

Media Devices

Example of streamer with access to media devices

This streamer can be used to publish or playback WebRTC streams on Web Call Server and allows to select media devices and parameters for the published video

- camera
- microphone
- FPS (Frames Per Second)
- resolution (width, height)

On the screenshot below a stream is being published from the client.

The screenshot displays a web interface titled "Media Devices". It features two video elements side-by-side. The left element, labeled "Local", shows a video of a pig in a field and is in a "PUBLISHING" state. The right element, labeled "Player", shows the same video and is in a "PLAYING" state. Both video elements have a resolution of 640x360. Below the videos are controls for "test" and "Stop". At the bottom, there is a "ws://localhost:8080" address bar with a "Disconnect" button and a "Timeout" field set to "1000 msec". The status "ESTABLISHED" is shown at the bottom center. On the left and right sides of the interface, there are "Video stats" and "Audio stats" sections with various performance metrics.

Category	Metric	Value
Video stats	Codec	H264
	Codec Rate	90000
	Packets Sent	1185
	Bytes Sent	841051
	Fir Count	0
	Pli Count	2
	Nack Count	0
Audio stats	Codec	opus
	Codec Rate	48000
	Packets Sent	699
	Bytes Sent	112699
	Nack Count	0
	Bitrate	64496
	Connection	

Category	Metric	Value
Video stats	Codec	H264
	Codec Rate	90000
	Packets Lost	0
	Packets Received	518
	Bytes Received	541528
	Jitter Buffer Emitted Count	188
	Fir Count	0
Audio stats	Codec	opus
	Codec Rate	48000
	Packets Lost	0
	Packets Discarded	0
	Packets Received	346
	Bytes Received	55866
	Jitter Buffer Emitted Count	330240

Two video elements are displayed on the page

- **Local** - video from the camera
- **Player** - the video received from the server

Code of the example

The path to the source code of the example on WCS server is:

`/usr/local/FlashphonerWebCallServer/client/examples/demo/streaming/media_devices_manager`

- `manager.css` - file with styles
- `media_device_manager.html` - page of the streamer
- `manager.js` - script providing functionality for the streamer

This example can be tested using the following address:

https://host:8888/client/examples/demo/streaming/media_devices_manager/media_device_manager.html

Here host is the address of the WCS server.

Analyzing the code

To analyze the code, let's take the version of file `manager.js` with hash `ecbadc3`, which is available [here](#) and can be downloaded with corresponding build `2.0.212`.

1. Initialization of the API

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

```
Flashphoner.init({
  screenSharingExtensionId: extensionId,
  mediaProvidersReadyCallback: function (mediaProviders) {
    if (Flashphoner.isUsingTemasys()) {
      $("#audioInputForm").hide();
      $("#videoInputForm").hide();
    }
  }
})
```

2. List available input media devices

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

When input media devices are listed, drop-down lists of microphones and cameras on client page are filled.

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

3. List available output media devices

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

When output media devices are listed, drop-down lists of speakers and headphones on client page are filled.

```
Flashphoner.getMediaDevices(null, true, MEDIA_DEVICE_KIND.OUTPUT).then(function (list)
{
  list.audio.forEach(function (device) {
```

```

        ...
    });
    ...
}).catch(function (error) {
    $("#notifyFlash").text("Failed to get media devices");
});

```

4. Get audio and video publishing constraints from client page

`getConstraints()` code

Publishing sources:

- camera (`sendVideo`)
- microphone (`sendAudio`)

```

constraints = {
    audio: $("#sendAudio").is(':checked'),
    video: $("#sendVideo").is(':checked'),
};

```

Audio constraints:

- microphone choice (`deviceId`)
- error correction for Opus codec (`fec`)
- stereo mode (`stereo`)
- audio bitrate (`bitrate`)

```

if (constraints.audio) {
    constraints.audio = {
        deviceId: $('#audioInput').val()
    };
    if ($("#fec").is(':checked'))
        constraints.audio.fec = $("#fec").is(':checked');
    if ($("#sendStereoAudio").is(':checked'))
        constraints.audio.stereo = $("#sendStereoAudio").is(':checked');
    if (parseInt($('#sendAudioBitrate').val()) > 0)
        constraints.audio.bitrate = parseInt($('#sendAudioBitrate').val());
}

```

Video constraints:

- camera choice (`deviceId`)
- publishing video size (`width`, `height`)
- minimal and maximal video bitrate (`minBitrate`, `maxBitrate`)
- FPS (`frameRate`)

```

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

```

```

        constraints.video.deviceId = {exact: $('#videoInput').val()};
    }
    if (parseInt($('#sendVideoMinBitrate').val()) > 0)
        constraints.video.minBitrate = parseInt($('#sendVideoMinBitrate').val());
    if (parseInt($('#sendVideoMaxBitrate').val()) > 0)
        constraints.video.maxBitrate = parseInt($('#sendVideoMaxBitrate').val());
    if (parseInt($('#fps').val()) > 0)
        constraints.video.frameRate = parseInt($('#fps').val());

```

5. Get access to media devices for local test

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

Audio and video constraints and `div` element to display captured video are passed to the method.

```

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

```

6. Connecting to the server

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

```

Flashphoner.createSession({urlServer: url, timeout: tm}).on(SESSION_STATUS.ESTABLISHED,
function (session) {
    ...
}).on(SESSION_STATUS.DISCONNECTED, function () {
    ...
}).on(SESSION_STATUS.FAILED, function () {
    ...
});

```

7. Receiving the event confirming successful connection

`ConnectionStatusEvent ESTABLISHED` [code](#)

```

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

```

8. Stream publishing

`Session.createStream()`, `Stream.publish()` [code](#)

```

publishStream = session.createStream({
  name: streamName,
  display: localVideo,
  cacheLocalResources: true,
  constraints: constraints,
  mediaConnectionConstraints: mediaConnectionConstraints,
  sdpHook: rewriteSdp,
  transport: transportInput,
  cvoExtension: cvo,
  stripCodecs: strippedCodecs,
  videoContentHint: contentHint
  ...
});
publishStream.publish();

```

9. Receiving the event confirming successful streaming

`StreamStatusEvent PUBLISHING` code

```

publishStream = session.createStream({
  ...
}).on(STREAM_STATUS.PUBLISHING, function (stream) {
  $("#testBtn").prop('disabled', true);
  var video = document.getElementById(stream.id());
  //resize local if resolution is available
  if (video.videoWidth > 0 && video.videoHeight > 0) {
    resizeLocalVideo({target: video});
  }
  enablePublishToggles(true);
  if ($("#muteVideoToggle").is(":checked")) {
    muteVideo();
  }
  if ($("#muteAudioToggle").is(":checked")) {
    muteAudio();
  }
  //remove resize listener in case this video was cached earlier
  video.removeEventListener('resize', resizeLocalVideo);
  video.addEventListener('resize', resizeLocalVideo);
  publishStream.setMicrophoneGain(currentGainValue);
  setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
  onPublishing(stream);
}).on(STREAM_STATUS.UNPUBLISHED, function () {
  ...
}).on(STREAM_STATUS.FAILED, function () {
  ...
});
publishStream.publish();

```

10. Stream playback

`Session.createStream()`, `Stream.play()` code

```

previewStream = session.createStream({
  name: streamName,
  display: remoteVideo,
  constraints: constraints,
  transport: transportOutput,
  stripCodecs: strippedCodecs
  ...

```

```
});  
previewStream.play();
```

11. Receiving the event confirming successful playback

`StreamStatusEvent PLAYING` [code](#)

```
previewStream = session.createStream({  
    ...  
}).on(STREAM_STATUS.PLAYING, function (stream) {  
    playConnectionQualityStat.connectionQualityUpdateTimestamp = new Date().valueOf();  
    setStatus("#playStatus", stream.status());  
    onPlaying(stream);  
    document.getElementById(stream.id()).addEventListener('resize', function (event) {  
        $("#playResolution").text(event.target.videoWidth + "x" +  
event.target.videoHeight);  
        resizeVideo(event.target);  
    });  
    //wait for incoming stream  
    if (Flashphoner.getMediaProviders()[0] == "WebRTC") {  
        setTimeout(function () {  
            if(Browser.isChrome()) {  
                detectSpeechChrome(stream);  
            } else {  
                detectSpeech(stream);  
            }  
        }, 3000);  
    }  
    ...  
});  
previewStream.play();
```

12. Stop stream playback

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

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

13. Receiving the event confirming successful playback stop

`StreamStatusEvent STOPPED` [code](#)

```
previewStream = session.createStream({  
    ...  
}).on(STREAM_STATUS.STOPPED, function () {  
    setStatus("#playStatus", STREAM_STATUS.STOPPED);  
    onStopped();  
    ...  
});  
previewStream.play();
```

14. Stop stream publishing

`Stream.stop()` code

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

15. Receiving the event confirming successful publishing stop

`StreamStatusEvent UNPUBLISHED` code

```
publishStream = session.createStream({
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function () {
    setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
    onUnpublished();
    ...
});
publishStream.publish();
```

16. Mute publisher audio

`Stream.muteAudio()` code:

```
function muteAudio() {
    if (publishStream) {
        publishStream.muteAudio();
    }
}
```

17. Mute publisher video

`Stream.muteVideo()` code:

```
function muteVideo() {
    if (publishStream) {
        publishStream.muteVideo();
    }
}
```

18. Show WebRTC stream publishing statistics

`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;
}
}
...
});

```

19. Show WebRTC stream playback statistics

`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;
        }
        ...
    }
});

```


20. Speech detection using ScriptProcessor interface (any browser except Chrome)

`AudioContext.createMediaStreamSource()`, `AudioContext.createScriptProcessor()` [code](#)

```
function detectSpeech(stream, level, latency) {
  var mediaStream = document.getElementById(stream.id()).srcObject;
  var source = audioContext.createMediaStreamSource(mediaStream);
  var processor = audioContext.createScriptProcessor(512);
  processor.onaudioprocess = handleAudio;
  processor.connect(audioContext.destination);
  processor.clipping = false;
  processor.lastClip = 0;
  // threshold
  processor.threshold = level || 0.10;
  processor.latency = latency || 750;

  processor.isSpeech =
    function () {
      if (!this.clipping) return false;
      if ((this.lastClip + this.latency) < window.performance.now())
        this.clipping = false;
      return this.clipping;
    };

  source.connect(processor);

  // Check speech every 500 ms
  speechIntervalID = setInterval(function () {
    if (processor.isSpeech()) {
      $("#talking").css('background-color', 'green');
    } else {
      $("#talking").css('background-color', 'red');
    }
  }, 500);
}
```

Audio data handler [code](#)

```
function handleAudio(event) {
  var buf = event.inputBuffer.getChannelData(0);
  var bufLength = buf.length;
  var x;
  for (var i = 0; i < bufLength; i++) {
    x = buf[i];
    if (Math.abs(x) >= this.threshold) {
      this.clipping = true;
      this.lastClip = window.performance.now();
    }
  }
}
```

21. Speech detection using incoming audio WebRTC statistics in Chrome browser

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

```
function detectSpeechChrome(stream, level, latency) {
  statSpeechDetector.threshold = level || 0.010;
  statSpeechDetector.latency = latency || 750;
  statSpeechDetector.clipping = false;
  statSpeechDetector.lastClip = 0;
```

```
speechIntervalID = setInterval(function() {
  stream.getStats(function(stat) {
    let audioStats = stat.inboundStream.audio;
    if(!audioStats) {
      return;
    }
    // Using audioLevel WebRTC stats parameter
    if (audioStats.audioLevel >= statSpeechDetector.threshold) {
      statSpeechDetector.clipping = true;
      statSpeechDetector.lastClip = window.performance.now();
    }
    if ((statSpeechDetector.lastClip + statSpeechDetector.latency) <
window.performance.now()) {
      statSpeechDetector.clipping = false;
    }
    if (statSpeechDetector.clipping) {
      $("#talking").css('background-color', 'green');
    } else {
      $("#talking").css('background-color', 'red');
    }
  });
}, 500);
}
```