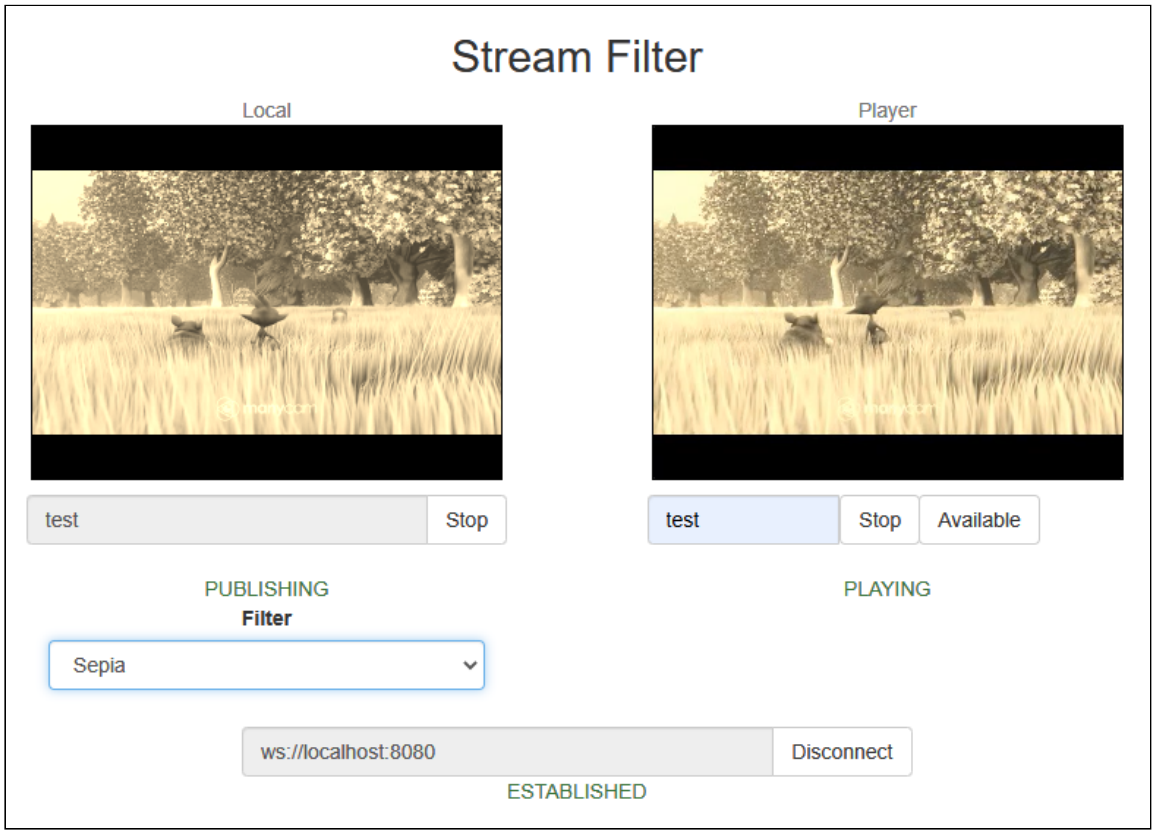
Stream Filter

Streamer example with picture filter application

This example shows how to apply a filter or another changes (beautification etc) to picture while publishing a stream using canvas element



This feature works in all the main browsers except iOS Safari 12

Code of the example

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

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

- stream_filter.css - styles file
- stream_filter.html - client page
- stream_filter.js - main script to work

The example can be tested by the following URL:

https://host:8888/client2/examples/demo/streaming/stream_filter/stream_filter.html

Where host - WCS server address.

Analyzing the code

To analyze the code take the file `stream_filter.js` version with hash `ecbadc3`, which is available here and can be downloaded with SDK build 2.0.212.

1. API initializing

Flashphoner.init() code

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

2. Connecting to the 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

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

4. Video streaming

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

When stream is created, the following parameters are passed

- `streamName` - name of the stream
- `localVideo` - `div` element, in which video from camera will be displayed

To apply a filter, the video captured from web camera will be drawn on the canvas using the option `useCanvasMediaStream: true`

```
session.createStream({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true
    receiveVideo: false,
    receiveAudio: false,
    useCanvasMediaStream: true
    ...
}).publish();
```

5. Receiving the event confirming successful streaming

`StreamStatusEvent PUBLISHING` code

The picture drawing on the canvas with FPS 30 is started by this event

```
session.createStream({
    ...
}).on(STREAM_STATUS.PUBLISHING, function(stream){
    setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
    onPublishing(stream);
    intervalId = setInterval(draw, 1000.0 / 30);
}).on(STREAM_STATUS.UNPUBLISHED, function(){
    ...
}).on(STREAM_STATUS.FAILED, function(){
    ...
}).publish();
```

6. Stream playback

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

When stream is created, the following parameters are passed

- `streamName` - name of the stream (including the stream published on step above)
- `remoteVideo` - `div` element, in which video playback will be displayed

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

7. Receiving the event confirming successful stream playback

`StreamStatusEvent PLAYING` code

```
session.createStream({
    name: streamName,
    display: remoteVideo
    ...
}).on(STREAM_STATUS.PLAYING, function(stream) {
    setStatus("#playStatus", stream.status());
    onPlaying(stream);
}).on(STREAM_STATUS.STOPPED, function() {
    ...
}).on(STREAM_STATUS.FAILED, function() {
    ...
}).play();
```

8. Stream playback stop

`Stream.stop()` code

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

9. Receiving the event confirming successful playback stop

`StreamStatusEvent STOPPED` code

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

10. Streaming stop

`Stream.stop()` code

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

11. Receiving the event confirming successful streaming stop

`StreamStatusEvent UNPUBLISHED` code

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

12. The picture drawing on the canvas and applying the filter

`draw()` code

```
function draw() {
    let localVideo = document.getElementById('localVideo');
    let canvas = localVideo.children[0];
    if (canvas) {
        let ctx = canvas.getContext('2d');
        // First need to draw video on the canvas
        ctx.drawImage(canvas.children[0], 0, 0);
        // next get image data
        let imageData = ctx.getImageData(0, 0, canvas.width, canvas.height);
        // next need to apply filter to the image
        let filtered = currentFilter(imageData);
        // and finally draw filtered image on the canvas
        ctx.putImageData(filtered, 0, 0);
    }
}
```

13. Filter list initializing and choosing the filter to apply

`applyFilter()` code

```
var filters = [empty, sepia, threshold, invert];
var currentFilter = empty;
...
```

```
function applyFilter() {
    let filter = $('#filter').val();
    currentFilter = filters[filter];
}

function empty(imageData) {
    return imageData;
}
```