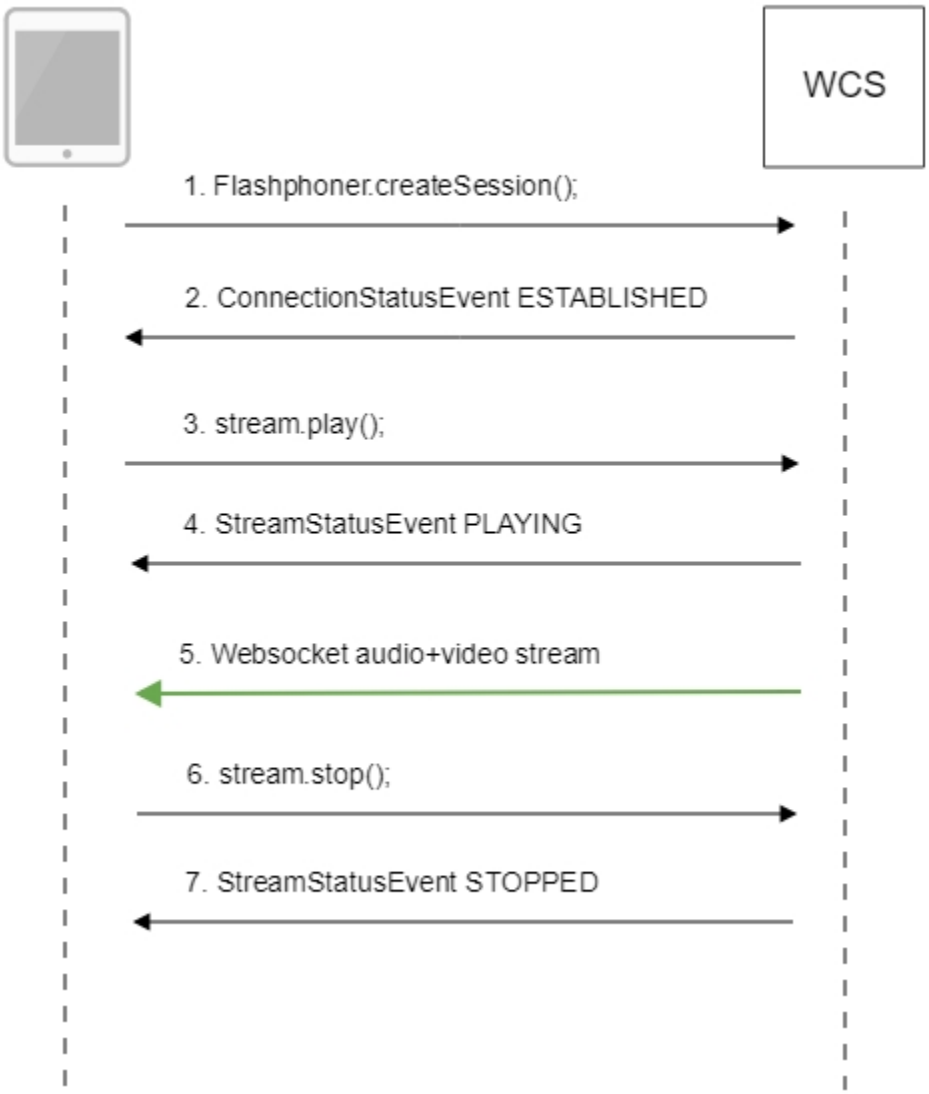


Call flow

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.

ConnectionStatusEvent ESTABLISHEDcode

```
    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.

StreamStatusEvent, status PLAYINGcode

```
    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.

StreamStatusEvent, status STOPPEDcode

```
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();
```

Known issues

1. When stream is published from Flash client 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 appliucation with mseCutByIFrameOnly=true setting enabled and resolution explicitly set, for examplehttps://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:

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

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

```
video_encoder_h264_gop=30
```