Capturing VOD from a file

WCS offers possibility to capture a media stream from an MP4 file located on the local disk of the server (Video on Demand, VOD). The received stream can be played, republished, managed just like any stream on the WCS server. First of all, this option is intended to play previously recorded broadcasts in a browsers or a mobile application on the client side.

Overview

To capture VOD from a file, specify a link to the vod file as a stream name when calling the session.createStream() function, as follows:

```
vod://sample.mp4
```

where sample.mp4 is the name of the file that should be located in /usr/local/FlashphonerWebCallServer/media folder. Since build 5.2.687, a custom folder can specified with the following parameter in flashphoner.properties file

```
media_dir=/usr/local/FlashphonerWebCallServer/media
```

If a file with such name does not exist, the server returns the StreamStatusEvent FAILED message, where the "info" field has the reason: "File not found".

A stream created this way can be displayed to one user (personal VOD). Second viewer cannot subscribe to personal VOD stream, such stream cannot be transcoded, added to mixer or played by HLS.

If a full-featured online-broadcast is required, provide the link to a file as follows:

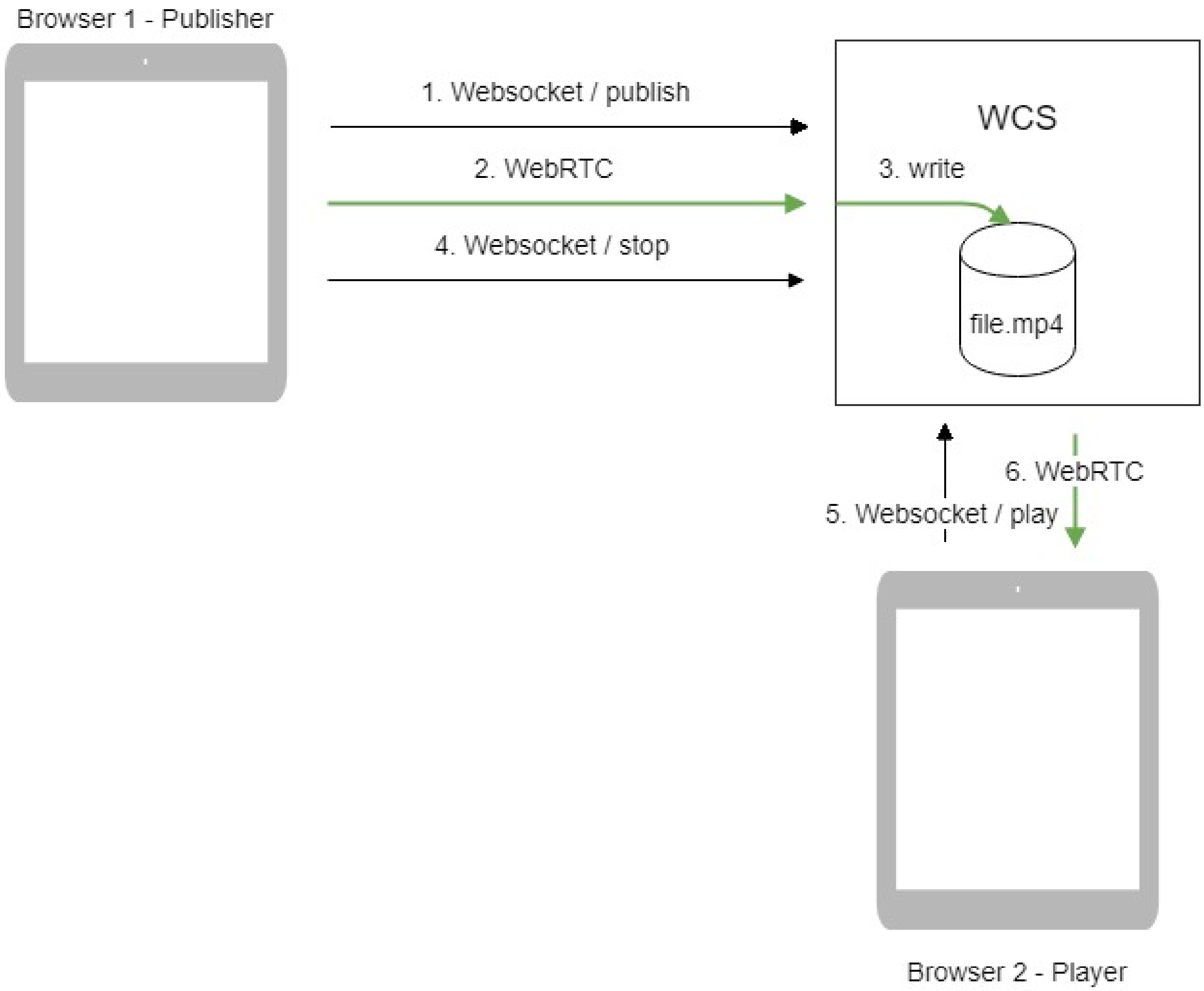
```
vod-live://sample.mp4
```

Multiple user can connect to such a stream simultaneously. VOD live stream can be transcoded, added to mixer or played by HLS.

Supported formats and codecs

- Container: MP4
- Video: H.264
- Audio: AAC

Operation flowchart



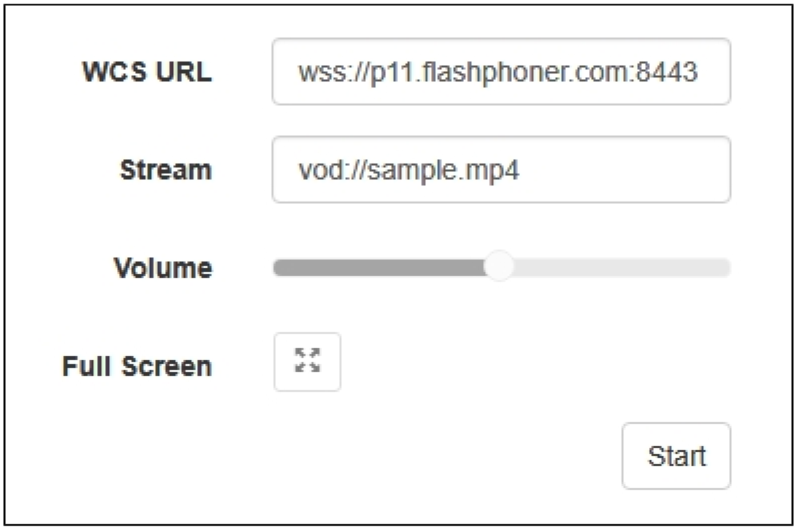
1. The browser connects to the server via WebSocket and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream as H.264 + AAC to the server, enabling recording with the parameter record: true.
3. The WCS server records the stream to a file.
4. The browser stops publishing.
5. The second browser establishes a connection via WebSocket, creates a stream, specifies the file name, and sends the play command.
6. The second browser receives the WebRTC stream and plays this stream on the page.

Quick manual on testing

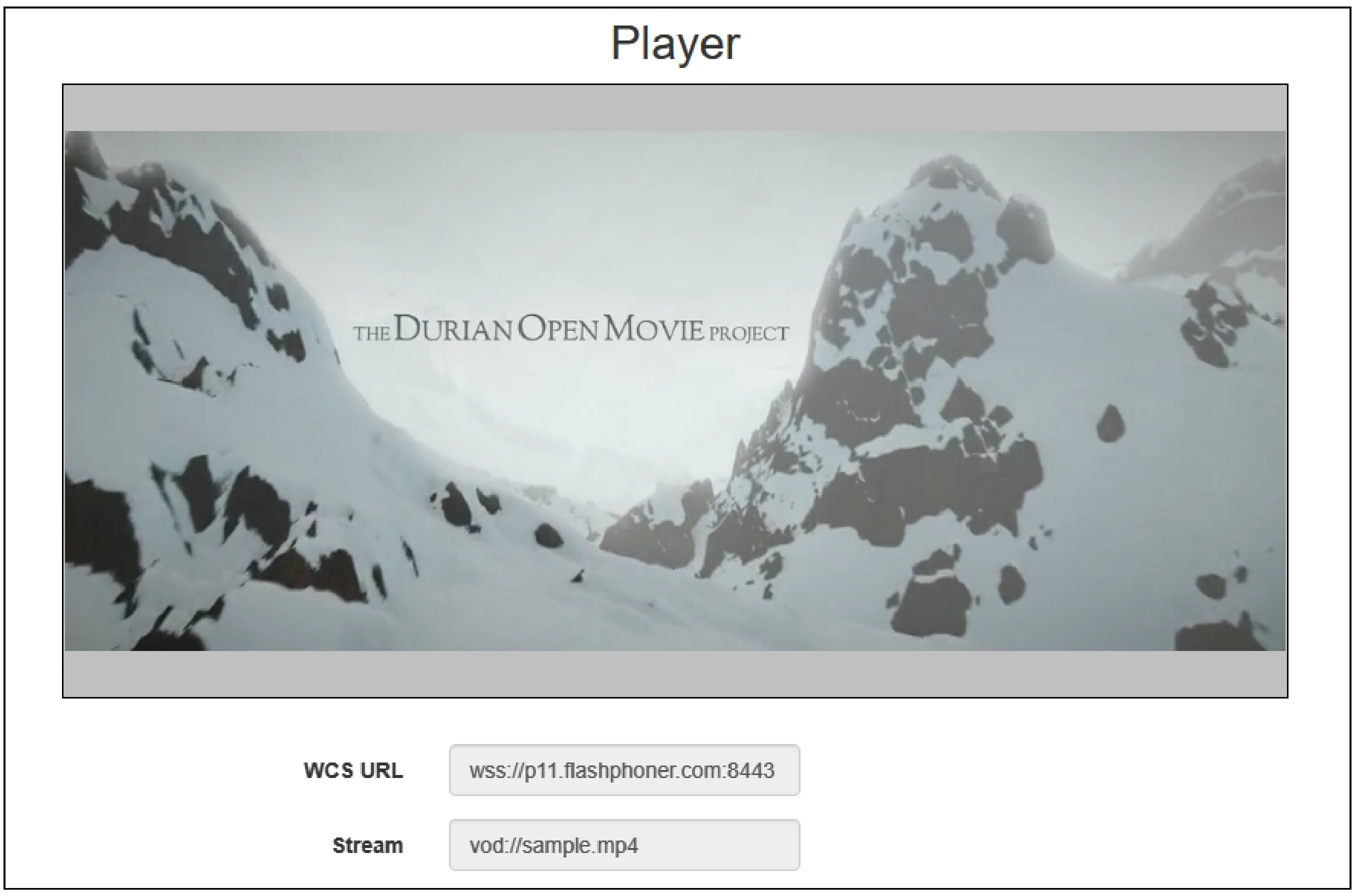
1. For the test we use thePlayerweb application to play the file.

2. Upload the file to the /usr/local/FlashphonerWebCallServer/media/ directory.

3. Open the Player web application and enter the name of the file in the Stream field:



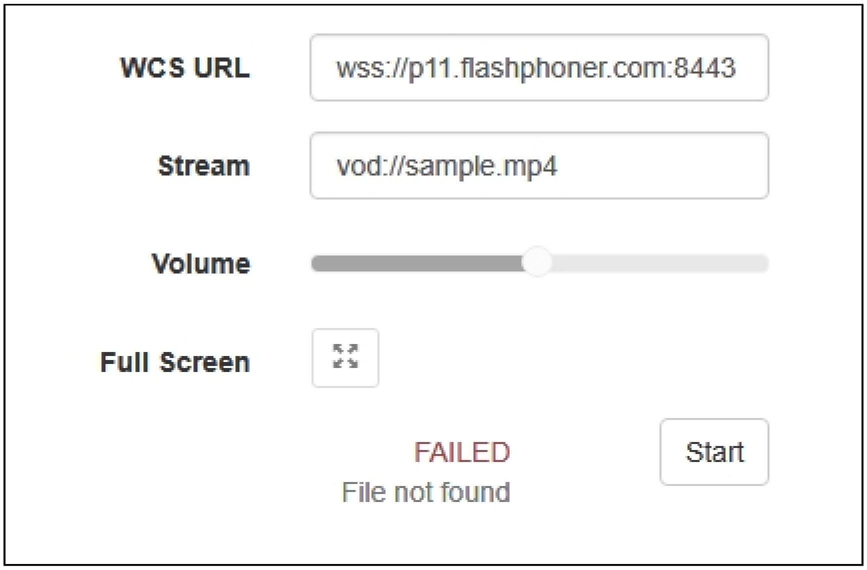
4. Click Start. The file starts playing:



5. Click Stop to stop the playback.

6. Delete the file from /usr/local/FlashphonerWebCallServer/media/

7. Click Start. You should see the FAILED status and the "File not found" message:



Call flow

Below is the call flow when using:

the Stream Recording example to publish the stream and record the file

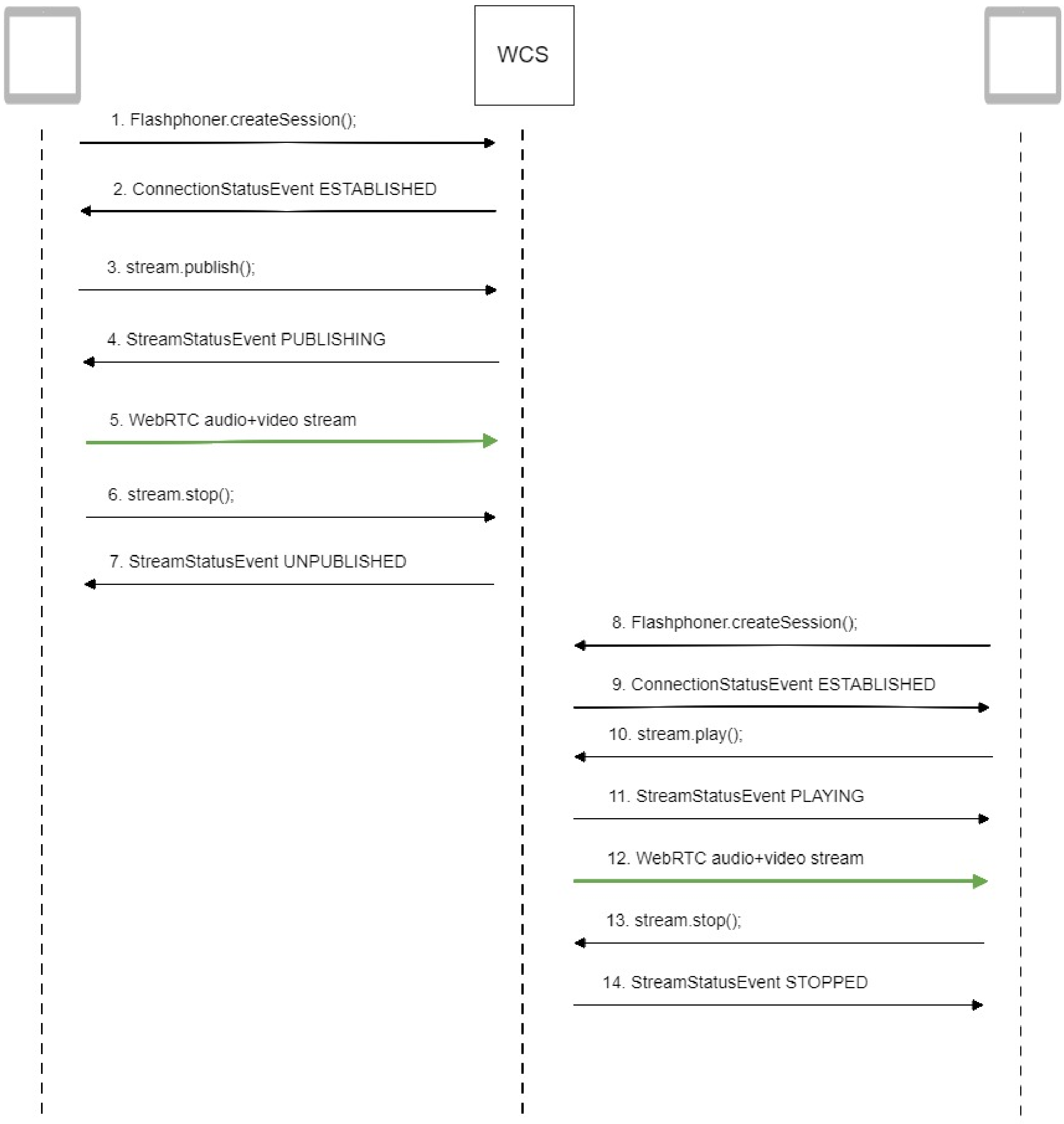
recording.html

recording.js

the Player example to play the VOD stream

player.html

player.js



1. Establishing a connection to the server to publish and record the stream.

Flashphoner.createSession();code

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

2. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

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

3. Publishing the stream with recording enabled.

stream.publish();code

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

4. Receiving from the server an event confirming successful publishing of the stream.

StreamStatusEvent, status PUBLISHINGcode

```
    session.createStream({
        name: streamName,
        display: localVideo,
        record: true,
        receiveVideo: false,
        receiveAudio: false
    }).on(STREAM_STATUS.PUBLISHING, function(stream) {
        setStatus(stream.status());
        onStarted(stream);
    }).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
        ...
    }).on(STREAM_STATUS.FAILED, function(stream) {
        ...
    }).publish();
```

5. Sending audio and video stream via WebRTC.

6. Stopping publishing the stream.

stream.stop();code

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

7. Receiving from the server an event confirming unpublishing of the stream.

StreamStatusEvent, status UNPUBLISHEDcode

```
session.createStream({
    name: streamName,
    display: localVideo,
    record: true,
    receiveVideo: false,
    receiveAudio: false
}).on(STREAM_STATUS.PUBLISHING, function(stream) {
    ...
}).on(STREAM_STATUS.UNPUBLISHED, function(stream) {
    setStatus(stream.status());
    showDownloadLink(stream.getRecordInfo());
    onStopped();
}).on(STREAM_STATUS.FAILED, function(stream) {
    ...
}).publish();
```

8. Establishing a connection to the server to play the stream.

Flashphoner.createSession();code

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

9. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

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

10. Playing the stream.

stream.play();code

```
if (Flashphoner.getMediaProviders()[0] === "MSE" && mseCutByIFrameOnly) {
    options.mediaConnectionConstraints = {
        cutByIFrameOnly: mseCutByIFrameOnly
    }
}
if (resolution_for_wsplayer) {
    options.playWidth = resolution_for_wsplayer.playWidth;
    options.playHeight = resolution_for_wsplayer.playHeight;
} else if (resolution) {
    options.playWidth = resolution.split("x")[0];
    options.playHeight = resolution.split("x")[1];
}
stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
    ...
});
stream.play();
```

11. Receiving from the server an event confirming successful playing of the stream.

StreamStatusEvent, status PLAYINGcode

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function(stream) {
        $("#preloader").show();
        setStatus(stream.status());
        onStarted(stream);
    }).on(STREAM_STATUS.STOPPED, function() {
        ...
    }).on(STREAM_STATUS.FAILED, function(stream) {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
        ...
    });
    stream.play();
```

12. Receiving of the audio-video stream via Websocket and playing it via WebRTC

13. Stopping publishing the stream.

stream.stop();code

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

14. Receiving from the server an event confirming successful stopping of the playback of the stream.

StreamStatusEvent, status STOPPEDcode

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    }).on(STREAM_STATUS.PLAYING, function(stream) {
        ...
    }).on(STREAM_STATUS.STOPPED, function() {
        setStatus(STREAM_STATUS.STOPPED);
        onStopped();
    }).on(STREAM_STATUS.FAILED, function(stream) {
        ...
    }).on(STREAM_STATUS.NOT_ENOUGH_BANDWIDTH, function(stream){
        ...
    });
    stream.play();
```

VOD loop

VOD live translation supports VOD loop: after end of file, capturing starts from file begin. This feature is enabled with the following parameter inflashphoner.propertiesfile

```
vod_live_loop=true
```

VOD capturing from AWS S3 or from other S3 compatible storage

VOD stream can be captured from file placed to AWS S3 storage. Comparing with VOD capture from local disk, file from external storage is downloaded and captured sequentally.

To capture VOD from AWS S3 file, specify a link to the vod file as a stream name when calling the session.createStream() function, as follows:

```
vod://s3/bucket/sample.mp4
```

where

- bucket is S3 bucket name
- sample.mp4 is file name

Since build5.2.939it is possible to set the full file URL in S3 storage, this allows to capture VOD from other S3 storages (Digital Ocean, Selectel etc)

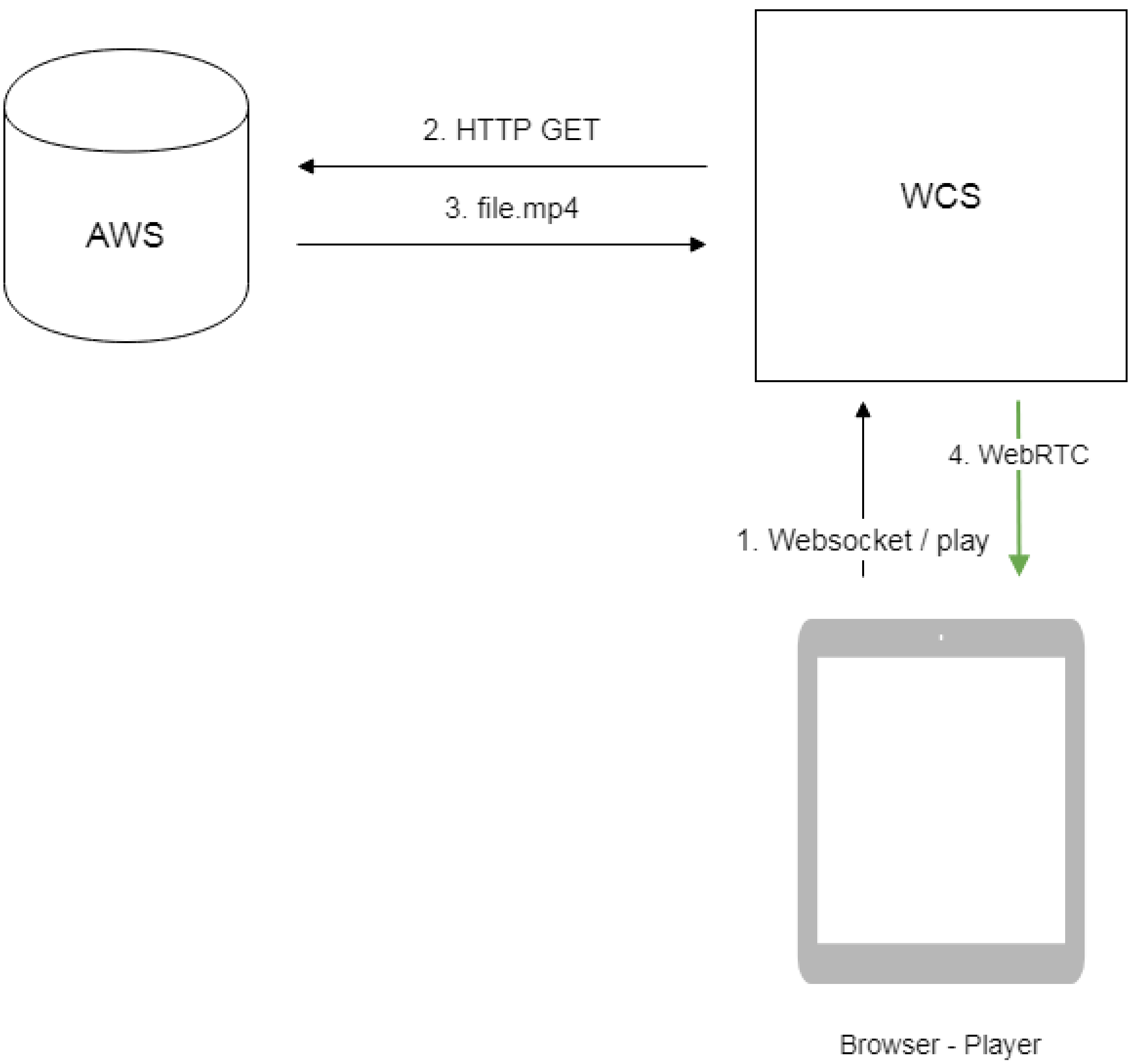
Digital Ocean Spaces URL example

```
vod://s3/https://ams3.digitaloceanspaces.com/myspace/folder/file.mp4
```

Selectel URL example

```
vod://s3/https://s3.selcdn.ru/mystorage/file.mp4
```

Operation flowchart



1. Browser requests VOD capture from AWS file

2. WCS server sends request to AWS

3. File is downloaded to WCS server

4. WebRTC stream from file is sending to browser for playback

Set up

S3 credentials configuration

AWS

To download files from AWS S3 bucket, S3 credentials must be set inflashphoner.propertiesfile

```
aws_s3_credentials=zone;login;hash
```

Where

- zone - AWS region where bucket is placed
- login - Access Key ID
- hash - Secret Accesss Key

S3 credentials setting example:

```
aws_s3_credentials=eu-central-1;AA22BB33CC44DE;DhlAkpZ4adclHhbLwhTNL4hvWTo80Njo
```

Digital Ocean Spaces

To download files from DO Spaces set the credentials as

```
aws_s3_credentials=ams3;access_key;secret
```

Where

- ams3 - digitaloceanspaces.com subdomain
- access_key - storage access key
- secret - storage access secret code

Selectel

To download files fromSelectel S3set the credentials as

```
aws_s3_credentials=ru-1a;login;password
```

Where

- ru-1a - storage region
- login - user name
- password - password

Capturing VOD stream from file while it is downloading

To capture stream from file while it is downloading, the following parameter should be set

```
vod_mp4_container_new=true
```

If channel bandwidth between WCS and S3 storage is low, or this channel is not stable enough, file bufferization may be enabled. The buffer size is set in milliseconds with the following parameter

```
vod_mp4_container_new_buffer_ms=10000
```

In this case, buffer size is 10 seconds.

File format requirements

Header section (moov) should always be before data section (mdat). File structure should be like this:

```
Atom ftyp @ 0 of size: 32, ends @ 32
Atom moov @ 32 of size: 357961, ends @ 357993
...
Atom free @ 357993 of size: 8, ends @ 358001
Atom mdat @ 358001 of size: 212741950, ends @ 213099951
```

File structure can be checked withAtomicParsleyutility

```
AtomicParsley file.mp4 -T 1
```

If the file structure does not match the requiremets, this file will not be played. Wrong file structure can be fixed if necessary with ffmpeg without reencoding

```
ffmpeg -i bad.mp4 -acodec copy -vcodec copy -movflags +faststart good.mp4
```

File name requirements

Official AWS S3documentationdoes not recommend to use spaces along another special characters, but does not prohibits them. If the file name contains spaces, they should be replaced by '%20', for example

```
vod://s3/bucket/sample%20with%20spaces.mp4
```

VOD capture management with REST API

REST query should be HTTP/HTTPS POST request as:

- HTTP:http://test.flashphoner.com:8081/rest-api/vod/startup
- HTTPS:https://test.flashphoner.com:8444/rest-api/vod/startup

Where:

- test.flashphoner.com - WCS server address
- 8081 - standard REST / HTTP port
- 8444 - standard HTTPS port
- rest-api - mandatory part of URL
- /vod/startup - REST method used

REST queries and responses

REST query	REST query example	REST response example	Response states	Description
/vod/startup	`{ "uri":"vod-live://sample.mp4", "localStreamName": "test" }`		409 - Conflict 500 - Internal error	Capture VOD stream from file

<table>
<tr>
<td>/vod/find</td>
<td><pre>{
 "localStreamName":
"test"
}</pre></td>
<td><pre>[
    {
        "localMediaSessionId": "29ec3236-1093-
42bb-88d6-d4ac37af3ac0",
        "localStreamName": "test",
        "uri": "vod-live://sample.mp4",
        "status": "PROCESSED_LOCAL",
        "hasAudio": true,
        "hasVideo": true,
        "record": false,
        "loop": false
    }
]</pre></td>
<td>200 – OK

404 – not found</td>
<td>Find VOD streams by criteria</td>
</tr>
<tr>
<td>/vod
/find_all</td>
<td></td>
<td><pre>[
    {
        "localMediaSessionId": "29ec3236-1093-
42bb-88d6-d4ac37af3ac0",
        "localStreamName": "test",
        "uri": "vod-live://sample.mp4",
        "status": "PROCESSED_LOCAL",
        "hasAudio": true,
        "hasVideo": true,
        "record": false,
        "loop": false
    }
]</pre></td>
<td>200 – OK

404 – not found</td>
<td>Find all VOD streams</td>
</tr>
<tr>
<td>/vod
/terminate</td>
<td><pre>{
 "uri":"vod://sample.
mp4",
 "localStreamName":
"test"
}</pre></td>
<td></td>
<td>200 - Stream is stopped

404 - Stream not found</td>
<td>Stop VOD stream</td>
</tr>
</table>

Parameters

Name	Description	Example
uri	File name to capture	`vod://sample.mp4`
localStreamName	Stream name	`test`
status	Stream status	`PROCESSED_LOCAL`
localMediaSessionId	Mediasession Id	`29ec3236-1093-42bb-88d6-d4ac37af3ac0`
hasAudio	Stream has audio	`true`
hasVideo	Stream has video	`true`
record	Stream is recording	`false`
loop	VOD is looped	`false`

VOD looping on demand

Since build 5.2.1528 it is possible to enable VOD looping while creating VOD live translation via REST API

```
{
 "uri":"vod-live://sample.mp4",
 "localStreamName": "test",
 "loop": true
}
```

By default, if `loop` parameter is not set, `vod_live_loop` is applied. If the parameter is set, its value is applied as follows

- `true` - file will be looped
- `false` - file will be played once, then VOD live translation will stop

The `loop` parameter has a precedence over `vod_live_loop` value.

Known limits

`/rest-api/vod/startup` query can be used for VOD live translations creation only. However, `find`, `find_all `and `terminate` queries can be applied both to VOD and VOD live translations.

VOD stream publishing timeout after all subscribers gone off

By default, VOD stream stays published on server during 30 seconds after last subscriber gone off, if file duration exceeds this interval. This timeout can be changed with the following parameter

```
vod_stream_timeout=60000
```

In this case, VOD stream stays published during 60 seconds.

Known issues

1. AAC frames of type 0 are not supported by ffmpeg decoder and will be ignored while stream pulled playback

Symptoms: warnings in theclient log:

```
10:13:06,815 WARN AAC - AudioProcessor-c6c22de8-a129-43b2-bf67-1f433a814ba9 Dropping AAC frame that starts with 0, 119056e500
```

Solution:switch to FDK AAC decoder

```
use_fdk_aac=true
```

2. Files with B-frames can be played unsmoothly, with artifacts and freezes

Symptoms: periodic freezes and artifacts while playing VOD file, warnongs in the client log

```
09:32:31,238 WARN 4BitstreamNormalizer - RTMP-pool-10-thread-5 It is B-frame!
```

Solution: reencode this file to exclude B-frames, for example

```
ffmpeg -i bad.mp4 -preset ultrafast -acodec copy -vcodec h264 -g 24 -bf 0 good.mp4
```

3. When VOD is captured from a long-duration file, or a number of VOD streams are captured simultaneously, server process can terminate with Out of memory

Symptoms: server process terminates; "Map failed" inserver logand in error*.log

```
19:30:53,277 ERROR DefaultMp4SampleList - Thread-34 java.io.IOException: Map failed
    at sun.nio.ch.FileChannelImpl.map(FileChannelImpl.java:940)
    at com.googlecode.mp4parser.FileDataSourceImpl.map(FileDataSourceImpl.java:62)
    at com.googlecode.mp4parser.BasicContainer.getByteBuffer(BasicContainer.java:223)
    at com.googlecode.mp4parser.authoring.samples.DefaultMp4SampleList$SampleImpl.asByteBuffer
(DefaultMp4SampleList.java:204)
    at com.flashphoner.media.F.A.A.A$1.A(Unknown Source)
    at com.flashphoner.media.M.B.C.D(Unknown Source)
    at com.flashphoner.server.C.A.B.A(Unknown Source)
    at com.flashphoner.server.C.A.B.C(Unknown Source)
    at java.lang.Thread.run(Thread.java:748)
Caused by: java.lang.OutOfMemoryError: Map failed
    at sun.nio.ch.FileChannelImpl.map0(Native Method)
    at sun.nio.ch.FileChannelImpl.map(FileChannelImpl.java:937)
    ... 8 more
```

```
Event: 1743.157 Thread 0x00007fc480375000 Exception <a 'java/lang/OutOfMemoryError': Map failed>
(0x00000000a1d750b0) thrown at [/HUDSON/workspace/8-2-build-linux-amd64/jdk8u161/10277/hotspot/src/share/vm
/prims/jni.cpp, line 735]
```

Solution:

1. Increase maximum number of regions of virtual memory

```
sysctl -w vm.max_map_count=262144
```

and virtual memory amount allocated to server process by changing in /usr/local/FlashphonerWebCallServer/bin/webcallserver file the string

```
ulimit -n 20000
```

to the strings

```
ulimit -n 20000
ulimit -v 1000000000
```

2. Starting from build 5.2.57, set the following parameter

```
vod_mp4_container_isoparser_heap_datasource=true
```