

MCU client

Example of client for MCU conference participant

This example can be used to arrange an [MCU](#) video conference on Web Call Server. Each participant of such conference can publish a WebRTC stream and play a mixer stream with audio and video from the other participants and own video (without own audio).

The following settings are required in WCS [flashphoner.properties](#)

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

When a participant joins a conference using the client

- a stream with video of the participant, named `participantName` + `#` + `roomName`, is published, for example `Alice#room1`
- the participant's stream is added to mixer named `roomName` (if such mixer does not exist, it is auto created)
- a new mixer output stream named `roomName` + `-` + `participantName` + `roomName` and containing video from all the participants (including this one) and audio only from the other participants is created and played for the participant, for example `room1-Aliceroom1`


On the screenshot below the participant is publishing a stream and playing his conference mixer stream:

MCU Client

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



Conference

WCS URL 

Login

Room

Volume

Audio ☒

Full Screen

PLAYING

Code of the example

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

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

- mcu_client.css - file with styles
- mcu_client.html - page of MCU conference participant
- mcu_client.js - script providing functionality for participating in MCU conference

This example can be tested using the following address:

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

Here host is the address of the WCS server.

Analyzing the code

To analyze the code, let's take file `mcu_client.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();
```

2. Connection to server

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

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

3. Receiving the event confirming successful connection

`ConnectionStatusEvent ESTABLISHED` [code](#)

On receiving the event, streaming is started

```
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. Get publishing and playing constraints from the client page

`getConstraints()` [code](#)

Audio constraint: `true` or `false` (depending on the value both published and played stream will have or have not audio)

Video constraint: `true` (published and played streams will have video)

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

5. Video streaming

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

When stream is created, the following parameters are passed

- `streamName` - name of the stream (`login` + `#` + `roomName` in this case, where login is the name of the participant)
- `mockLocalDisplay` - `div` element, required for the local camera video (will not be displayed to the user in the case)
- `constraints` - `getConstraints()` invocation (in this case is used to specify if the published stream will have audio)

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

6. Receiving the event confirming successful streaming

`StreamStatusEvent PUBLISHING` [code](#)

On receiving the event, a stream for playing the participant's conference mixer is created

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

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

7. Playback of conference stream

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

When stream is created, the following parameters are passed

- `streamName` - name of the stream (`roomName` + `-` + `login` + `roomName` in this case, where login is the name of the participant)
- `remoteVideo` - `div` element, in which the video will be displayed
- `constraints` - `getConstraints()` invocation (in this case is used to specify if the played stream will have audio)

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

8. Receiving the event confirming playback

`StreamStatusEvent PLAYING` [code](#)

```
conferenceStream = session.createStream({
  name: streamName,
  display: remoteVideo,
  constraints: getConstraints()
}).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. Stop of playback and streaming on leaving the conference

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

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

10. Receiving the event confirming streaming stop

`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. Receiving the event confirming playback stop

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