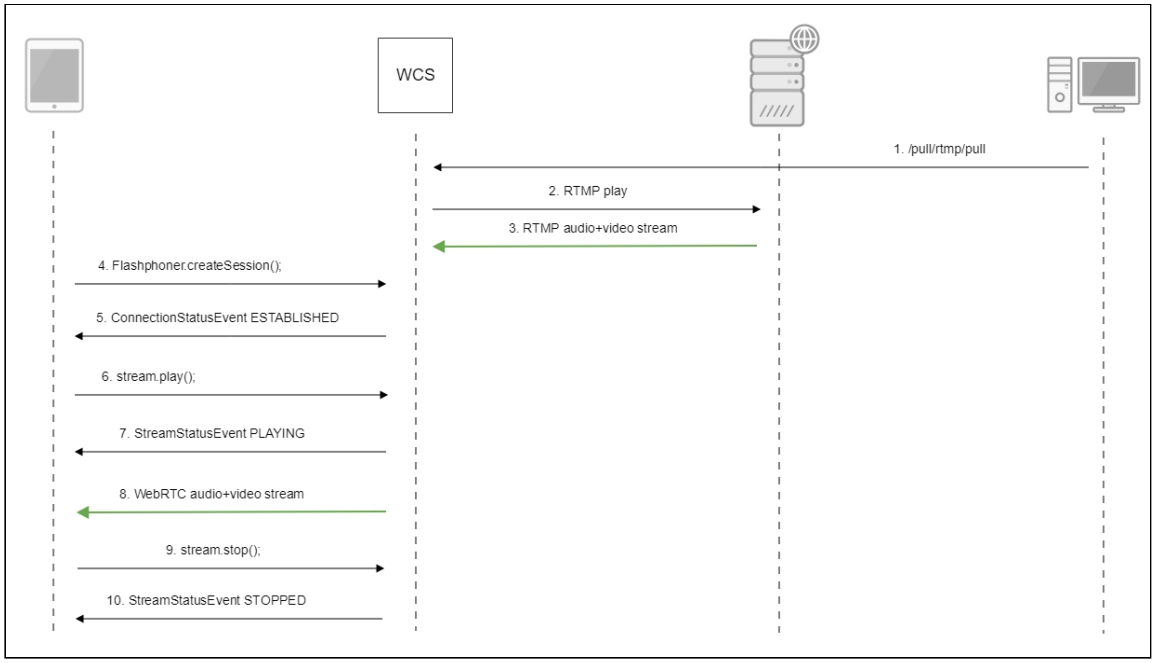
С другого WCS сервера по WebRTC

Описание

WCS может по требованию захватывать WebRTC-видеопоток, раздаваемый с другого WCS-сервера. Захваченный поток может раздаваться на любые из поддерживаемых платформ, по любой из поддерживаемых технологий. Для управления захватом WebRTC-потока используется REST API.

Схема работы



1. Браузер соединяется с сервером WCS1 по протоколу Websocket и отправляет команду `publishStream`.
2. Браузер захватывает микрофон и камеру и отправляет WebRTC поток на сервер.
3. REST-клиент отправляет на сервер WCS2 запрос `/pull/pull`.
4. WCS2 запрашивает поток с WCS1.
5. WCS2 получает WebRTC поток с WCS1.
6. Второй браузер устанавливает соединение с сервером WCS2 по Websocket и отправляет команду `playStream`.
7. Второй браузер получает WebRTC поток и воспроизводит этот поток на странице.

REST API

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

- HTTP: `http://test.flashphoner.com:8081/rest-api/pull/pull`
- HTTPS: `https://test.flashphoner.com:8444/rest-api/pull/pull`

Здесь:

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

REST-методы и статусы ответа

REST метод	Тело запроса	Тело ответа	Статусы ответа	Описание
`/pull/pull`	`{ "uri":"wss://demo.flashphoner.com:8443", "localStreamName": "testStream", "remoteStreamName": "testStream" }`		409 Conflict 500 Internal error	Извлечь WebRTC-поток по указанному URL

REST метод	Тело запроса	Тело ответа	Статусы ответа	Описание
`/pull/find_all`		[{ "localMediaSessionId": "5a072377-73c1-4caf-abd3", "remoteMediaSessionId": null, "localStreamName": "testStream", "remoteStreamName": "testStream", "uri": "wss://demo.flashphoner.com:8443", "status": "NEW" }]	200 OK 404 Not found 500 Internal error	Найти все извлеченные WebRTC-потоки

REST метод	Тело запроса	Тело ответа	Статусы ответа	Описание
`/pull/terminate`	{ "uri":"wss://demo.flashphoner.com:8443", "localStreamName": "testStream", "remoteStreamName": "testStream" }		200 OK 404 Not found 500 Internal error	Завершить извлеченный WebRTC-поток

Параметры

Parameter	Description	Example
uri	Websocket URL WCS-сервера	`wss://demo.flashphoner.com:8443`
localMediaSessionId	Идентификатор медиасессии	`5a072377-73c1-4caf-abd3`
remoteMediaSessionId	Идентификатор медиасессии на другом сервере	`12345678-abcd-dead-beaf`
localStreamName	Локальное имя, присвоенное захваченному потоку. По данному имени поток может быть запрошен с WCS сервера	`testStream`
remoteStreamName	Имя захватываемого потока на другом сервере	`testStream`
status	Текущий статус потока	`NEW`

Настройка

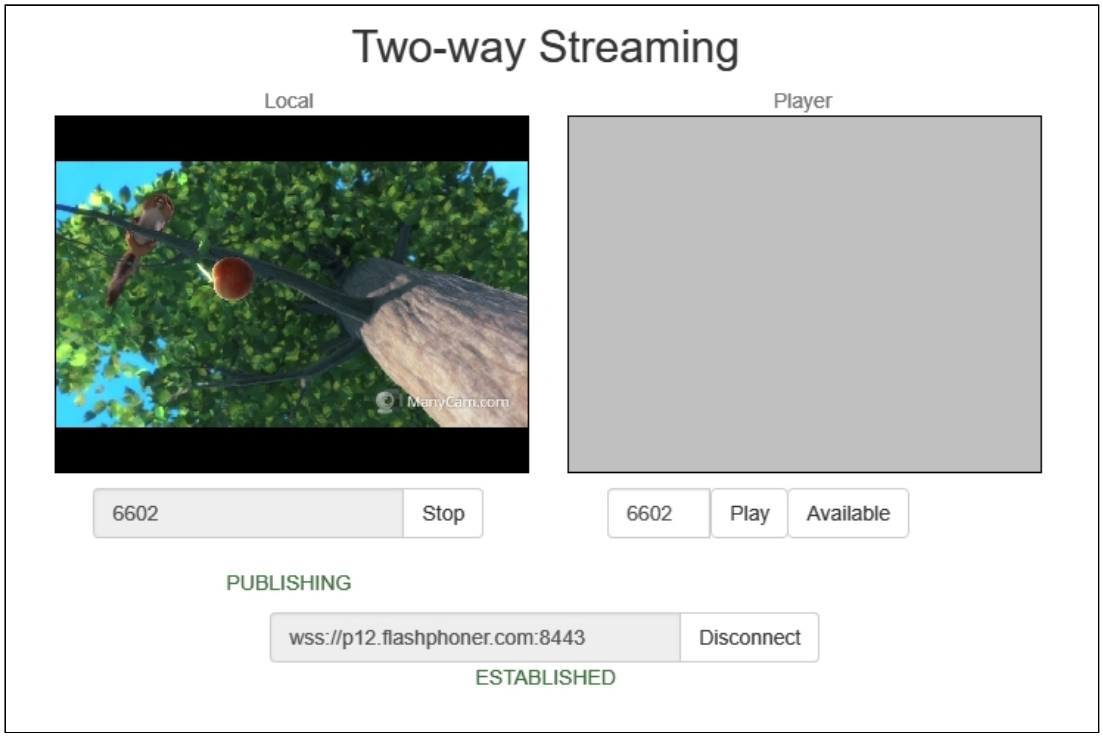
По умолчанию, захват потока производится по незащищенному соединению, т.е. URL WCS-сервера должен задаваться в виде `ws://demo.flashphoner.com:8080`. Чтобы использовать Secure Websocket, необходимо в файле настроек flashphoner.properties указать параметр

```
wcs_agent_ssl=true
```

Изменения в настройку должны быть внесены на обоих WCS-серверах: том, который публикует поток, и том, который этот поток захватывает.

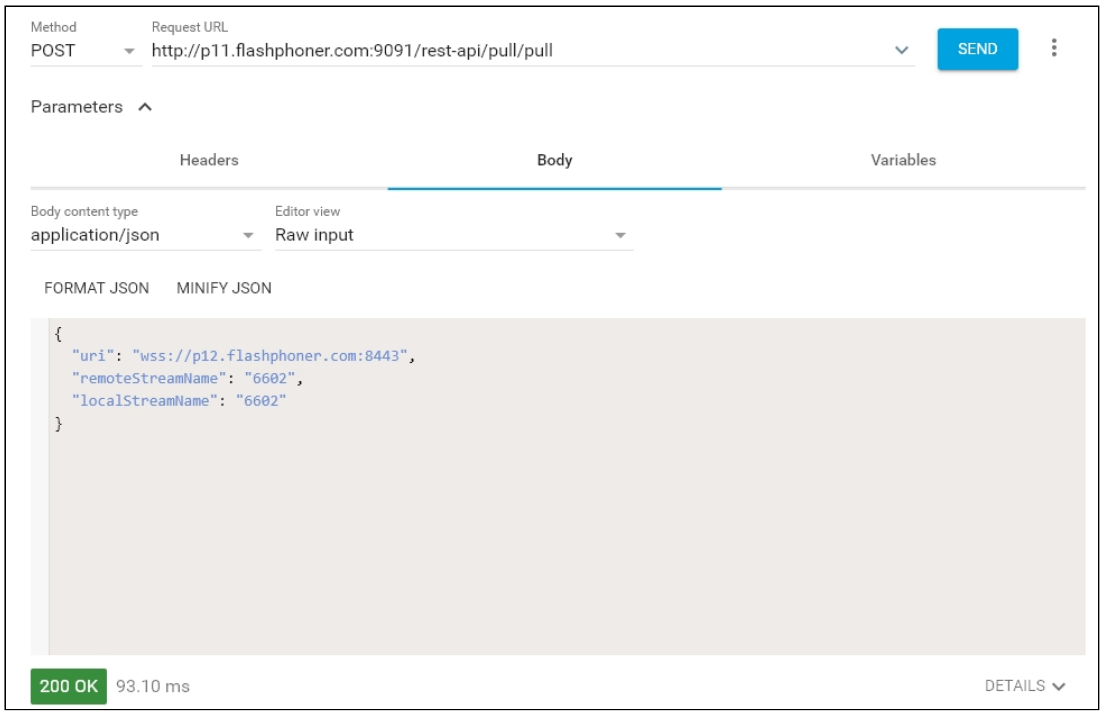
Краткое руководство по тестированию

1. Для теста используем:
2. два WCS-сервера;
3. браузер Chrome и REST-клиент для отправки запросов на сервер;
4. веб-приложение Two Way Streaming для публикации потока;
5. веб-приложение Player для воспроизведения захваченного потока в браузере.
6. Откройте веб-приложение Two Way Streaming, опубликуйте поток на первом сервере

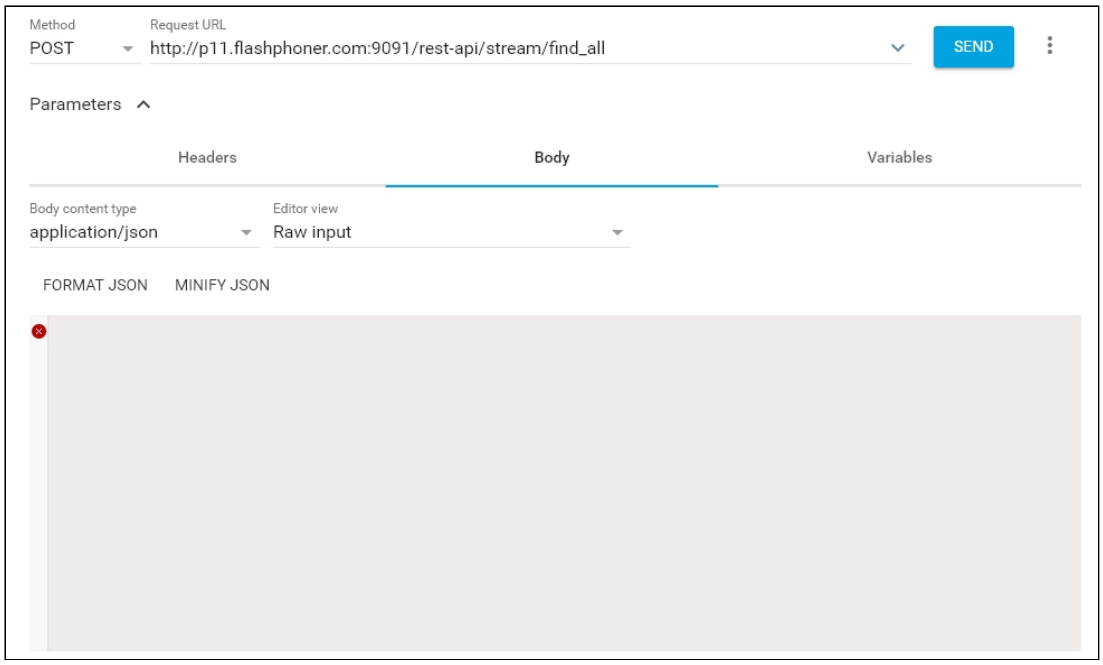


7. Откройте REST-клиент. Отправьте на второй сервер запрос `/pull/pull`, указав в параметрах:
8. URL WCS-сервера, с которого будет захватываться поток;

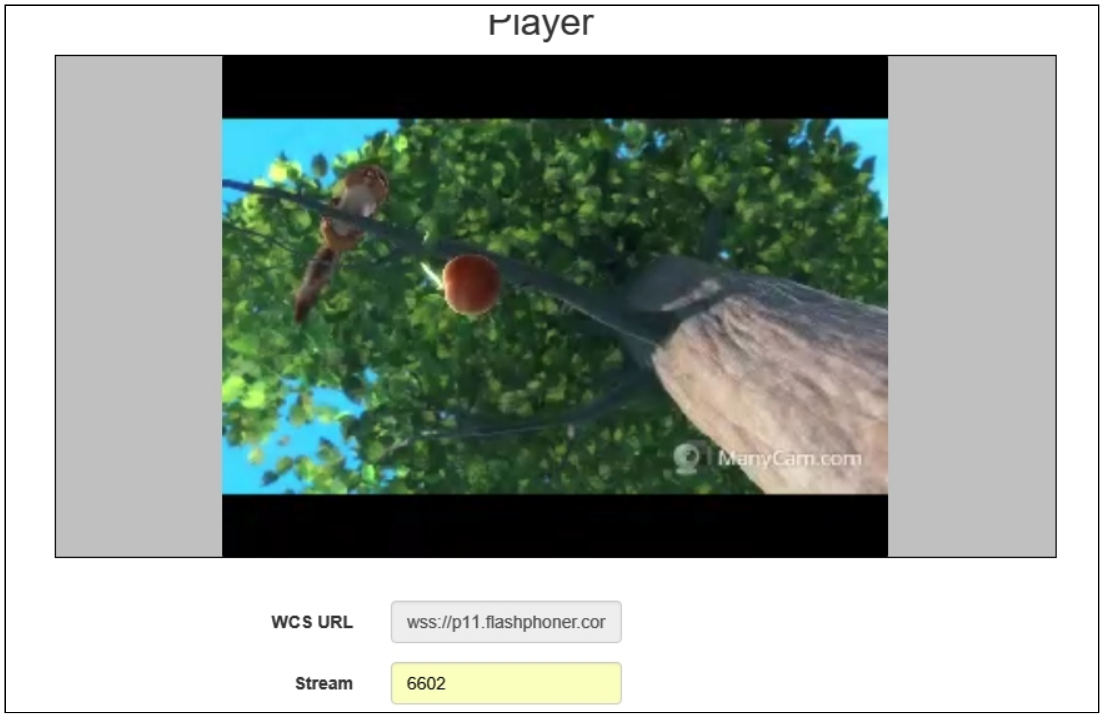
9. имя потока, опубликованного на сервере;
10. локальное имя потока



11. Убедитесь, что поток захвачен сервером. Для этого отправьте запрос `/pull/find_all`:



12. Откройте веб-приложение Player на втором сервере, укажите в поле `Stream` локальное имя потока, нажмите `Start`



Последовательность выполнения операций

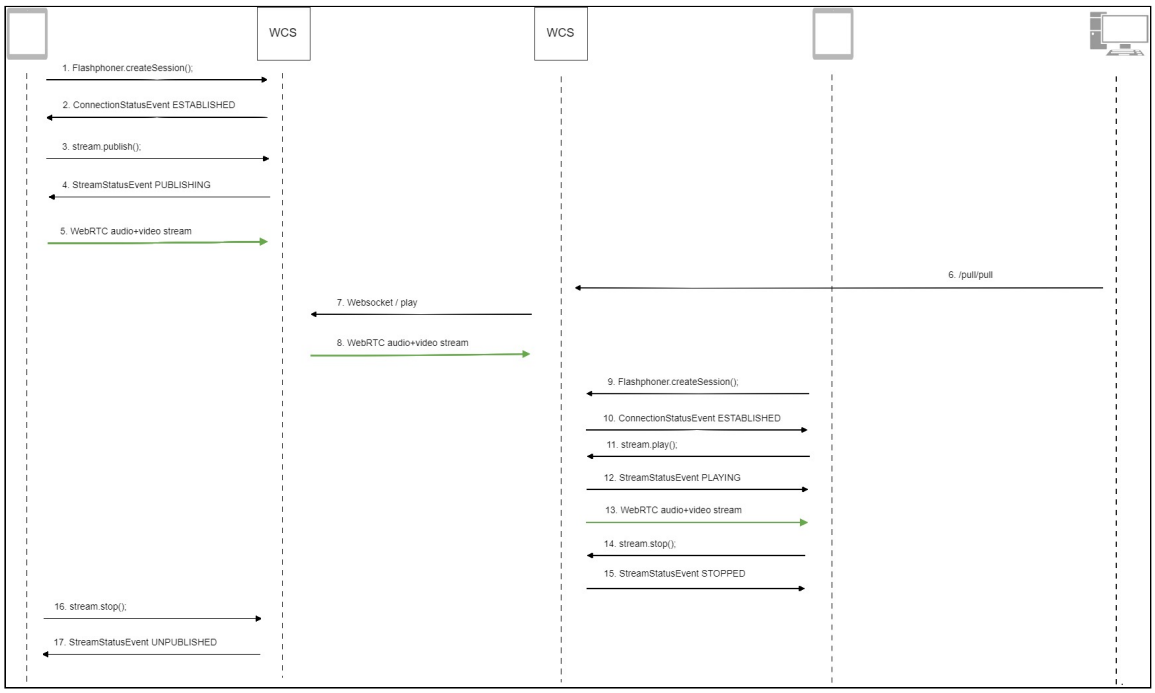
Ниже описана последовательность вызовов при использовании примера Two Way Streaming для публикации потока на одном WCS сервере и Player для воспроизведения потока на другом WCS сервере

two_way_streaming.html

two_way_streaming.js

player.html

player.js



1. Установка соединения с сервером
   `Flashphoner.createSession()` code

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

2. Получение от сервера события, подтверждающего успешное соединение
   `SESSION_STATUS.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 () {
  ...
});
```

3. Публикация потока
   `Stream.publish()` code

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

4. Получение от сервера события, подтверждающего успешную публикацию потока
   `STREAM_STATUS.PUBLISHING` code

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

5. Отправка потока по WebRTC на первый сервер
6. Отправка REST-запроса `/pull/pull` на второй сервер
7. Запрос потока с первого сервера
8. Отправка потока по WebRTC на второй сервер

9. Установка соединения со вторым сервером

`Flashphoner.createSession()` code

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

10. Получение от сервера события, подтверждающего успешное соединение

`SESSION_STATUS.ESTABLISHED` code

```
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(){
    ...
});
```

11. Запрос на воспроизведение потока

`Stream.play()` code

```
stream = session.createStream(options).on(STREAM_STATUS.PENDING,
function(stream) {
    var video = document.getElementById(stream.id());
    if (!video.hasListeners) {
        video.hasListeners = true;
        video.addEventListener('playing', function () {
            $("#preloader").hide();
        });
        video.addEventListener('resize', function (event) {
            var streamResolution = stream.videoResolution();
            if (Object.keys(streamResolution).length === 0) {
                resizeVideo(event.target);
            } else {
                // Change aspect ratio to prevent video stretching
                var ratio = streamResolution.width /
streamResolution.height;
                var newHeight = Math.floor(options.playWidth / ratio);
                resizeVideo(event.target, options.playWidth, newHeight);
            }
        });
    }
```

```
    ...
});
stream.play();
```

12. Получение от сервера события, подтверждающего успешный захват и проигрывание потока
    `STREAM_STATUS.PLAYING` code

```
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();
```

13. Отправка потока по WebRTC зрителю
14. 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);
    ...
}
```

15. Получение от сервера события, подтверждающего остановку воспроизведения потока
    `STREAM_STATUS.STOPPED` code

```
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();
```

16. Остановка публикации потока

`Stream.stop()` code

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

17. Получение от сервера события, подтверждающего остановку публикации потока

`STREAM_STATUS.UNPUBLISHED` code

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