Захват VOD из файла

- Описание
  - Поддерживаемые форматы и кодеки
  - Схема работы
- Краткое руководство по тестированию
- Последовательность выполнения операций
- Циклический захват потока из файла
- Захват файла, размещенного на AWS
  - Схема работы
  - Настройка
  - Требования к формату файлов
  - REST-методы и статусы ответа
    - Параметры
  - Ограничения
- Настройка продолжительности публикации VOD потока после отключения подписчиков
- Известные проблемы

WCS предоставляет возможность захвата медиапотока из файла MP4, расположенного на локальном диске сервера (Video on Demand, VOD). Полученный поток можно воспроизвести, ретранслировать, управлять им, как любым потоком на WCS-сервере. Прежде всего, данная возможность предназначена для воспроизведения записанных ранее трансляций в браузере или мобильном приложении клиента.

Описание

Для захвата VOD из файла в качестве имени потока при вызове функции session.createStream() должна быть указана ссылка на файл в виде:

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

где sample.mp4 - имя файла, который должен находиться в каталоге /usr/local/FlashphonerWebCallServer/media/

В случае, если файл с таким именем отсутствует, сервер вернет сообщение StreamStatusEvent FAILED, в поле "info" которого будет указан диагноз "File not found".

Поток, созданный таким образом, предназначен для трансляции одному пользователю (персональный VOD). В случае, если необходимо организовать полноценную онлайн-трансляцию, следует указать ссылку на файл в виде:

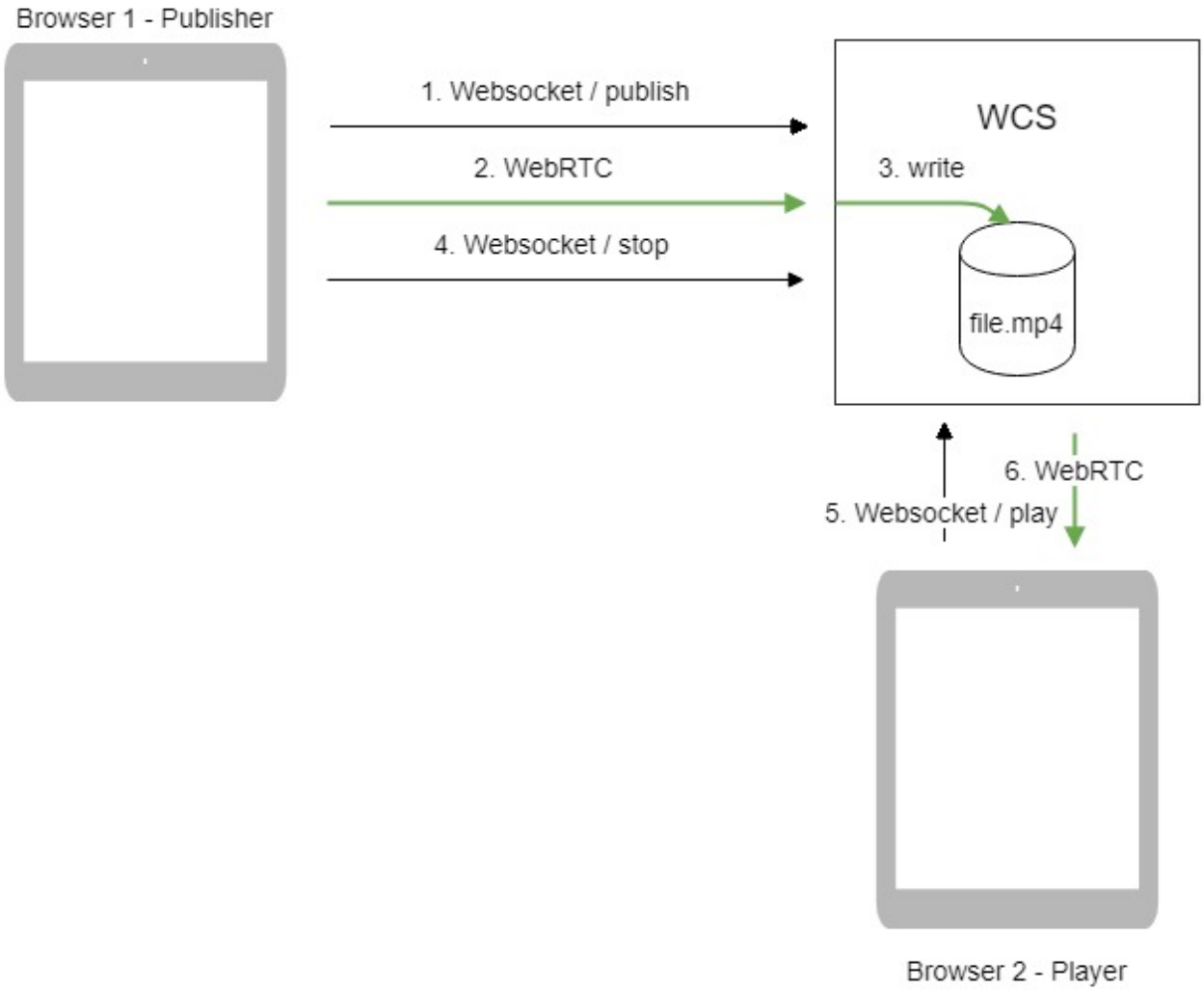
```
vod-live://sample.mp4
```

К такому потоку могут подключиться одновременно несколько пользователей в реальном времени.

Поддерживаемые форматы и кодеки

- Контейнер: MP4
- Видео: H.264
- Аудио: AAC

Схема работы



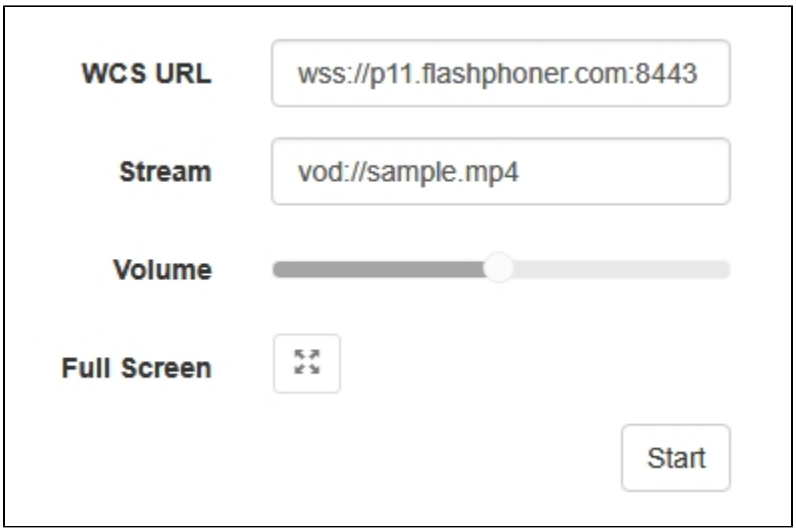
1. Браузер соединяется с сервером по протоколу Websocket и отправляет команду publish.
2. Браузер захватывает микрофон и камеру и отправляет WebRTC поток H.264 + AAC на сервер с параметром record: true.
3. WCS-сервер записывает поток в файл.
4. Браузер останавливает публикацию.
5. Второй браузер устанавливает соединение по Websocket, создает поток с указанием имени файла и отправляет команду play.
6. Второй браузер получает WebRTC поток и воспроизводит этот поток на странице.

Краткое руководство по тестированию

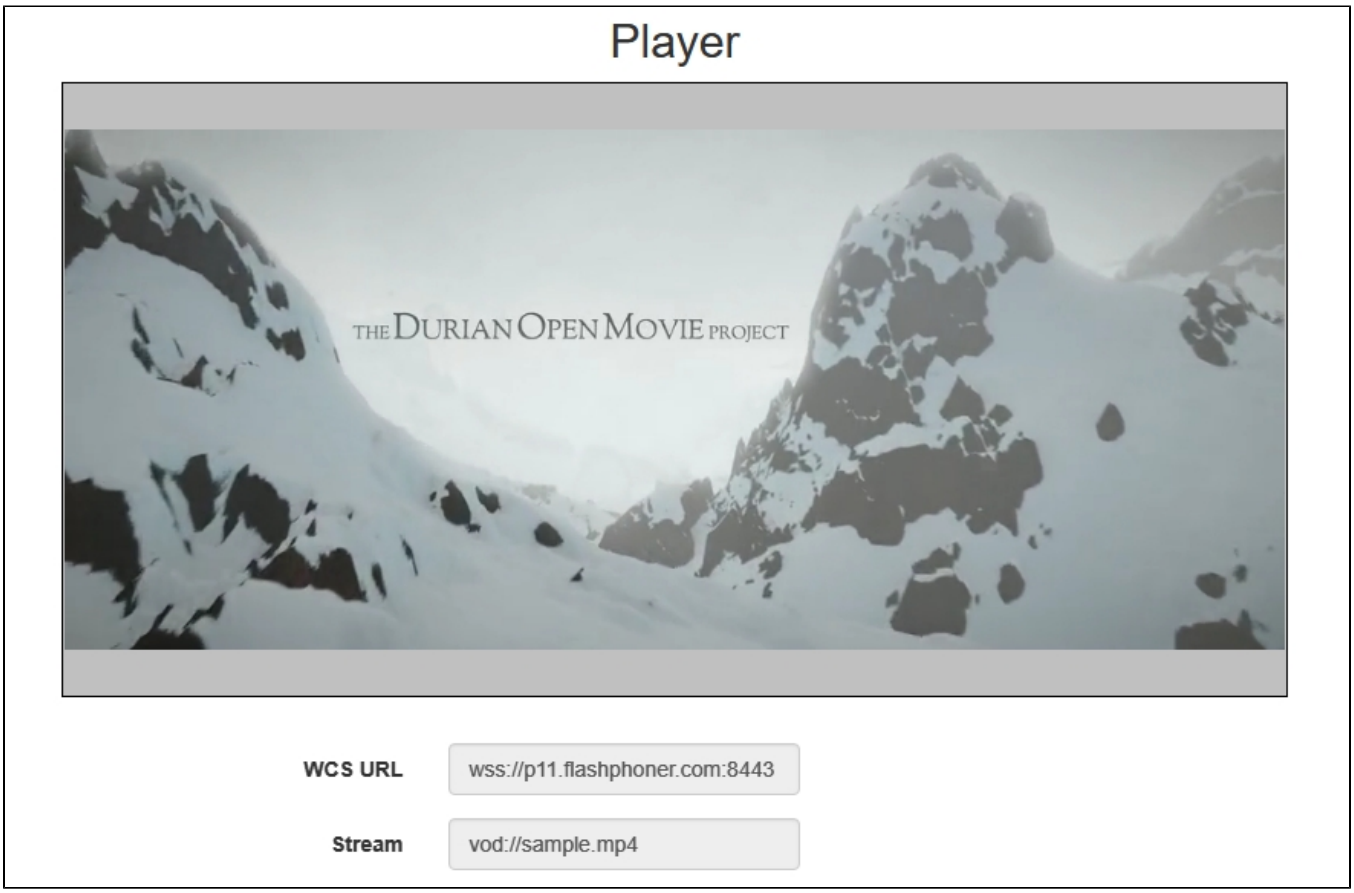
1. Для теста используем веб-приложениеPlayerдля воспроизведения файла.

2. Загрузите файл в каталог /usr/local/FlashphonerWebCallServer/media/

3. Откройте веб-приложение Player, укажите в поле Stream имя файла:



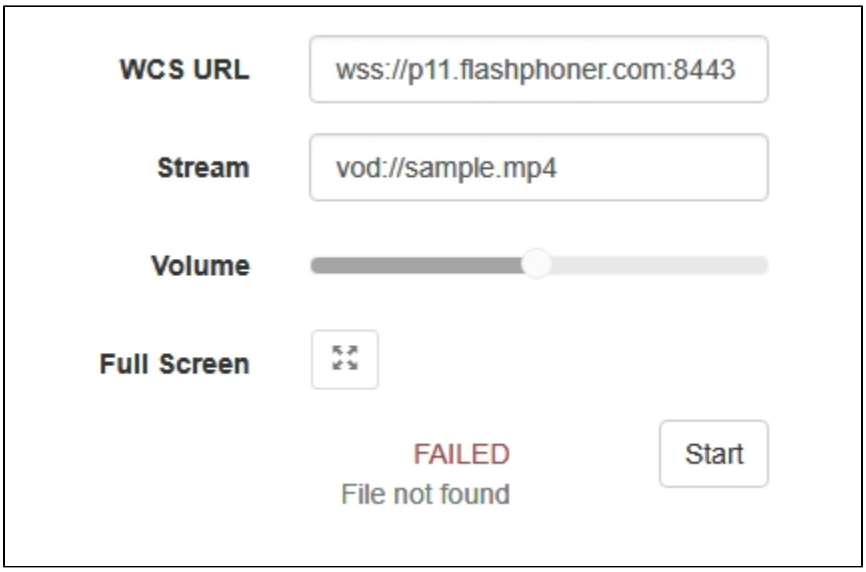
4. Нажмите Start. Начнется воспроизведение файла:



5. Нажмите Stop для остановки воспроизведения.

6. Удалите файл из каталог /usr/local/FlashphonerWebCallServer/media/

7. Нажмите Start. Отобразится статус FAILED и сообщение "File not found":



Последовательность выполнения операций

Ниже описана последовательность вызовов при использовании:

примера Stream Recording для публикации потока и записи файла

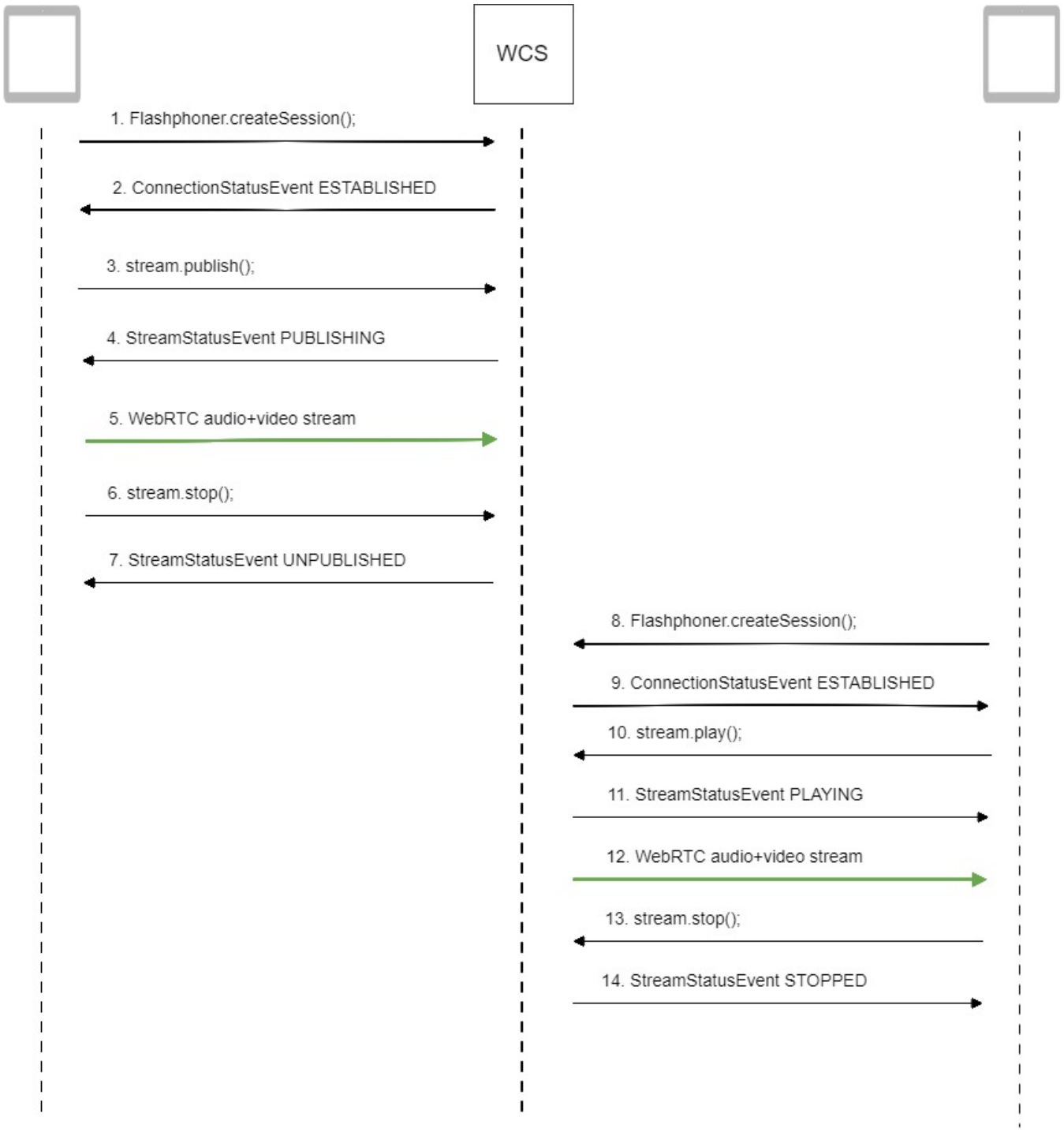
recording.html

recording.js

примера Player для воспроизведения VOD-потока

player.html

player.js



1. Установка соединения с сервером для публикации и записи потока.

Flashphoner.createSession();code

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

2. Получение от сервера события, подтверждающего успешное соединение.

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. Публикация потока с указанием признака записи:

stream.publish();code

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

4. Получение от сервера события, подтверждающего успешную публикацию потока.

StreamStatusEvent, статус 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. Отправка аудио-видео потока по WebRTC

6. Остановка публикации потока.

stream.stop();code

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

7. Получение от сервера события, подтверждающего остановку публикации потока.

StreamStatusEvent, статус 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. Установка соединения с сервером для воспроизведения потока.

Flashphoner.createSession();code

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

9. Получение от сервера события, подтверждающего успешное соединение.

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. Воспроизведение потока.

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. Получение от сервера события, подтверждающего успешное воспроизведение потока.

StreamStatusEvent, статус 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. Прием аудио-видео потока по Websocket и воспроизведение по WebRTC

13. Остановка воспроизведения потока.

stream.stop();code

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

14. Получение от сервера события, подтверждающего остановку воспроизведения потока.

StreamStatusEvent, статус 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();
```

Циклический захват потока из файла

Для трансляций vod-live поддерживается циклический захват потока, после окончания файла захват начинается сначала. Эта возможность включается настройкойв файлеflashphoner.properties

```
vod_live_loop=true
```

Захват файла, размещенного на AWS

Поток может быть захвачен из файла, размещенного на AWS в хранилище S3. В отличие от VOD захвата файла с локального диска, файл, размещенный на внешнем хранилище, загружается и воспроизводится последовательно.

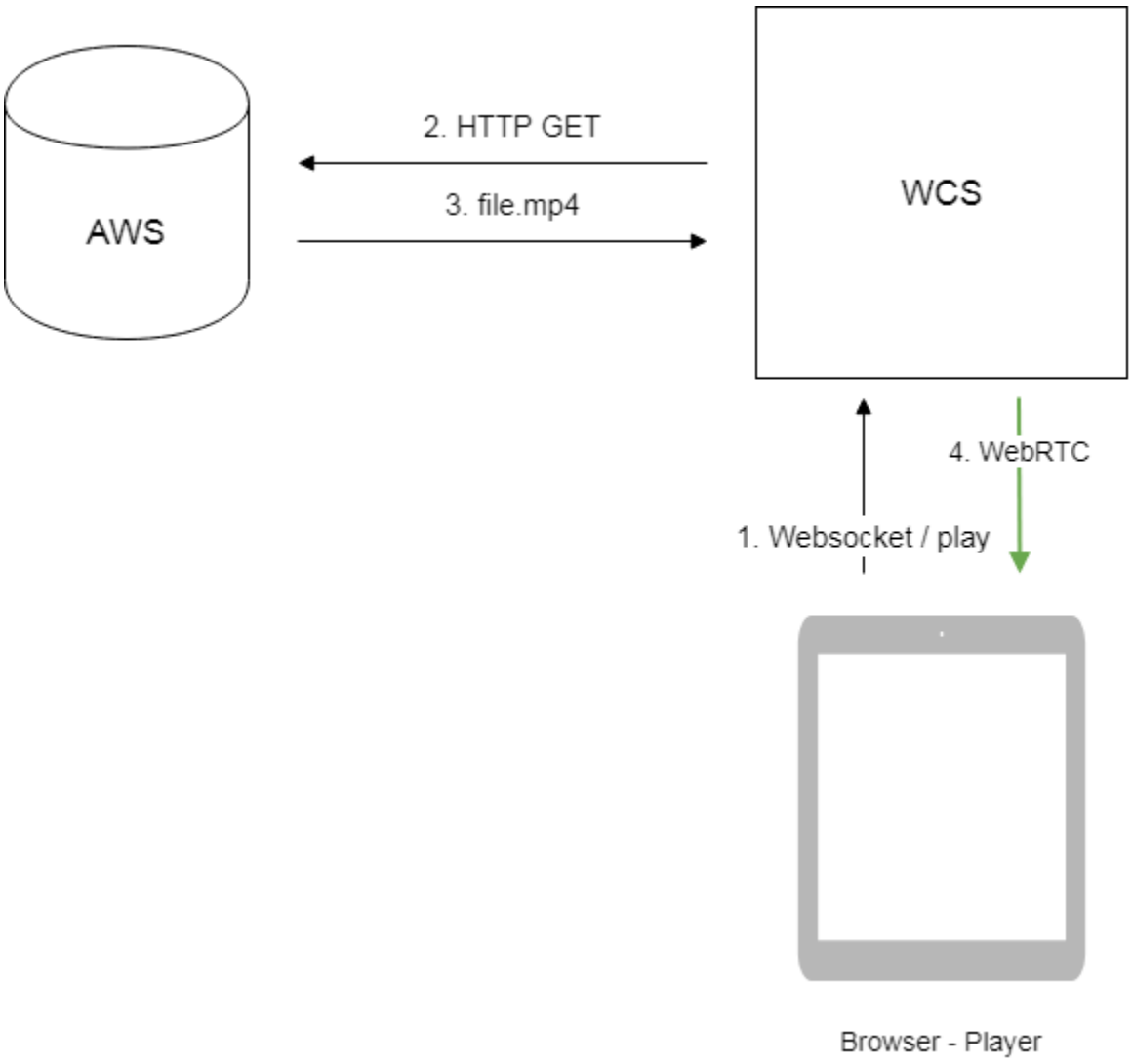
Для захвата VOD из файла на AWS в качестве имени потока при вызове функции session.createStream() должна быть указана ссылка на файл в виде:

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

где

- bucket - имя корзины S3
- sample.mp4 - имя файла

Схема работы



1. Браузер запрашивает захват потока из файла на AWS

2. WCS сервер направляет запрос AWS

3. Файл загружается на WCS сервер

4. WebRTC поток из файла передается в браузер для воспроизведения

Настройка

Для загрузки файлов из AWS необходимо указать в файле настроек flashphoner.properties данные для доступа к хранилищу S3

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

Чтобы захватывать поток из файла во время его загрузки, необходимо указать следующую настройку

```
vod_mp4_container_new=true
```

Требования к формату файлов

Заголовок (moov) должен всегда располагаться перед данными (mdat). Примерная структура файла должна быть такой:

```
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
```

Проверить структуру файла можно при помощи утилиты AtomicParsley

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

При необходимости, структуру файла можно исправить при помощи ffmpeg без перекодирования

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

Управление VOD при помощи REST API

REST-запрос должен быть HTTP/HTTPS POST запросом в таком виде:

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

Здесь:

- test.flashphoner.com - адрес WCS-сервера
- 8081 - стандартный REST / HTTP порт WCS-сервера
- 8444 - стандартный HTTPS порт
- rest-api - обязательная часть URL
- /vod/startup - используемый REST-метод

REST-методы и статусы ответа

REST-метод	Пример тела REST-запроса	Пример тела REST-ответа	Статусы ответа	Описание
/vod /startup	`{ "uri":"vod-live://sample.mp4", "localStreamName": "test" }`		409 - Conflict 500 - Internal error	Захватить поток из указанного файла

/vod/find	<pre>{
	
	"localStreamName":
	"test"
	}</pre>
		<pre>[
		    {
		        "localMediaSessionId": "29ec3236-
		1093-42bb-88d6-d4ac37af3ac0",
		        "localStreamName": "test",
		        "uri": "vod-live://sample.mp4",
		        "status": "PROCESSED_LOCAL",
		        "hasAudio": true,
		        "hasVideo": true,
		        "record": false
		    }
		]</pre>
			200 – потоки найдены
			
			404 – потоки не найдены
				Найти VOD-потоки по указанному критерию

/vod
/find_all
		<pre>[
		    {
		        "localMediaSessionId": "29ec3236-
		1093-42bb-88d6-d4ac37af3ac0",
		        "localStreamName": "test",
		        "uri": "vod-live://sample.mp4",
		        "status": "PROCESSED_LOCAL",
		        "hasAudio": true,
		        "hasVideo": true,
		        "record": false
		    }
		]</pre>
			200 – потоки найдены
			
			404 – потоки не найдены
				Найти все VOD-потоки

/vod
/terminate
	<pre>{
	  "uri":"
	vod://sample.mp4",
	
	"localStreamName":
	"test"
	}</pre>
			200 - поток завершен
			
			404 - поток не найден
				Завершить VOD-поток

Параметры

Имя параметра	Описание	Пример
uri	Имя файла для захвата потока	`vod://sample.mp4`
localStreamName	Имя создаваемого потока	`test`
status	Текущий статус потока	`PROCESSED_LOCAL`
localMediaSessionId	Идентификатор медиасессии	`29ec3236-1093-42bb-88d6-d4ac37af3ac0`
hasAudio	В потоке есть аудио	`true`
hasVideo	В потоке есть видео	`true`
record	Поток записывается	`false`

Ограничения

Запрос /rest-api/vod/startup может применяться только для создания VOD live трансляций. При этом, запросы find, find_all и terminate могут быть применены как к VOD, так и к VOD live трансляциям.

Настройка продолжительности публикации VOD потока после отключения подписчиков

По умолчанию, VOD поток остается опубликованным на сервере в течение 30 секунд после отключения последнего подписчика, при условии, что продолжительность файла превышает этот интервал. Данное время может быть изменено при помощи настройки

```
vod_stream_timeout=60000
```

В этом случае, VOD поток останется опубликованным в течение 60 секунд.

Известные проблемы

1. AAC фреймы типа 0 не поддерживаются декодером на базе ffmpeg и будут игнорироваться при воспроизведении захваченного потока

Симптомы: предупреждения вклиентском логе:

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

Решение: переключиться на использование FDK AAC декодера

```
use_fdk_aac=true
```

2. Файлы, содержащие B-фреймы, могут проигрываться неплавно, с фризами или артефактами

Симптомы: периодические фризы, артефакты при проигрывании файла через VOD, предупреждения в клиентском логе

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

Решение: перекодировать файл таким образом, чтобы исключить B-фреймы, например

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

3. При захвате VOD из продолжительного файла процесс сервера может завершиться с Out of memory при превышении максимального числа областей виртуальной памяти (vm.max_map_count)

Симптомы: процесс сервера завершается; "Map failed" в серверном логе и в 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]
```

Решение: увеличить максимальное число областей виртуальной памяти

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