

Screen Sharing with Camera

Overview

The example shows a presentation case: webcam and screen sharing streams publishing from a single page with optional streams mixing at server side.

Screen sharing parameters:

- **FPS** - framerate per second
- **Width** - picture width
- **Height** - picture height

Server stream mixing parameters:

- **Add to mixer** - add streams automatically to mixer
- **Mixer** - mixer name

Connection parameters:

- Websocket URL of WCS sever

Screen Sharing with Camera

FPS: 5 Width: 1280 Height: 720

Add to Mixer? Mixer: mixer

My Screen

test-screen#mixer
PUBLISHING

wss://test1.flashphoner.com:8443/test

My Camera

test-camera#mixer
PUBLISHING

Disconnect Stop

The code of the example

The code of the example is on WCS server by the following path:

```
/usr/local/FlashphonerWebCallServer/client2/examples/demo/streaming/screen-camera-mixer
```

- `screen-camera-mixer.css` - styles file
- `screen-camera-mixer.html` - example HTML page
- `screen-camera-mixer.js` - main example script

The example may be tested by the following address:

```
https://host:8888/client2/examples/demo/streaming/screen-camera-mixer/screen-camera-mixer.html
```

where `host` is WCS server address.

Analyzing the code

To analyze the code let's take a `screen-camera-mixer.js` version with hash 32144d9, which is available [here](#) and may be downloaded in build [2.0.243](#).

1. API initialization.

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

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

2. Connecting to the server

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

```
const connect = function() {
  ...
  let url = field("url");
  ...
  console.log("Create new session with url " + url);
  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

`STREAM_STATUS.ESTABLISHED` [code](#)

```
const connect = function() {
  ...
  let url = field("url");
  ...
  console.log("Create new session with url " + url);
  Flashphoner.createSession({urlServer:
url}).on(SESSION_STATUS.ESTABLISHED, function(session){
  //session connected, start streaming
  setStatus("screen", SESSION_STATUS.ESTABLISHED);
  setStatus("camera", SESSION_STATUS.ESTABLISHED);
  onConnected(session);
}).on(SESSION_STATUS.DISCONNECTED, function(){
  ...
}).on(SESSION_STATUS.FAILED, function(){
  ...
});
}
```

4. Screen sharing stream publishing

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

The following parameters are passed to `createStream()` method:

- `streamName` - stream name
- `constraints.video.width` - picture width
- `constraints.video.height` - picture height
- `constraints.video.frameRate` - publishing framerate
- `constraints.video.type: "screen"` - stream type: screen sharing
- `constraints.video.withoutExtension: true` - screen sharing from browser without a special extension
- `constraints.video.mediaSource: "screen"` - for screen sharing from Firefox browser only
- `localVideoScreen` - div tag to display a local video
- `disableConstraintsNormalization = true` - disable constraints normalization (for MacOS Safari only)

```

const startStreamingScreen = function(session) {
  let streamName = getStreamName("screen", field("url"));
  let constraints = {
    video: {
      width: parseInt($('#width').val()),
      height: parseInt($('#height').val()),
      frameRate: parseInt($('#fps').val()),
      type: "screen",
      withoutExtension: true
    }
  };
  if (Browser.isFirefox()) {
    constraints.video.mediaSource = "screen";
  }
  let options = {
    name: streamName,
    display: localVideoScreen,
    constraints: constraints
  }
  if (isSafariMacOS()) {
    options.disableConstraintsNormalization = true;
  }
  session.createStream(options).on(STREAM_STATUS.PUBLISHING,
function(screenStream) {
  ...
}).on(STREAM_STATUS.UNPUBLISHED, function() {
  ...
}).on(STREAM_STATUS.FAILED, function(stream) {
  ...
}).publish();
}

```

5. Receiving the event confirming successful screen publishing

`STREAM_STATUS.PUBLISHING` [code](#)

Web camera stream publishing starts on this event

```

const startStreamingScreen = function(session) {
  ...
  session.createStream(options).on(STREAM_STATUS.PUBLISHING,
function(screenStream) {
  /*
   * User can stop sharing screen capture using Chrome "stop" button.
   * Catch onended video track event and stop publishing.
   */
  document.getElementById(screenStream.id()).srcObject.getVideoTracks()
[0].onended = function (e) {
    screenStream.stop();
  };
  document.getElementById(screenStream.id()).addEventListener('resize',
function(event){
    resizeVideo(event.target);
  });
  setStatus("screen", STREAM_STATUS.PUBLISHING, screenStream);
}

```

```

    startStreamingCamera(session, screenStream);
  }).on(STREAM_STATUS.UNPUBLISHED, function() {
    ...
  }).on(STREAM_STATUS.FAILED, function(stream) {
    ...
  }).publish();
}

```

6. Web camera stream publishing

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

The following parameters are passed to `createStream()` method:

- `streamName` - stream name
- `localVideoCamera` - div tag to display a local video

```

const startStreamingCamera = function(session, screenStream) {
  let streamName = getStreamName("camera", field("url"));
  let options = {
    name: streamName,
    display: localVideoCamera
  }
  session.createStream(options).on(STREAM_STATUS.PUBLISHING,
function(cameraStream) {
    ...
  }).on(STREAM_STATUS.UNPUBLISHED, function() {
    ...
  }).on(STREAM_STATUS.FAILED, function(stream) {
    ...
  }).publish();
}

```

7. Receiving the event confirming successful camera publishing

`STREAM_STATUS.PUBLISHING` code

```

const startStreamingCamera = function(session, screenStream) {
  ...
  session.createStream(options).on(STREAM_STATUS.PUBLISHING,
function(cameraStream) {
    document.getElementById(cameraStream.id()).addEventListener('resize',
function(event){
      resizeVideo(event.target);
    });
    setStatus("camera", STREAM_STATUS.PUBLISHING, cameraStream);
    onStarted(cameraStream);
  }).on(STREAM_STATUS.UNPUBLISHED, function() {
    ...
  }).on(STREAM_STATUS.FAILED, function(stream) {
    ...
  });
}

```

```
}).publish();  
}
```

8. Stop camera publishing

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

```
const setPublishButton = function(action, session, cameraStream) {  
  $("#publishBtn").text(action).off('click').click(function(){  
    if (action == "Start") {  
      ...  
    } else if (action === "Stop") {  
      $(this).prop('disabled', true);  
      cameraStream.stop();  
    }  
  }).prop('disabled', false);  
}
```

9. Stop screen publishing

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

```
const startStreamingCamera = function(session, screenStream) {  
  ...  
  session.createStream(options).on(STREAM_STATUS.PUBLISHING,  
  function(cameraStream) {  
    ...  
  }).on(STREAM_STATUS.UNPUBLISHED, function() {  
    setStatus("camera", STREAM_STATUS.UNPUBLISHED);  
    screenStream.stop();  
  }).on(STREAM_STATUS.FAILED, function(stream) {  
    ...  
  }).publish();  
}
```