

MCU client

- [Пример клиента для участника многоточечной конференции](#)
- [Код примера](#)
- [Работа с кодом примера](#)

Пример клиента для участника многоточечной конференции

Данный пример может использоваться для организации многоточечной видео конференции ([MCU](#)) на Web Call Server. Каждый участник такой конференции может публиковать WebRTC-поток и воспроизводить микшированный поток с аудио и видео других участников и собственным видео (без собственного аудио).

Для работы примера требуются следующие настройки в конфиге [flashphoner.properties](#) WCS-сервера

```
mixer_auto_start=true  
mixer_mcu_audio=true  
mixer_mcu_video=true
```

При подключении нового участника,использующего данный клиент, к конференции

- публикуется поток с видео участника и именем <participantName> + "#" + <roomName>
- этот поток добавляется к микшеру с именем <roomName> (если такой микшер еще не существует, то он создается)
- публикуется другой микшер с именем <roomName> + "-" + <participantName> + <roomName>, который содержит видео всех участников (включая данного) и аудио только от других участников, и начинается воспроизведение этого микшера

На скриншоте ниже участник конференции публикует поток и воспроизводит микшированный поток конференции:

MCU Client

Before use: please set the server parameters as described [here](#)



Conference

WCS URL

ws://localhost:8080



Login

Alice

Room

room1

Volume



Audio



Full Screen



PLAYING

Leave

Код примера

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

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

mcu_client.css- файл стилей

mcu_client.html- страница участника MCU-конференции

mcu_client.js- скрипт для участия в MCU-конференции

Тестировать данный пример можно по следующему адресу:

https://host:8888/client/examples/demo/streaming/mcu_client/mcu_client.html

Здесь host - адрес WCS-сервера.

Работа с кодом примера

Для разбора кода возьмем файл `mcu_client.js` с хешем `ecbadc3`, который находится [здесь](#) и доступен для скачивания в соответствующей сборке [2.0.212](#).

1. Инициализация API

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

```
Flashphoner.init();
```

2. Подключение к серверу

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

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

3. Получение от сервера события, подтверждающего успешное соединение

`ConnectionStatusEvent ESTABLISHED`[code](#)

При получении данного события публикуется видеопоток этого участника

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

4. Получение граничных параметров для публикации и воспроизведения со страницы клиента

`getConstraints()` [code](#)

Параметр `audio:true` или `false` (в зависимости от значения и публикация, и воспроизведение будут производиться с аудио или без)

Параметр `video:true` (потоки будут публиковаться и воспроизводиться с видео)

```
function getConstraints() {  
    var constraints = {  
        audio: $("#hasAudio").is(':checked'),  
        video: true  
    };  
    return constraints;  
}
```

5. Публикация видеопотока

`session.createStream(), stream.publish()` [code](#)

При создании потока передаются следующие параметры

- `streamName` - имя потока, публикуемого участником конференции (в данном случае `login + "#" + roomName`, где `login` - имя участника)
- `mockLocalDisplay` - div-элемент, требуемый для отображения видео локальной камеры (в данном случае не будет виден пользователю)

- constraints - getConstraints() [code](#) (чтобы указать, должна ли публикация производиться с аудио)

```
publishStream = session.createStream({
  name: streamName,
  display: mockLocalDisplay,
  receiveVideo: false,
  receiveAudio: false,
  constraints: getConstraints()
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
});
publishStream.publish();
```

6. Получение от сервера события, подтверждающего успешную публикацию потока

StreamStatusEvent PUBLISHING [code](#)

При получении данного события создается видеопоток для воспроизведения микшера конференции для этого участника

```
publishStream = session.createStream({
  ...
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  //play preview
  playStream(session);
  ...
});
```

7. Воспроизведение потока конференции

session.createStream(), stream.play() [code](#)

При создании потока передаются следующие параметры

- streamName - имя микшера, который будет воспроизводиться для участника (в данном случае roomName + "-" + login + roomName, где login - имя участника)
- remoteVideo - div-элемент, в котором будет отображаться видео
- constraints - getConstraints() [code](#) (чтобы указать, должно ли воспроизводиться аудио)

```
conferenceStream = session.createStream({
  name: streamName,
  display: remoteVideo,
  constraints: getConstraints()
  ...
});
conferenceStream.play();
```

8. Получение от сервера события, подтверждающего воспроизведение потока

StreamStatusEvent PLAYING [code](#)

```
conferenceStream = session.createStream({
  ...
}).on(STREAM_STATUS.PENDING, function (stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
  $("#preloader").hide();
  setStatus(stream.status());
  onStart();
}).on(STREAM_STATUS.STOPPED, function () {
  ...
}).on(STREAM_STATUS.FAILED, function (stream) {
  ...
});
```

9. Остановка публикации и воспроизведения видеопотоков при отключении от конференции

stream.stop() [code](#)

```
function stopStreams() {
  if(conferenceStream) {
    conferenceStream.stop();
  }
  if(publishStream) {
    publishStream.stop();
  }
}
```

10. Получение от сервера события, подтверждающего остановку публикации

StreamStatusEvent UNPUBLISHED [code](#)

```
publishStream = session.createStream({
  ...
}).on(STREAM_STATUS.PUBLISHING, function (publishStream) {
  ...
}).on(STREAM_STATUS.UNPUBLISHED, function () {
  onStopped();
}).on(STREAM_STATUS.FAILED, function (stream) {
  ...
});
```

11. Получение от сервера события, подтверждающего остановку воспроизведения

StreamStatusEvent STOPPED [code](#)

```
conferenceStream = session.createStream({
  ...
}).on(STREAM_STATUS.PENDING, function (stream) {
  ...
}).on(STREAM_STATUS.PLAYING, function (stream) {
  ...
}).on(STREAM_STATUS.STOPPED, function () {
  $("#preloader").hide();
  setStatus(STREAM_STATUS.STOPPED);
  onStopped();
}).on(STREAM_STATUS.FAILED, function (stream) {
  ...
});
```