

Managing camera and microphone

- [Microphone settings](#)
- [Camera settings](#)
- [Testing camera and microphone capturing locally](#)
- [SDP parameters replacing](#)
 - [Rising up the bitrate of video stream published in Chrome browser](#)
 - [Bandwidth management](#)
- [Setting up codecs](#)
- [Sound device selection](#)
- [WebRTC statistics displaying](#)
- [Picture parameters management](#)
 - [Picture resolution management](#)
 - [Frame rate management](#)
- [Seamless switching between web camera stream and screen share stream during publication](#)
 - [Known limitations](#)
- [Known issues](#)

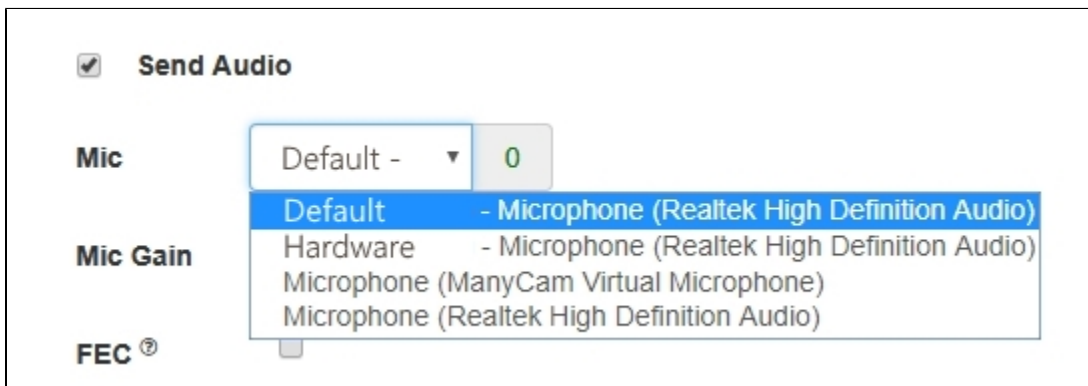
WCS allows configuring the camera and the microphone from a browser. Let's see how this can be done and what parameters you can adjust when an audio and video stream is captured. We use the Media Devices web application as an example:

[media_device_manager.html](#)

[manager.js](#)

Microphone settings

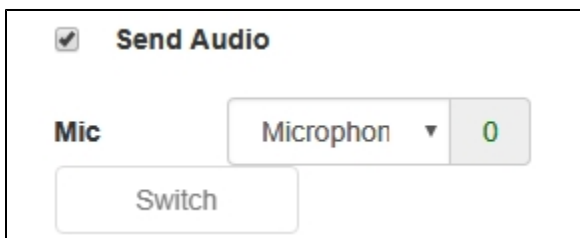
1. Selecting the microphone from the list

A screenshot of the 'Media Devices' web application interface. At the top, there is a checkbox labeled 'Send Audio' which is checked. Below this, there are three main sections: 'Mic', 'Mic Gain', and 'FEC'. The 'Mic' section has a dropdown menu currently showing 'Default -' with a small green '0' next to it. A dropdown menu is open below the 'Mic' section, listing four options: 'Default - Microphone (Realtek High Definition Audio)' (highlighted in blue), 'Hardware - Microphone (Realtek High Definition Audio)', 'Microphone (ManyCam Virtual Microphone)', and 'Microphone (Realtek High Definition Audio)'. The 'Mic Gain' section has a slider. The 'FEC' section has a checkbox.

code:

```
Flashphoner.getMediaDevices(null, true, MEDIA_DEVICE_KIND.INPUT).then(function (list) {
    list.audio.forEach(function (device) {
        ...
    });
    ...
}).catch(function (error) {
    $("#notifyFlash").text("Failed to get media devices");
});
```

2. Microphone switching while stream is publishing

A screenshot of the 'Media Devices' web application interface. At the top, there is a checkbox labeled 'Send Audio' which is checked. Below this, there are three main sections: 'Mic', 'Mic Gain', and 'FEC'. The 'Mic' section has a dropdown menu currently showing 'Microphon' with a small green '0' next to it. Below the 'Mic' section, there is a button labeled 'Switch'.

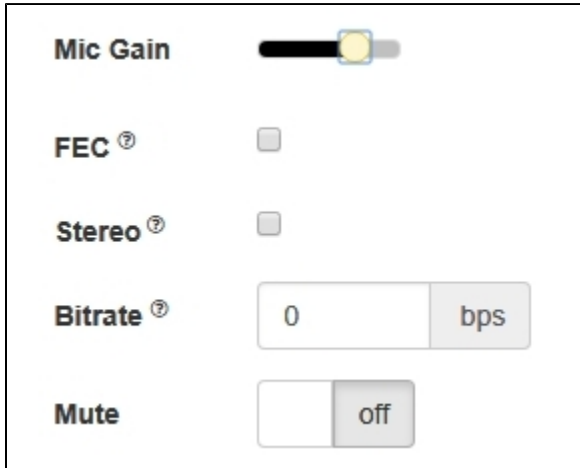
code:

```

$("#switchMicBtn").click(function (){
    stream.switchMic().then(function(id) {
        $('#audioInput option:selected').prop('selected', false);
        $("#audioInput option[value='"+ id +"']").prop('selected', true);
    }).catch(function(e) {
        console.log("Error " + e);
    });
}).prop('disabled', !($('#sendAudio').is(':checked')));

```

3. Adjusting microphone gain (works in Chrome only)



The screenshot shows a control panel with the following elements:

- Mic Gain:** A horizontal slider with a yellow knob.
- FEC:** A checkbox that is currently unchecked.
- Stereo:** A checkbox that is currently unchecked.
- Bitrate:** A text input field containing the number '0', followed by a button labeled 'bps'.
- Mute:** A toggle switch with the label 'off'.

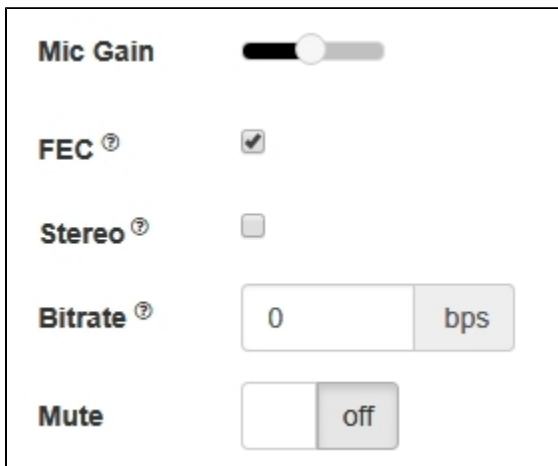
code:

```

$("#micGainControl").slider({
    range: "min",
    min: 0,
    max: 100,
    value: currentGainValue,
    step: 10,
    animate: true,
    slide: function (event, ui) {
        currentGainValue = ui.value;
        if(previewStream) {
            publishStream.setMicrophoneGain(currentGainValue);
        }
    }
});

```

4. Enabling error correction (for the Opus codec only)



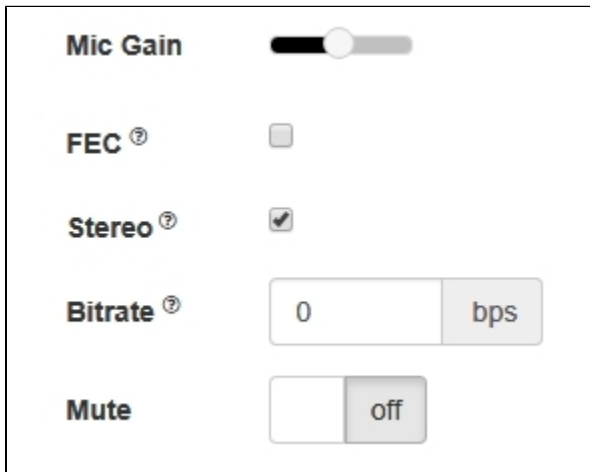
The screenshot shows a control panel similar to the previous one, but with the following differences:

- Mic Gain:** A horizontal slider with a white knob.
- FEC:** A checkbox that is now checked.
- Stereo:** A checkbox that is currently unchecked.
- Bitrate:** A text input field containing the number '0', followed by a button labeled 'bps'.
- Mute:** A toggle switch with the label 'off'.

code:

```
if (constraints.audio) {
  constraints.audio = {
    deviceId: $('#audioInput').val()
  };
  if ($("#fec").is(':checked'))
    constraints.audio.fec = ($("#fec").is(':checked'));
  ...
}
```

5. Setting stereo/mono mode.



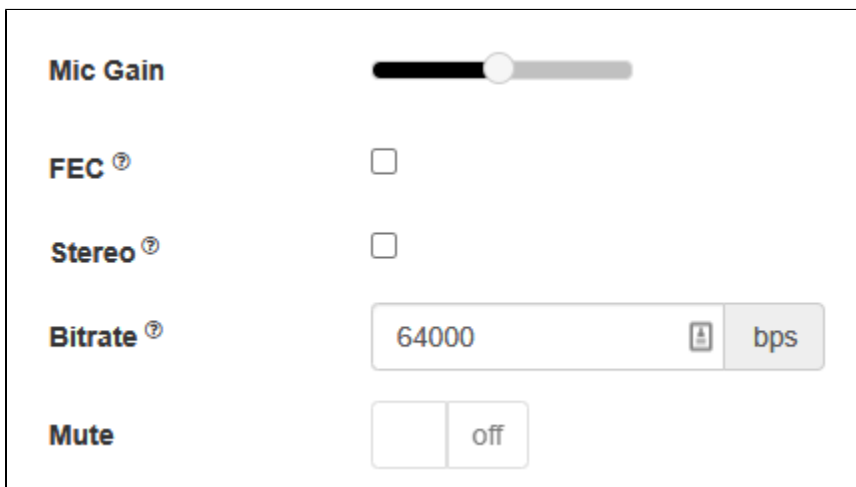
The screenshot shows a settings panel with five controls:

- Mic Gain**: A horizontal slider with a white knob.
- FEC**: A checkbox with a question mark icon, currently unchecked.
- Stereo**: A checkbox with a question mark icon, currently checked.
- Bitrate**: A text input field containing '0' and a unit selector dropdown set to 'bps'.
- Mute**: A toggle switch currently set to 'off'.

code:

```
if (constraints.audio) {
  constraints.audio = {
    deviceId: $('#audioInput').val()
  };
  ...
  if ($("#sendStereoAudio").is(':checked'))
    constraints.audio.stereo = ($("#sendStereoAudio").is(':checked'));
  ...
}
```

6. Setting audio bitrate in bps



The screenshot shows the same settings panel as before, but with the following changes:

- Stereo**: The checkbox is now unchecked.
- Bitrate**: The text input field now contains '64000'.

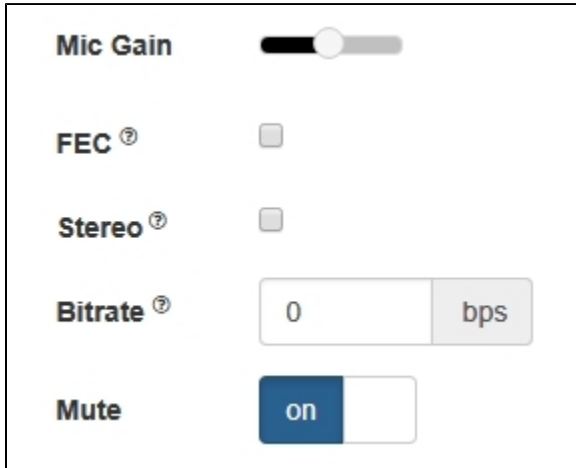
code:

```

if (constraints.audio) {
  constraints.audio = {
    deviceId: $('#audioInput').val()
  };
  ...
  if (parseInt($('#sendAudioBitrate').val()) > 0)
    constraints.audio.bitrate = parseInt($('#sendAudioBitrate').val());
}

```

7. Turning off the microphone (mute).



code:

```

if ($("#muteAudioToggle").is(":checked")) {
  muteAudio();
}

```

Camera settings

1. Camera selection



code:

```

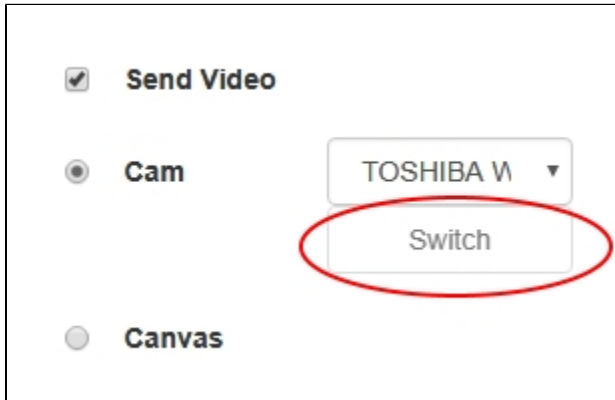
Flashphoner.getMediaDevices(null, true, MEDIA_DEVICE_KIND.INPUT).then(function (list) {
  ...
  list.video.forEach(function (device) {
    ...
  });
}).catch(function (error) {
  $("#notifyFlash").text("Failed to get media devices");
});

```

If audio devices access should not be requested while choosing a camera, `getMediaDevices()` function should be called with explicit constraints setting

```
Flashphoner.getMediaDevices(null, true, MEDIA_DEVICE_KIND.INPUT, {video: true, audio: false}).then(function (list) {  
    ...  
    list.video.forEach(function (device) {  
        ...  
    });  
}).catch(function (error) {  
    $("#notifyFlash").text("Failed to get media devices");  
});
```

2. Switching cameras while stream is publishing



code:

```
$("#switchBtn").text("Switch").off('click').click(function () {  
    stream.switchCam().then(function(id) {  
        $('#videoInput option:selected').prop('selected', false);  
        $("#videoInput option[value='"+ id +"']").prop('selected', true);  
    }).catch(function(e) {  
        console.log("Error " + e);  
    });  
});
```

Switching of the camera can be done "on the fly" during stream broadcasting. Here is how switching works:

- On PC cameras switch in the order they are defined in the device manager of the operating system.
- On Android, if Chrome is used, the default is the frontal camera. If Firefox is used, the default is the rear camera.
- On iOS in the Safari browser, by default the frontal camera is selected, but in the drop-down the rear camera is the first.

3. Specifying the resolution of the video

Size	640	480
FPS	30	
Bitrate	min 0	max 0
COD ?	☒	
Mute	☐ ☒ off	

code:

```
constraints.video = {
    deviceId: $('#videoInput').val(),
    width: parseInt($('#sendWidth').val()),
    height: parseInt($('#sendHeight').val())
};
if (Browser.isSafariWebRTC() && Browser.isiOS() && Flashphoner.getMediaProviders()[0] === "WebRTC") {
    constraints.video.deviceId = {exact: $('#videoInput').val()};
}
```

4. Setting FPS

Size	320	240
FPS	15	
Bitrate	min 0	max 0
COD ?	☒	
Mute	☐ ☒ off	

code:

```
if (constraints.video) {
    ...
    if (parseInt($('#fps').val()) > 0)
        constraints.video.frameRate = parseInt($('#fps').val());
}
```

5. Setting video bitrate in kbps

Size	320	240
FPS	30	
Bitrate	min 500	max 1000
COD ?	☒	
Mute	☐ ☒ off	

code:

```
if (constraints.video) {  
    ...  
    if (parseInt($('#sendVideoMinBitrate').val()) > 0)  
        constraints.video.minBitrate = parseInt($('#sendVideoMinBitrate').val());  
    if (parseInt($('#sendVideoMaxBitrate').val()) > 0)  
        constraints.video.maxBitrate = parseInt($('#sendVideoMaxBitrate').val());  
    ...  
}
```

6. Setting CPU Overuse Detection

Size	320	240
FPS	30	
Bitrate	min 0	max 0
COD ?	☐	
Mute	☐ ☒ off	

code:

```
if (!$("#cpuOveruseDetection").is(':checked')) {  
    mediaConnectionConstraints = {  
        "mandatory": {  
            googCpuOveruseDetection: false  
        }  
    }  
}
```

7. Turning off the camera (mute)

Size	320	240
FPS	30	
Bitrate	min 0	max 0
COD 	☒	
Mute	☒	

code:

```
if ($("#muteVideoToggle").is(":checked")) {  
    muteVideo();  
}
```

Testing camera and microphone capturing locally

Local camera and microphone test is intended to check capturing in browser without publishing stream to server.

Media Devices

Local



Preview



WCS

wss://p11.flashphc

Release

Start



Play audio

code:

```

function startTest() {
    Flashphoner.getMediaAccess(getConstraints(), localVideo).then(function (disp) {
        $("#testBtn").text("Release").off('click').click(function () {
            $(this).prop('disabled', true);
            stopTest();
        }).prop('disabled', false);

        window.AudioContext = window.AudioContext || window.webkitAudioContext;
        if (Flashphoner.getMediaProviders()[0] == "WebRTC" && window.AudioContext) {
            for (i = 0; i < localVideo.children.length; i++) {
                if (localVideo.children[i] && localVideo.children[i].id.indexOf("-LOCAL_CACHED_VIDEO") != -1) {
                    var stream = localVideo.children[i].srcObject;
                    audioContextForTest = new AudioContext();
                    var microphone = audioContextForTest.createMediaStreamSource(stream);
                    var javascriptNode = audioContextForTest.createScriptProcessor(1024, 1, 1);
                    microphone.connect(javascriptNode);
                    javascriptNode.connect(audioContextForTest.destination);
                    javascriptNode.onaudioprocess = function (event) {
                        var inpt_L = event.inputBuffer.getChannelData(0);
                        var sum_L = 0.0;
                        for (var i = 0; i < inpt_L.length; ++i) {
                            sum_L += inpt_L[i] * inpt_L[i];
                        }
                        $("#micLevel").text(Math.floor(Math.sqrt(sum_L / inpt_L.length) * 100));
                    }
                }
            }
        } else if (Flashphoner.getMediaProviders()[0] == "Flash") {
            micLevelInterval = setInterval(function () {
                $("#micLevel").text(disp.children[0].getMicrophoneLevel());
            }, 500);
        }
        testStarted = true;
    }).catch(function (error) {
        $("#testBtn").prop('disabled', false);
        testStarted = false;
    });
}

```

SDP parameters replacing

When publishing stream, there is a possibility to replace SDP parameters. In 'SDP replace' field string template is set for search for the parameter to replace, and in 'with' field new parameter value is set.

SDP
replace

0

with

0

To replace SDP parameters, a callback function is used that should be set on stream creation in `sdpHook` option of `createStream()` method:

stream creation [code](#)

```
publishStream = session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  constraints: constraints,
  mediaConnectionConstraints: mediaConnectionConstraints,
  sdpHook: rewriteSdp,
  ...
})
```

rewriteSdp function [code](#)

```
function rewriteSdp(sdp) {
  var sdpStringFind = $("#sdpStringFind").val().replace('\r\n', '\r\n');
  var sdpStringReplace = $("#sdpStringReplace").val().replace('\r\n', '\r\n');
  if (sdpStringFind != 0 && sdpStringReplace != 0) {
    var newSDP = sdp.sdpString.toString();
    newSDP = newSDP.replace(new RegExp(sdpStringFind, "g"), sdpStringReplace);
    return newSDP;
  }
  return sdp.sdpString;
}
```

Rising up the bitrate of video stream published in Chrome browser

SDP parameters replacement allows to rise video stream published bitrate. To do this, SDP parameter 'a' must be replaced by this template when publishing H264 stream:

```
a=fmtp:(.*) (.)
```

to

```
a=fmtp:$1 $2;x-google-min-bitrate=2500
```

where 2500 is the bitrate in kilobytes per second.

Similarly, video bitrate on start can be set (x-google-start-bitrate attribute) and maximum bitrate can be limited (x-google-max-bitrate attribute). Note that if minimum bitrate only is set, then resulting bitrate cannot be above 2500 kbps, probably maximum bitrate value is fixed on this level by default in Chrome browser. When higher bitrate values are required, for example, to publish high resolution streams, both minimum and maximum values must be explicitly set:

```
a=fmtp:$1 $2;x-google-max-bitrate=7000;x-google-min-bitrate=3000
```

In this case browser will try to keep bitrate in limits from 3000 to 7000 kbps when publishing a stream.

When publishing VP8 stream, SDP parameter 'a' must be replaced by this template

```
a=rtpmap:(.*) VP8/90000\r\n
```

to

```
a=rtpmap:$1 VP8/90000\r\na=fmtp:$1 x-google-min-bitrate=3000;x-google-max-bitrate=7000\r\n
```

This feature is available in Chrome browser only.

Bandwidth management

SDP parameters replacement allows to set bandwidth for stream published. To do this, SDP parameter 'c' must be replaced by this template when publishing stream

```
c=IN (.*)\r\n
```

to

```
c=IN $1\r\nb=AS:10000\r\n
```

Setting up codecs

When publishing the stream, there is a possibility to eliminate from WebRTC SDP codecs that should not be used to publish the given stream, for example:

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

Such a capability is handy when you need to find some workaround for bugs of a browser or if it conflicts with the given codec. For example, if H.264 does not work in a browser, you can turn it off and switch to VP8 when working via WebRTC.

Sound device selection

Sound output device can be selected (and switched "on the fly") while stream is playing in Chrome and MS Edge browsers.



code:

```
Flashphoner.getMediaDevices(null, true, MEDIA_DEVICE_KIND.OUTPUT).then(function (list) {
  list.audio.forEach(function (device) {
    ...
  });
}).catch(function (error) {
  $('#audioOutputForm').remove();
});
```

Note that Firefox and Safari browsers always return empty output devices list, therefore sound device selection does not supported for these browsers

WebRTC statistics displaying

A client application can get WebRTC statistics according to the [standard](#) while publishing or playing stream. The statistics can be displayed in browser, for example:

Media Devices



Note that in Safari browser audio only statistics can be displayed.

1. Statistics displaying while stream is published

stream.getStats()code:

```
publishStream.getStats(function (stats) {
  if (stats && stats.outboundStream) {
    if (stats.outboundStream.video) {
      showStat(stats.outboundStream.video, "outVideoStat");
      let vBitrate = (stats.outboundStream.video.bytesSent - videoBytesSent) * 8;
      if ($('#outVideoStatBitrate').length == 0) {
        let html = "<div>Bitrate: " + "<span id='outVideoStatBitrate' style='font-weight:
normal'>" + vBitrate + "</span>" + "</div>";
        $("#outVideoStat").append(html);
      } else {
        $('#outVideoStatBitrate').text(vBitrate);
      }
      videoBytesSent = stats.outboundStream.video.bytesSent;
      ...
    }

    if (stats.outboundStream.audio) {
      showStat(stats.outboundStream.audio, "outAudioStat");
      let aBitrate = (stats.outboundStream.audio.bytesSent - audioBytesSent) * 8;
      if ($('#outAudioStatBitrate').length == 0) {
        let html = "<div>Bitrate: " + "<span id='outAudioStatBitrate' style='font-weight:
normal'>" + aBitrate + "</span>" + "</div>";
        $("#outAudioStat").append(html);
      } else {
        $('#outAudioStatBitrate').text(aBitrate);
      }
      audioBytesSent = stats.outboundStream.audio.bytesSent;
    }
  }
  ...
});
```

2. Statistics displaying while stream is played

stream.getStats()code:

```

    previewStream.getStats(function (stats) {
      if (stats && stats.inboundStream) {
        if (stats.inboundStream.video) {
          showStat(stats.inboundStream.video, "inVideoStat");
          let vBitrate = (stats.inboundStream.video.bytesReceived - videoBytesReceived) * 8;
          if ($('#inVideoStatBitrate').length == 0) {
            let html = "<div>Bitrate: " + "<span id='inVideoStatBitrate' style='font-weight:
normal'>" + vBitrate + "</span>" + "</div>";
            $('#inVideoStat').append(html);
          } else {
            $('#inVideoStatBitrate').text(vBitrate);
          }
          videoBytesReceived = stats.inboundStream.video.bytesReceived;
          ...
        }

        if (stats.inboundStream.audio) {
          showStat(stats.inboundStream.audio, "inAudioStat");
          let aBitrate = (stats.inboundStream.audio.bytesReceived - audioBytesReceived) * 8;
          if ($('#inAudioStatBitrate').length == 0) {
            let html = "<div style='font-weight: bold'>Bitrate: " + "<span id='inAudioStatBitrate'
style='font-weight: normal'>" + aBitrate + "</span>" + "</div>";
            $('#inAudioStat').append(html);
          } else {
            $('#inAudioStatBitrate').text(aBitrate);
          }
          audioBytesReceived = stats.inboundStream.audio.bytesReceived;
        }
        ...
      }
    });

```

Picture parameters management

When video stream is published, it is possible to manage picture resolution and frame rate with constraints.

Picture resolution management

Picture resolution can be exactly set

```
constraints = {audio:true, video:{width:320,height:240}}
```

However, in some cases it is necessary to set height and width as range

```
constraints = {audio:true, video:{width:{min:160,max:320},height:{min:120,max:240}}}
```

For some browsers, iOS Safari for example, exact values should be set as range (in last [versions](#) there is a workaround on WebSDK level)

```
constraints = {audio:true, video:{width:{min:320,max:320},height:{min:240,max:240}}}
```

Frame rate management

Frame rate can be set exactly

```
constraints = {audio:true, video:{frameRate:30}}
```

or as range

```
constraints = {audio:true, video:{frameRate:{min:15,max:30}}}
```

In some cases, for example if web camera supports 24 fps only and frame rate is set to 30 fps then stream publishing will fail. In this case frame rate should be set as ideal

```
constraints = {audio:true, video:{frameRate:{ideal:30}}}
```

Seamless switching between web camera stream and screen share stream during publication

During webinars it is often necessary to switch between streams captured from conductors web camera and screen share while streams are published. Ideally, such switching should be seamless, and conductors microphone soundtrack must be continued. In latest WebSDK versions this feature is implemented for Chrome and Firefox browsers, let's look at the sample in Media Devices application.

Media Devices

Video stats
Bytes sent: 389875
Packets sent: 430
Frames encoded: 183
Audio stats
Bytes sent: 33404
Packets sent: 420

Local



640x480

WCS

wss://test2.flashphoner.com:84

PUBLISHING

Preview



640x480

Video stats
Bytes received: 382269
Packets received: 389
Frames decoded: 166
Audio stats
Bytes received: 31894
Packets received: 399

Screen share☐ off

Size

640

480

FPS

30

Bitrate

min

0

max

0

COD ☒

Mute☐ off

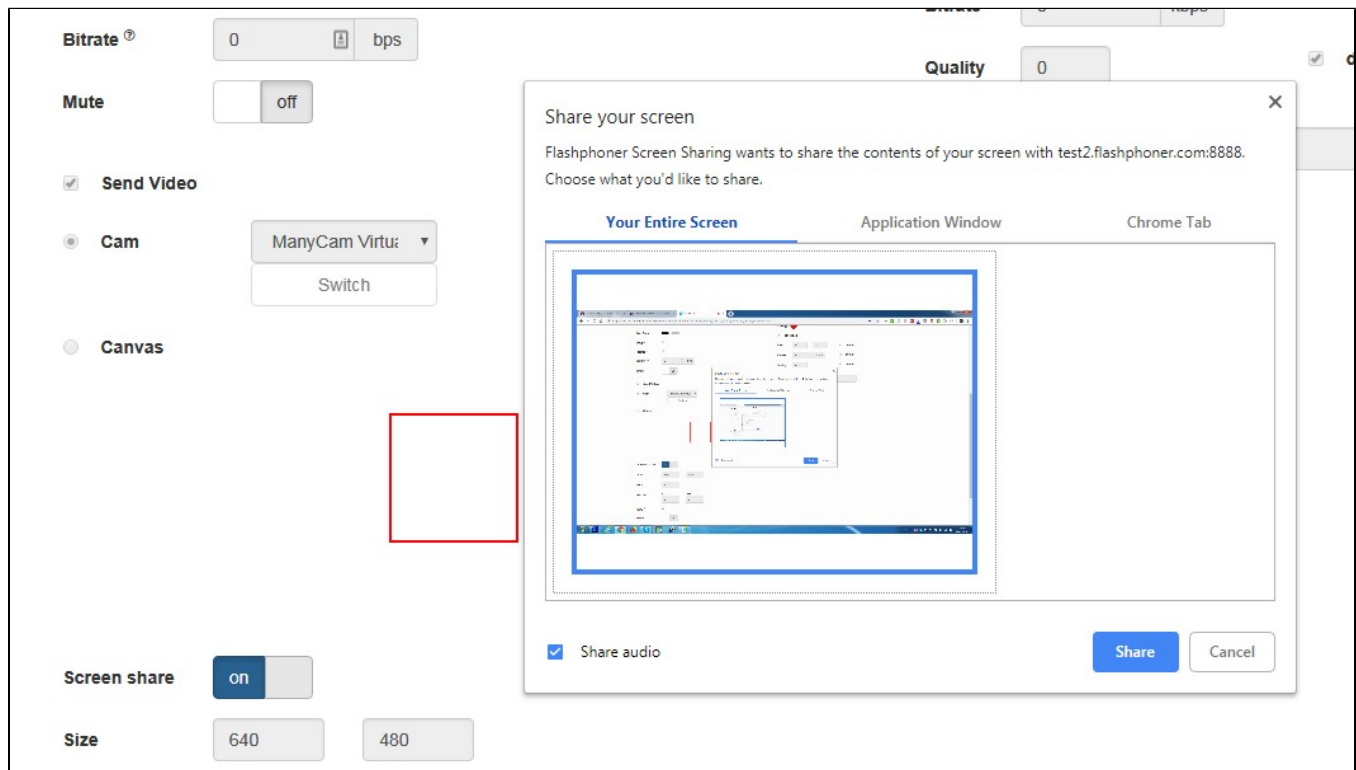
1. While streaming from chosen microphone and camera, when switch 'Screen share' is set to 'on' switchToScreen function will be invoked

`stream.switchToScreen`

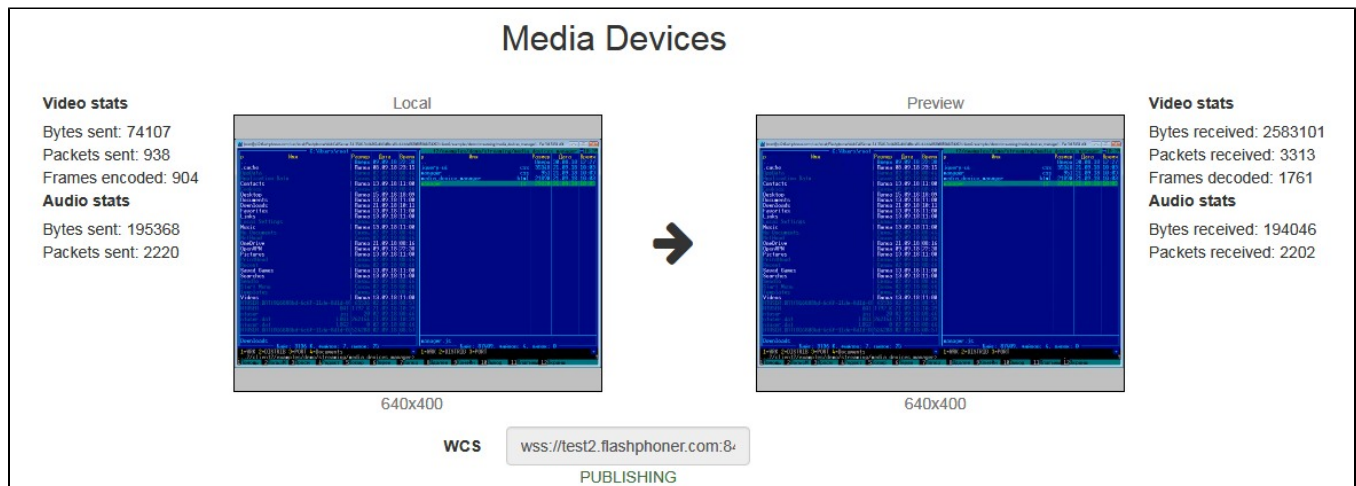
```
function switchToScreen() {
  if (publishStream) {
    $('#switchBtn').prop('disabled', true);
    $('#videoInput').prop('disabled', true);
    publishStream.switchToScreen($('#mediaSource').val()).catch(function () {
      $('#screenShareToggle').removeAttr("checked");
      $('#switchBtn').prop('disabled', false);
      $('#videoInput').prop('disabled', false);
    });
  }
}
```

Media stream source (screen) should be passed to this function.

2. Then user should choose whole screen or certain program window to share with Chrome [extension](#) or with Firefox dialog box:



3. Screen share stream is published to server



Sound translation source is not changed.

4. To revert back web camera streaming switchToCam function is invoked

stream.switchToCam`code`

```
function switchToCam() {  
    if (publishStream) {  
        publishStream.switchToCam();  
        $('#switchBtn').prop('disabled', false);  
        $('#videoInput').prop('disabled', false);  
    }  
}
```

Known limitations

1. Stream switching works in Chrome and Firefox browsers only.
2. It is impossible to switch web camera while screen share stream is published.
3. [Extension](#) is necessary to capture screen in Chrome browser.
4. Switching from web camera to screen share stream and back again works only if web camera stream was published first.

Known issues

1. Microphone swithing does not work in Safari browser.

Symptoms: microphone does not switch usingswitchMic() WCS WebSDK method.

Solution: use another browser, because Safari always uses sound input microphone, that is chosen in system sound menu (hold down the option (alt) button and click on the sound icon in the menu bar). When microphone is chosen in sound menu, Mac reboot is required.

If Logitech USB camers microphone does not work (when it is chosen in sound menu), format / sample rate changing in Audio MIDI Setup and rebooting can help.

2. iOS Safari freezes on playback when publisher changes a camera.

Symptoms: when camera is changed, the stream published playback freezes in iOS Safari browser.

Solution: enable transcoding using the following parameter in [flashphoner.properties](#) file

```
disable_streaming_proxy=true
```

or set a fixed resolution in player script for stream playback

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