Media Devices

Пример стримера с доступом к медиа-устройствам

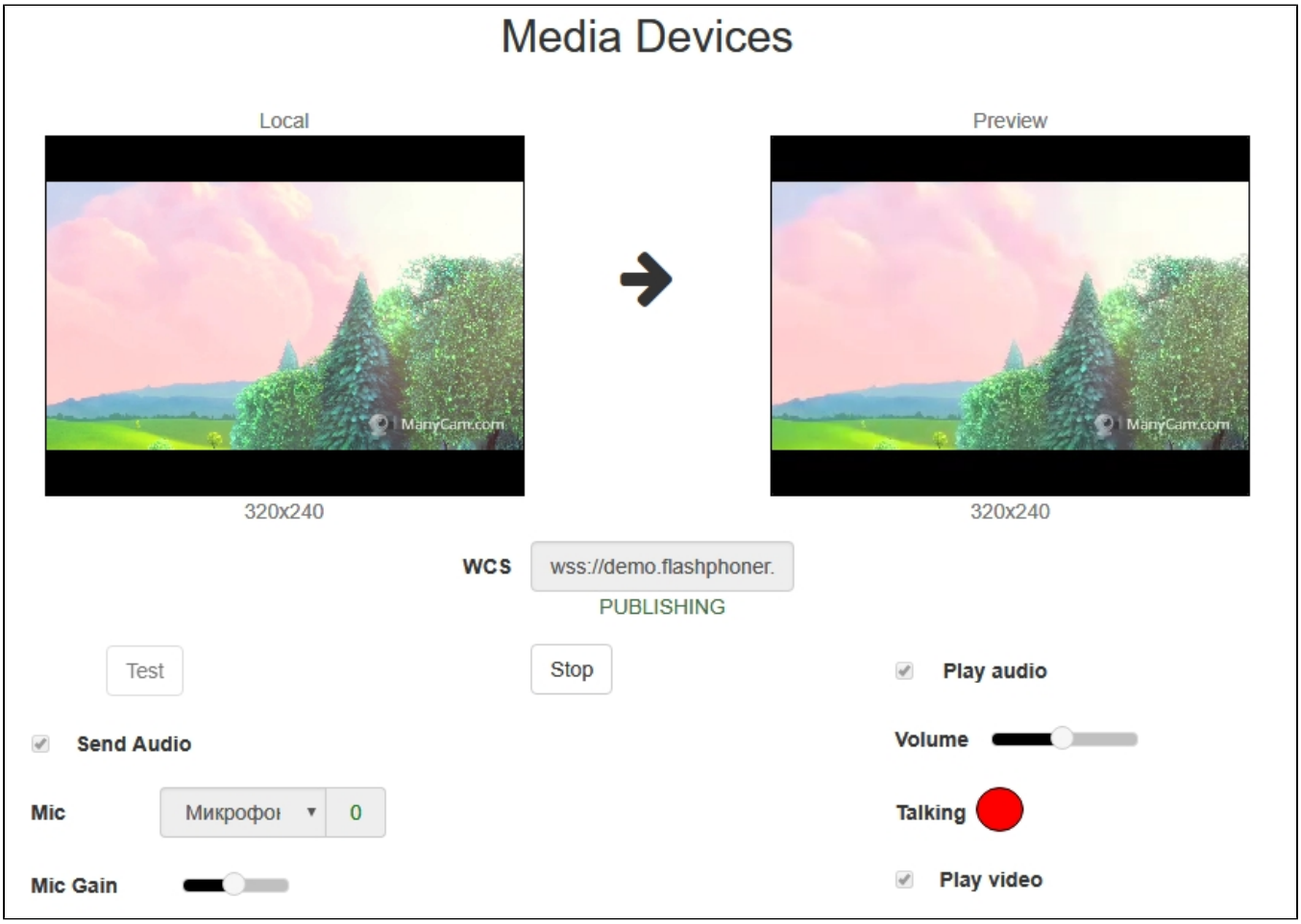
Данный стример может использоваться для публикации следующих типов потоков с Web Call Server

- WebRTC
- RTMFP
- RTMP

и позволяет выбрать медиа-устройства и параметры для публикуемого видео

- камера
- микрофон
- FPS (Frames Per Second)
- разрешение (ширина, высота)

На скриншоте ниже представлен пример во время публикации потока.



На странице вопроизводятся два видео:

- 'Local' - видео с камеры
- 'Preview' - видео, которое приходит с сервера

Код примера

Код данного примера находится на WCS-сервере по следующему пути:

/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/media_devices_manager

manager.css - файл стилей
media_device_manager.html - страница стримера
manager.js - скрипт, обеспечивающий работу стримера

Тестировать данный пример можно по следующему адресу:

https://host:8888/client2/examples/demo/streaming/media_devices_manager/media_device_manager.html

Здесь host - адрес WCS-сервера.

Работа с кодом примера

Для разбора кода возьмем версию файла manager.js с хэшем 66cc393, которая находится здесь и доступна для скачивания в соответствующей сборке 0.5.28.2753.133.

1. Инициализация API.

Flashphoner.init() code

```
        Flashphoner.init({
            screenSharingExtensionId: extensionId,
            flashMediaProviderSwfLocation: '../../../../media-provider.swf',
            mediaProvidersReadyCallback: function (mediaProviders) {
                //hide remote video if current media provider is Flash
                if (mediaProviders[0] == "Flash") {
                    $("#fecForm").hide();
                    $("#stereoForm").hide();
                    $("#sendAudioBitrateForm").hide();
                    $("#cpuOveruseDetectionForm").hide();
                }
                if (Flashphoner.isUsingTemasys()) {
                    $("#audioInputForm").hide();
                    $("#videoInputForm").hide();
                }
            }
        })
```

2. Получение списка доступных медиа-устройств ввода

Flashphoner.getMediaDevices() code

При получении списка медиа-устройств заполняются выпадающие списки микрофонов и камер на странице клиента.

```
    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. Получение списка доступных медиа-устройств вывода звука

Flashphoner.getMediaDevices() code

При получении списка медиа-устройств заполняются выпадающий список устройств вывода звука на странице клиента.

```
    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. Получение граничных параметров для публикации аудио и видео со страницы клиента

getConstraints() code

Источники публикации:

- камера (sendVideo)
- микрофон (sendAudio)

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

Параметры аудио:

- выбор микрофона (deviceId)
- коррекция ошибок для кодека Opus (fec)
- режим стерео (stereo)
- битрейт аудио (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());
    }
```

Параметры видео:

- выбор камеры (deviceId)
- размеры при публикации (width, height)
- минимальный и максимальный битрейт видео (minBitrate, maxBitrate)
- FPS (frameRate)

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

Flashphoner.getMediaAccess() code

В метод передаются граничные параметры для аудио и видео (constrains), а также localVideo - div-элемент, в котором будет отображаться видео с выбранной камеры.

```
    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. Подключение к серверу.

Flashphoner.createSession() code

```
    Flashphoner.createSession({urlServer: url, timeout: tm}).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();
    });
```

7. Получение от сервера события, подтверждающего успешное соединение.

ConnectionStatusEvent ESTABLISHEDcode

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

8. Публикация видеопотока

session.createStream(), publishStream.publish() code

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

9. Получение от сервера события, подтверждающего успешную публикацию потока

StreamStatusEvent PUBLISHINGcode

```
    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. Воспроизведение потока

session.createStream(), previewStream.play() code

```
    previewStream = session.createStream({
        name: streamName,
        display: remoteVideo,
        constraints: constraints,
        transport: transportOutput,
        stripCodecs: strippedCodecs
        ...
    });
    previewStream.play();
```

11. Получение от сервера события, подтверждающего успешное воспроизведение потока

StreamStatusEvent PLAYINGcode

```
    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 () {
                detectSpeech(stream);
            }, 3000);
        }
        ...
    });
    previewStream.play();
```

12. Остановка воспроизведения потока.

stream.stop() code

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

13. Получение от сервера события, подтверждающего остановку воспроизведения

StreamStatusEvent STOPPEDcode

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

14. Остановка публикации видеопотока

stream.stop() code

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

15. Получение от сервера события, подтверждающего успешную остановку публикации

StreamStatusEvent UNPUBLISHEDcode

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

16. Отключение микрофона

code:

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

17. Отключение камеры

code:

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

18. Отображение статистики при публикации потока

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. Отображение статистики при воспроизведениипотока

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;
                }
                ...
            }
        });
```