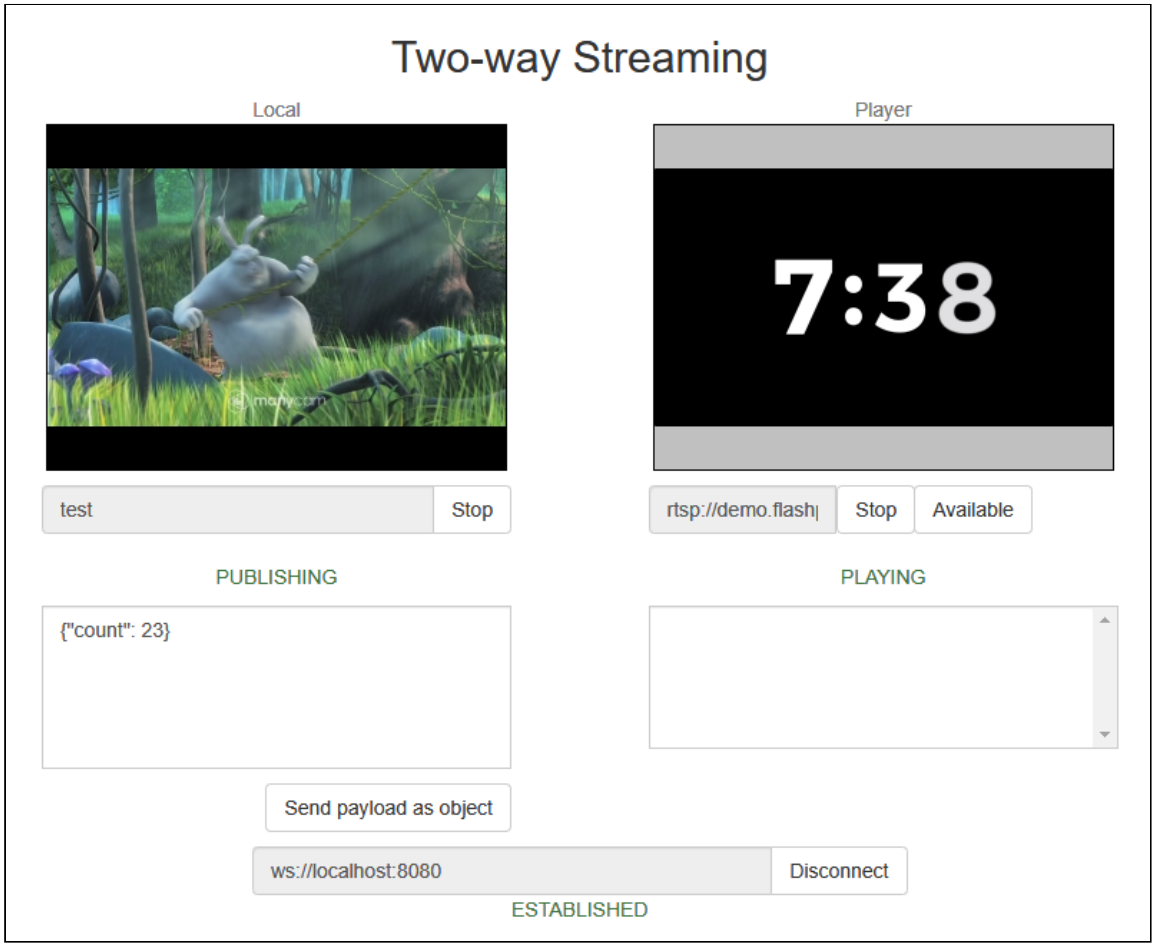
Two-way Streaming

Example with streamer and player on the same page

This example demonstrates how to publish a video stream while playing another one using the same web page.



Code of the example

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

/usr/local/FlashphonerWebCallServer/client/examples/demo/streaming/two_way_streaming

- two_way_streaming.css - file with styles
- two_way_streaming.html - page of the client
- two_way_streaming.js - script providing functionality for the example

This example can be tested using the following address:

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

Here host is the address of the WCS server.

Analyzing the code

To analyze the code, let's take the version of file `two_way_streaming.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

```
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` - element, in which video from camera will be displayed

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

5. Receiving the event confirming successful streaming

`StreamStatusEvent.PUBLISHING` code

```
session.createStream({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true
    ...
}).on(STREAM_STATUS.PUBLISHING, function(stream){
    setStatus("#publishStatus", STREAM_STATUS.PUBLISHING);
    onPublishing(stream);
}).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` - 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({
    name: streamName,
    display: remoteVideo
    ...
}).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({
    name: streamName,
    display: localVideo,
    cacheLocalResources: true
    ...
}).on(STREAM_STATUS.PUBLISHING, function(stream){
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function(){
    setStatus("#publishStatus", STREAM_STATUS.UNPUBLISHED);
    onUnpublished();
}).on(STREAM_STATUS.FAILED, function(){
    ...
}).publish();
```

12. Sendig data bound to the stream

`Stream.sendData()` code

```
function onPublishing(stream) {
    ...
    $('#sendDataBtn').off('click').click(function(){
        var streamData = field('streamData');
        stream.sendData(JSON.parse(streamData));
    }).prop('disabled',false);
}
```

13. Receiving data bound to the stream

`STREAM_EVENT`, `STREAM_EVENT_TYPE.DATA` code

```
    session.createStream({
        name: streamName,
        display: remoteVideo
    }).on(STREAM_STATUS.PENDING, function (stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function (stream) {
        ...
    }).on(STREAM_STATUS.STOPPED, function () {
        ...
    }).on(STREAM_STATUS.FAILED, function (stream) {
        ...
    }).on(STREAM_EVENT, function(streamEvent) {
        switch (streamEvent.type) {
```

```
            case STREAM_EVENT_TYPE.DATA:
                addPayload(streamEvent.payload);
                break;
        }
        console.log("Received streamEvent ", streamEvent.type);
    }).play();
```