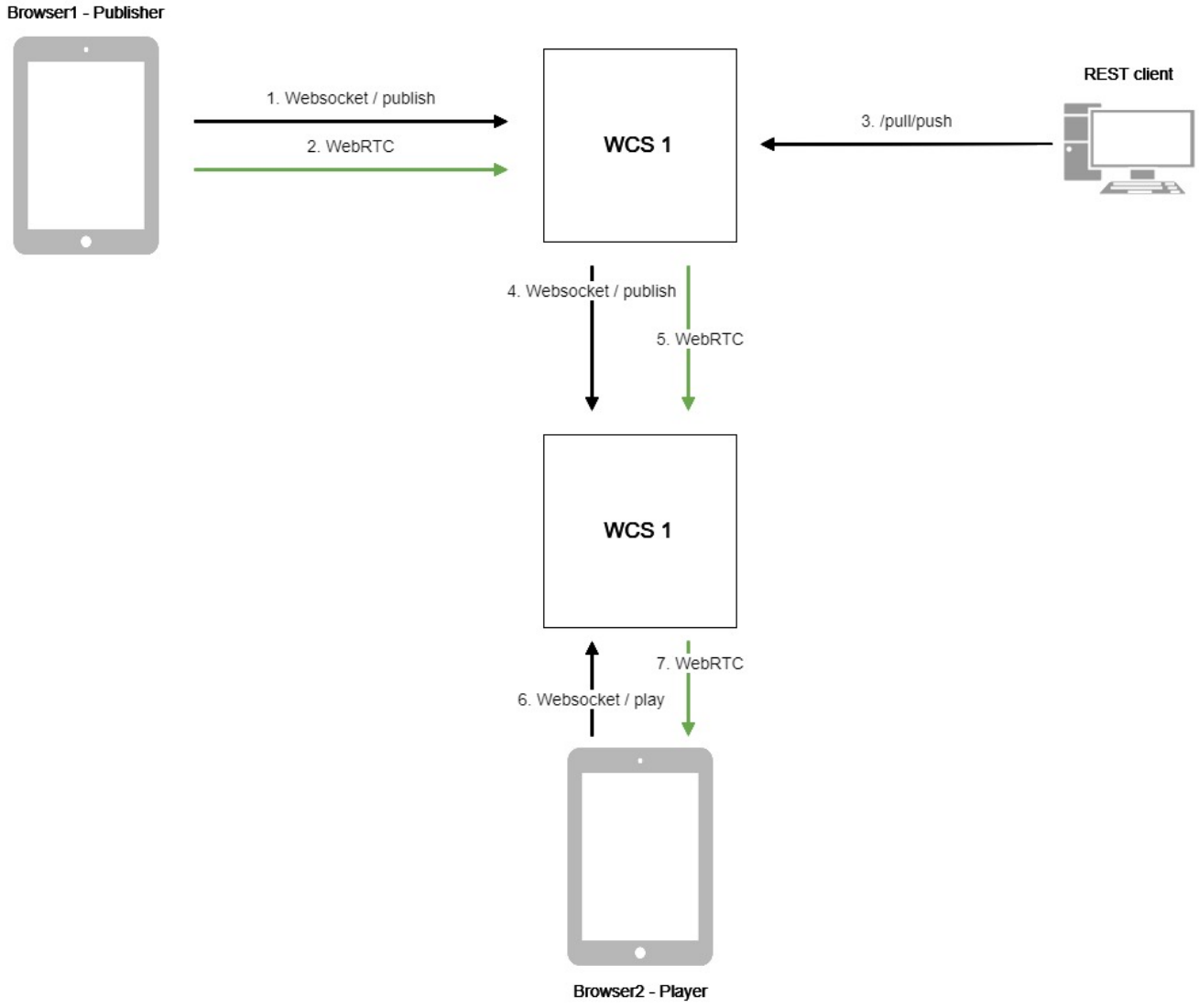
На другой WCS сервер по WebRTC

- Описание
  - Схема работы
- REST-вызовы
  - REST-методы и статусы ответа
  - Параметры
- Настройка
- Краткое руководство по тестированию
- Последовательность выполнения операций (Call flow)

Описание

WCS может по требованию ретранслировать видеопоток по WebRTC на другой WCS-сервер. Для управления ретрансляцией WebRTC-потока используется REST API.

Схема работы



1. Браузер соединяется с сервером WCS1 по протоколу Websocket и отправляет команду publish.
2. Браузер захватывает микрофон и камеру и отправляет WebRTC поток на сервер.
3. REST-клиент отправляет на сервер WCS1 запрос /pull/push.
4. WCS1 публикует поток на WCS2.
5. WCS2 получает WebRTC поток с WCS1.
6. Второй браузер устанавливает соединение с сервером WCS2 по Websocket и отправляет команду play.

7. Второй браузер получает WebRTC поток и воспроизводит этот поток на странице.

REST-вызовы

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

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

Здесь:

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

REST-методы и статусы ответа

REST-метод	Пример тела REST-запроса	Пример тела REST-ответа	Статусы ответа	Описание

/pull/push	<pre>{
	"uri":"wss://demo.flashphoner.com:8443",
	"localStreamName": "testStream",
	"remoteStreamName": "testStream"
	}</pre>
			409 - Conflict
			
			500 - Internal error
				Транслировать WebRTC-поток по указанному URL

Параметры

Имя параметра	Описание	Пример
uri	URL WebRTC-потока	wss://demo.flashphoner.com:8443
localMediaSessionId	Идентификатор сессии	5a072377-73c1-4caf-abd3
remoteMediaSessionId	Идентификатор сессии на дргуом 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, публикация 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
```

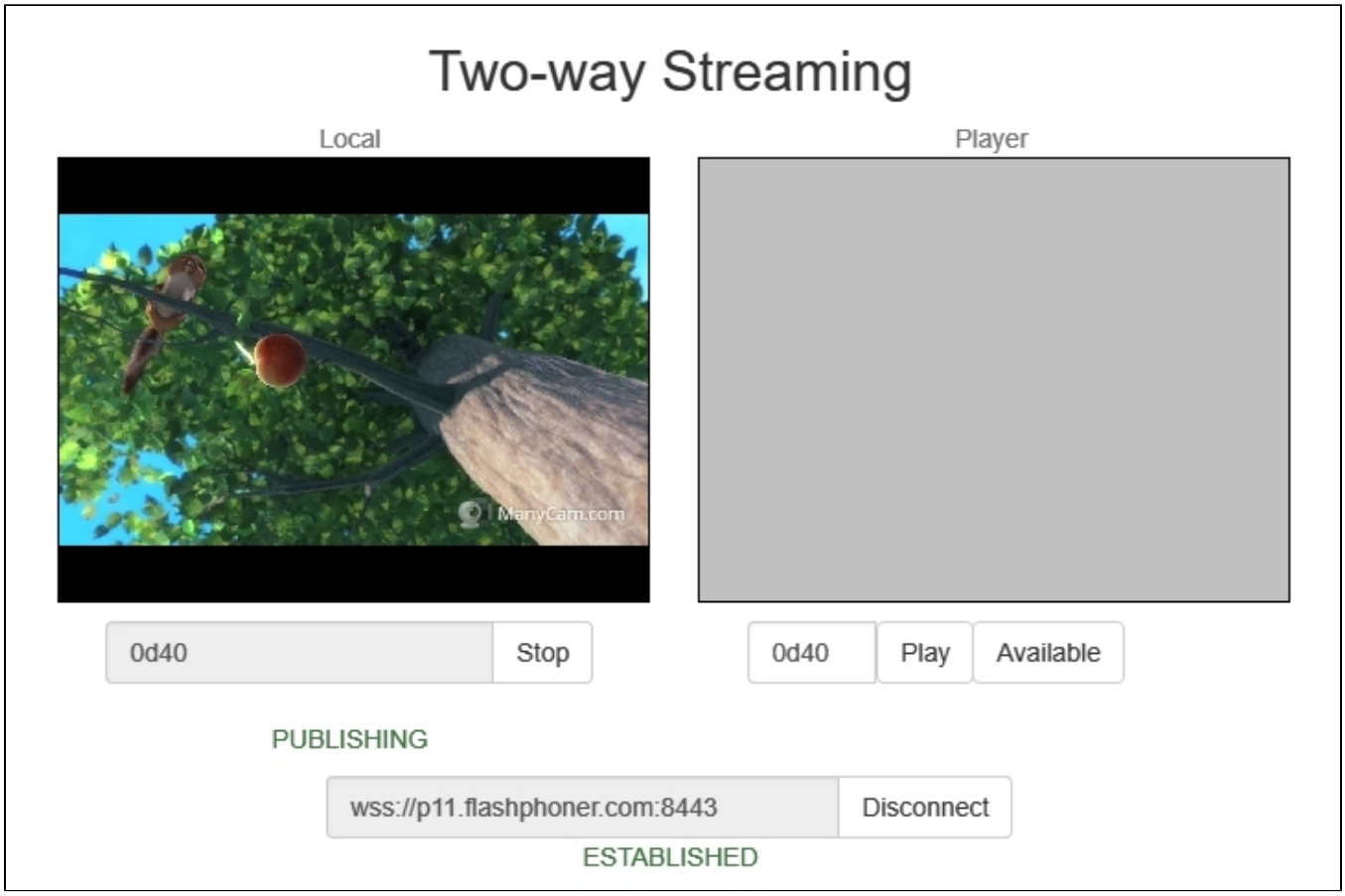
Изменения в настройку должны быть внесены на обоих WCS-серверах: том, который публикует поток, и том, на который этот поток публикуется.

Краткое руководство по тестированию

1. Для теста используем:

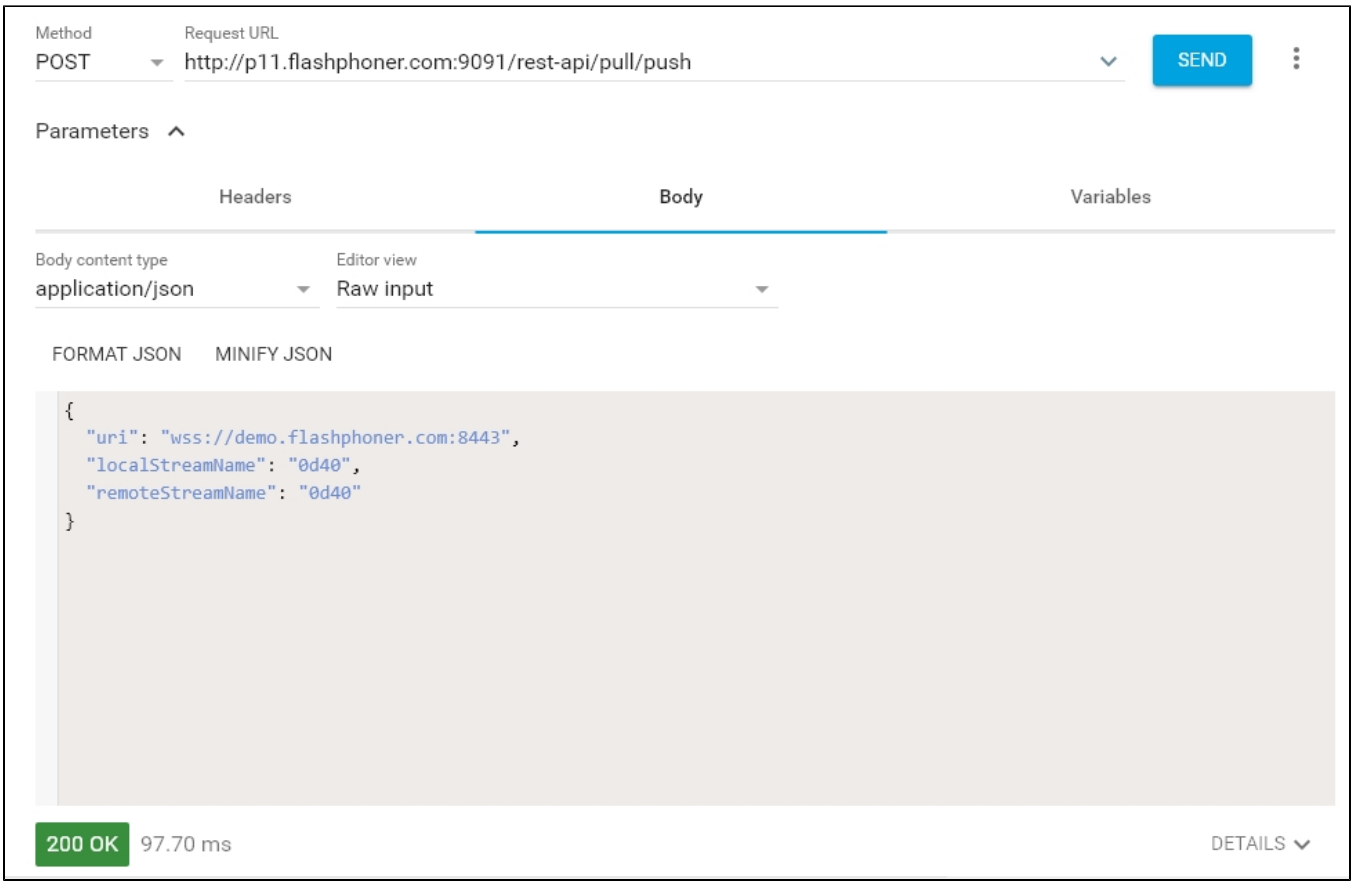
- два WCS-сервера;
- браузер Chrome иREST-клиентдля отправки запросов на сервер;
- веб-приложениеTwo Way Streamingдля публикации потока;
- веб-приложениеPlayerдля воспроизведения захваченного потока в браузере.

2. Откройте веб-приложение Two Way Streaming, опубликуйте поток на сервере

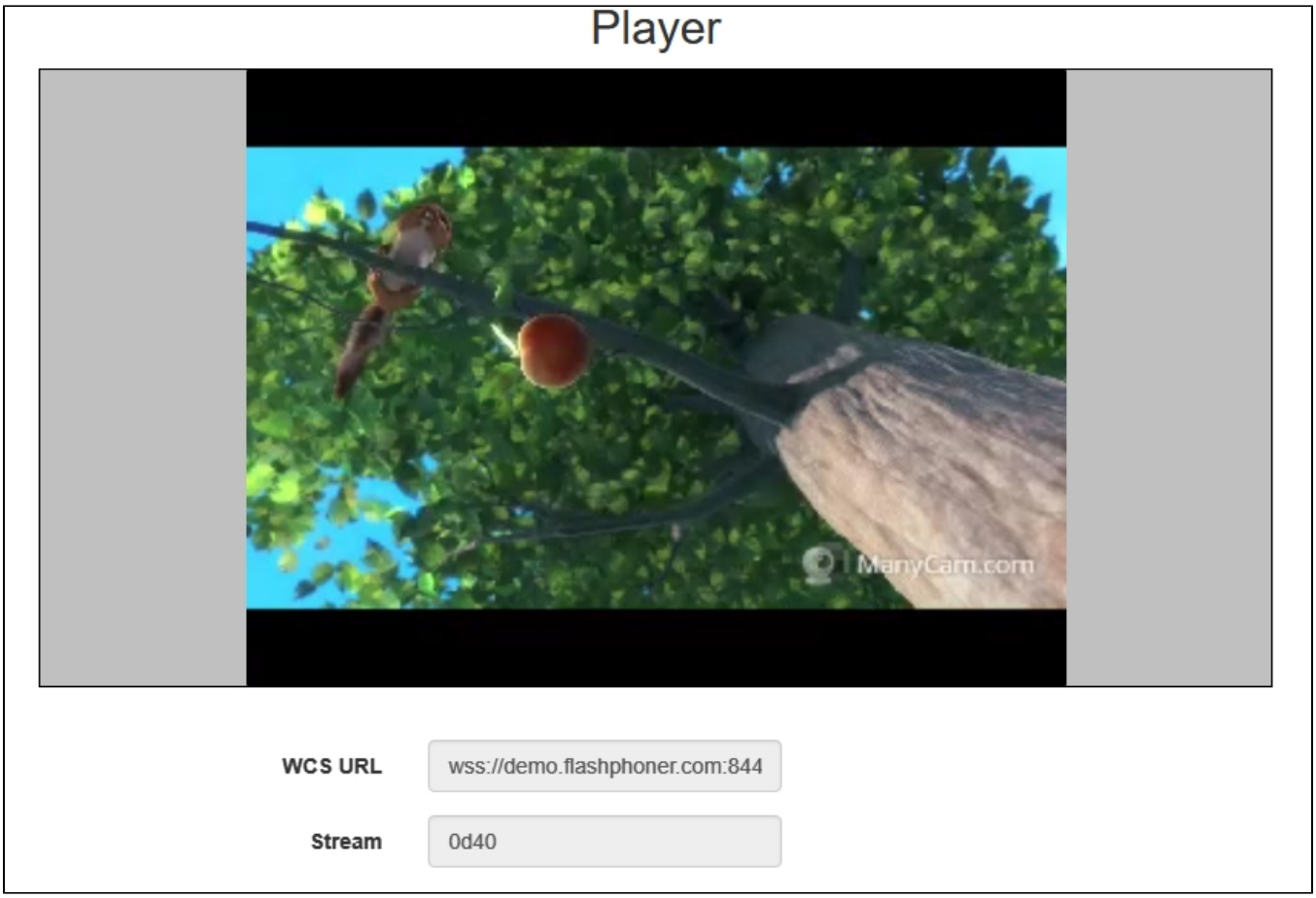


3. ОткройтеREST-клиент. Отправьте запрос /pull/push, указав в параметрах:

- URL WCS-сервера, с которого будет захватываться поток;
- имя потока, опубликованного на сервере
- локальное имя потока



4. Откройте веб-приложение Player, укажите в поле Stream локальное имя потока, нажмите Start



Последовательность выполнения операций (Call flow)

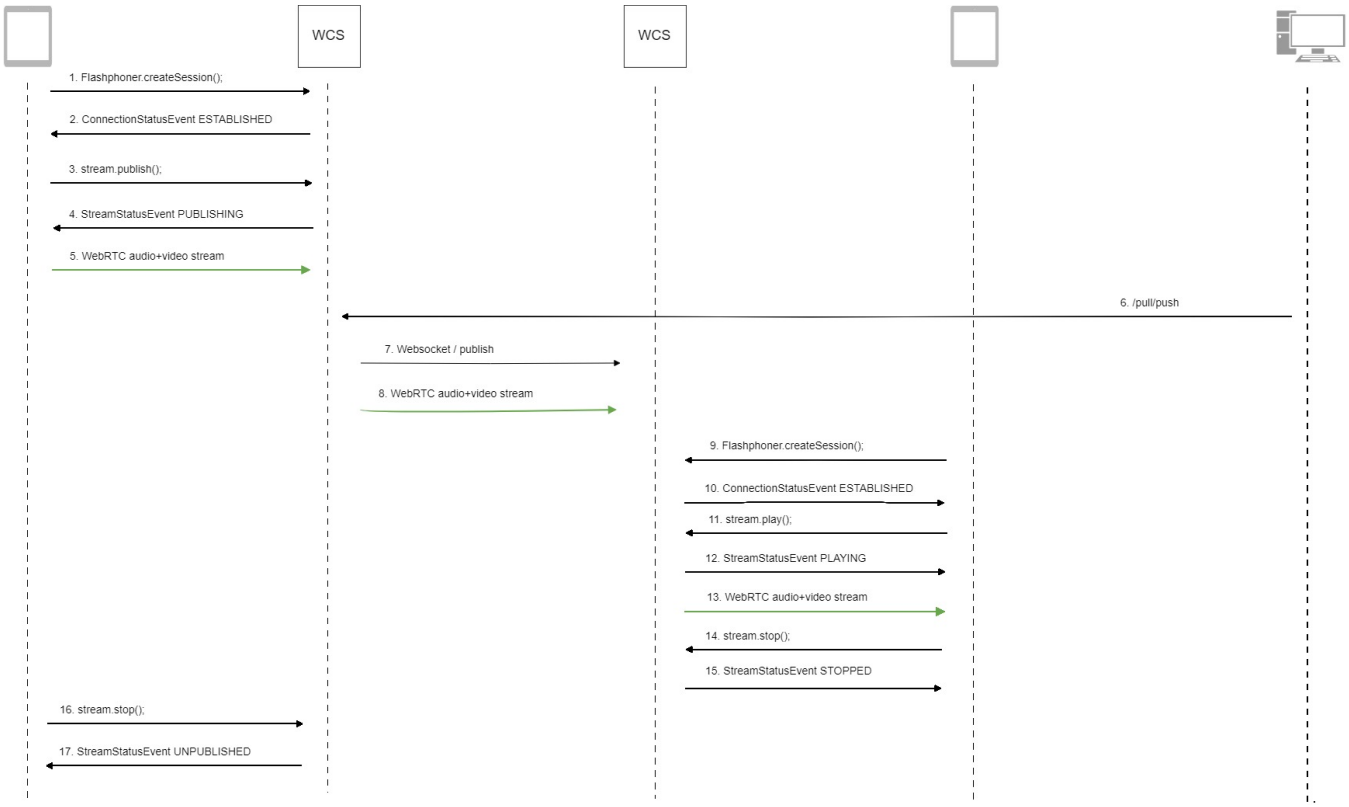
Ниже описана последовательность вызовов при использовании примера Two Way Streaming для публикации потока на одном WCS сервере и Player для воспроизведения потока на другом WCS сервере

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.

ConnectionStatusEvent ESTABLISHEDcode

```
    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
    ...
}).publish();
```

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

StreamStatusEvent, статус PUBLISHINGcode

```
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 аудио-видео потока по WebRTC на сервер

6. Отправка REST-запроса /pull/push на сервер

7. Публикация потока на второй сервер

8. Отправка аудио-видео 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.

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

11. Запрос на воспроизведение потока.

stream.play();code

```
    stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
        ...
    });
    stream.play();
```

12. Получение от сервера события, подтверждающего успешный захват и проигрывание 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();
```

13. Отправка аудио-видео потока по WebRTC

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.

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();
```

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.

StreamStatusEvent, статус UNPUBLISHEDcode

```
   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();
```