

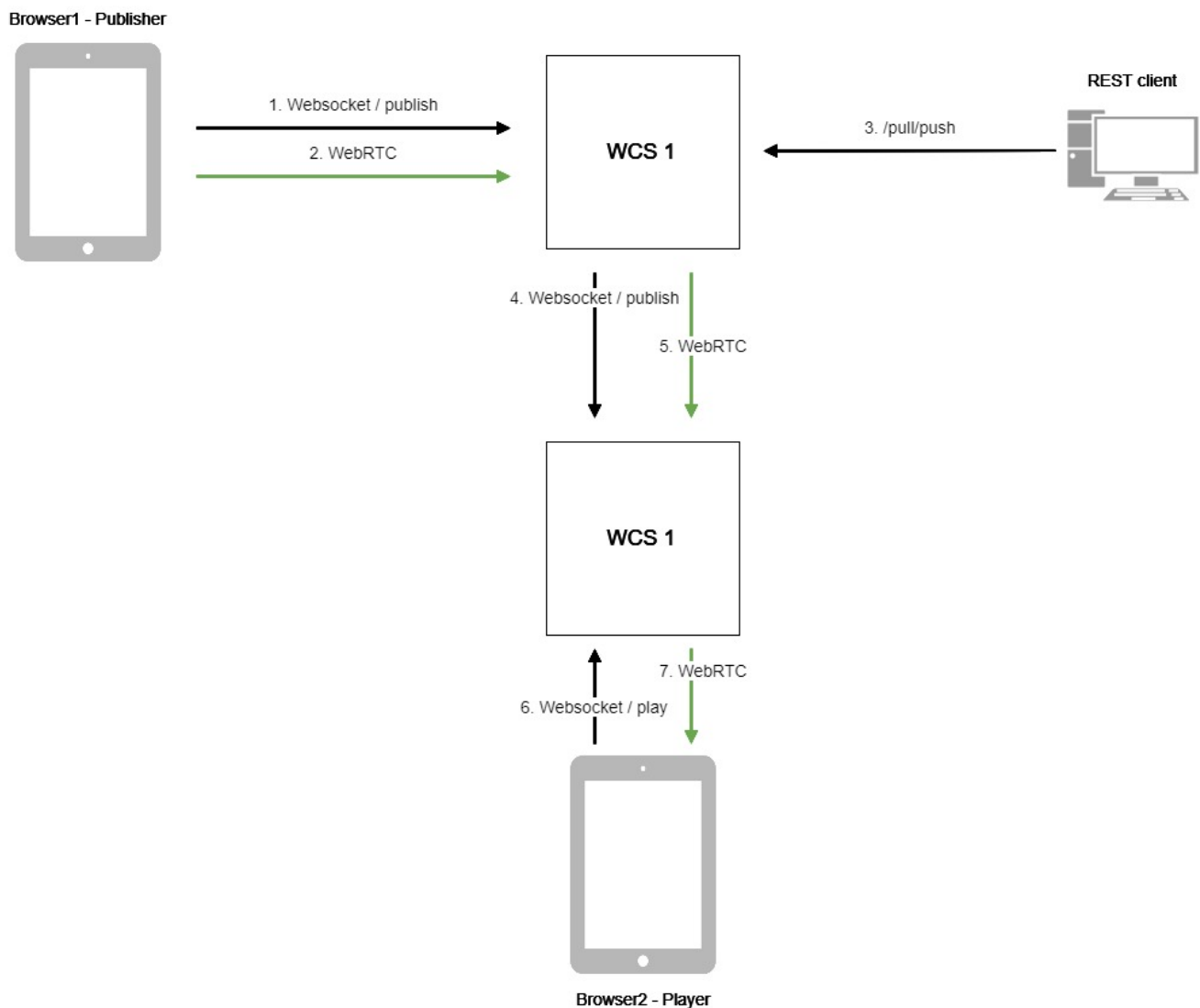
To another WCS server via WebRTC

- [Overview](#)
 - [Operation flowchart](#)
- [REST queries](#)
 - [REST-methods and response statuses](#)
 - [Parameters](#)
- [Configuration](#)
- [Quick manual on testing](#)
- [Call flow](#)

Overview

Upon request WCS can rebroadcast a video stream via WebRTC to another WCS server. Republishing of a WebRTC stream is managed by the REST API.

Operation flowchart



1. The browser connects to the WCS1 server via the Websocket protocol and sends the publish command.
2. The browser captures the microphone and the camera and send the WebRTC stream to the server.
3. The REST client sends to the WCS1 server the /pull/push query.
4. WCS1 publishes the stream to WCS2.
5. WCS2 receives the WebRTC stream from WCS1.
6. The second browser establishes a connection to the WCS2 server via Websocket and sends the play command.
7. The second browser receives the WebRTC stream and plays this stream on the page.

REST queries

A REST query must be an HTTP/HTTPS POST query in the following form:

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

Where:

- streaming.flashphoner.com- is the address of the WCS server
- 8081 - is the standard REST / HTTP port of the WCS server
- 8444- is the standard HTTPS port
- rest-api- is the required prefix
- /pull/push- is the REST-method used

REST-methods and response statuses

REST-method	Example of REST query	Example of REST response body	Response statuses	Description
/pull/push	<pre>{ "uri": "wss://demo.flashphoner.com:8443", "localStreamName": "testStream", "remoteStreamName": "testStream" }</pre>		409 - Conflict 500 - Internal error	Broadcasts the WebRTC stream at the specified URL

Parameters

Parameter name	Description	Example
uri	URL of the WebRTC stream	wss://demo.flashphoner.com:8443
localMediaSessionId	Session identifier	5a072377-73c1-4caf-abd3
remoteMediaSessionId	Identifier of the session on the remote server	12345678-abcd-dead-beaf
localStreamName	Local name assigned to the captured stream. The stream can be fetched from the WCS server using this name	testStream
remoteStreamName	Name of the captured stream on the remote server	testStream
status	Current status of the stream	NEW

Configuration

By default, WebRTC stream is pulled over unsecure Websocket connection, i.e. WCS server URL has to be set as <ws://demo.flashphoner.com:8080>. To use Secure Websocket, the parameter must be set in file [flashphoner.properties](#)

```
wcs_agent_ssl=true
```

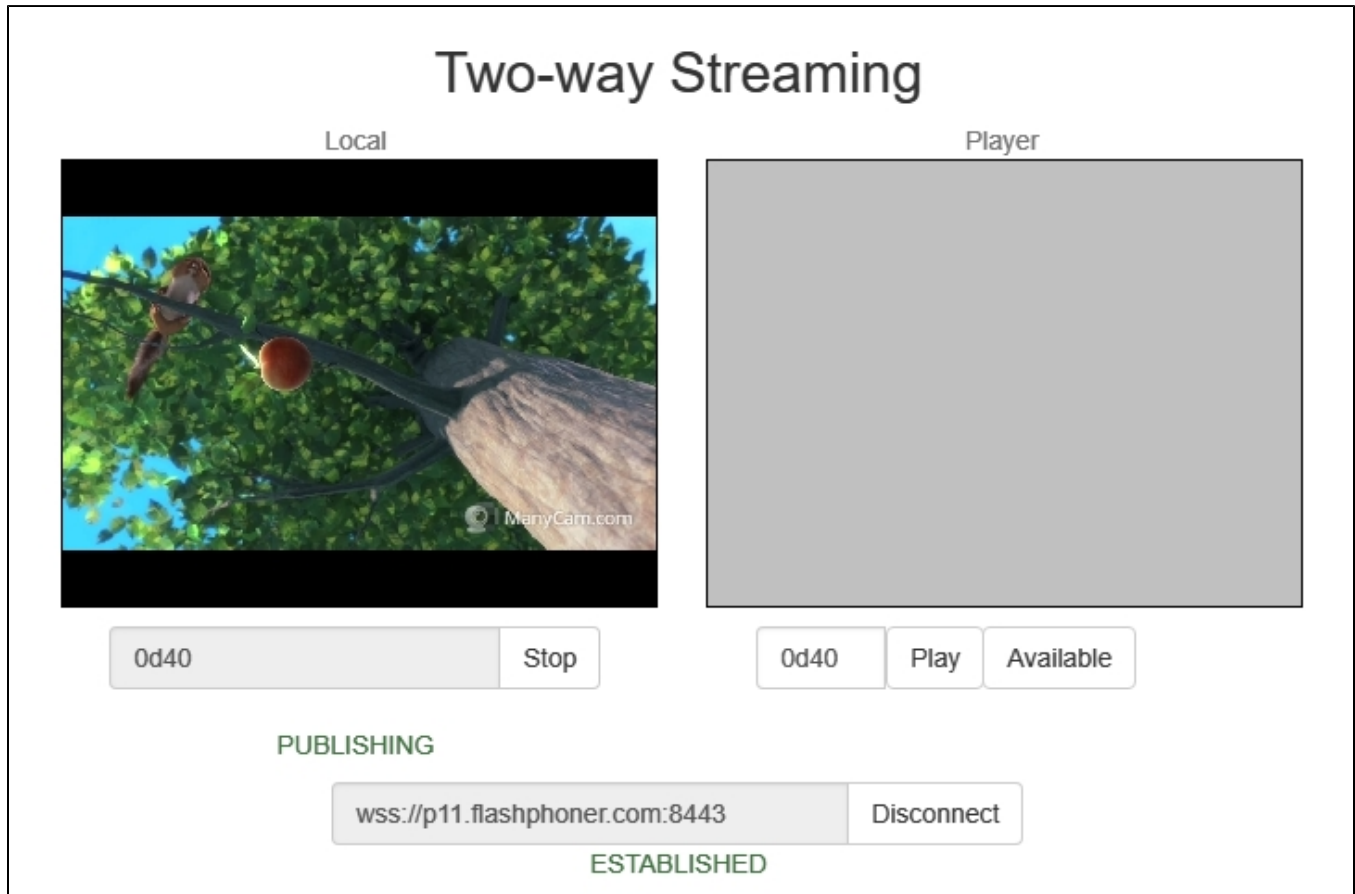
This change has to be made on both WCS servers: the server that publishes the stream and the server the stream is pushed to.

Quick manual on testing

1. For this test we use:

- two WCS servers;
- the Chrome browser and the [REST client](#) to send queries to the server;
- the [Two Way Streaming](#) web application to publish the stream;
- the [Player](#) web application to play the captured stream in a browser.

2. Open the Two Way Streaming web application, publish the stream on the server



3. Open the REST client. Send the /pull/push query and specify in its parameters:

- the URL of the WCS server the stream is captured from;
- the name of the stream published on the server
- the localname of the stream

Method

POST

Request URL

http://p11.flashphoner.com:9091/rest-api/pull/push

SEND

Parameters

^

Headers

Body

Variables

Body content type

application/json

Editor view

Raw input

FORMAT JSON

MINIFY JSON

```
{
  "uri": "wss://demo.flashphoner.com:8443",
  "localStreamName": "0d40",
  "remoteStreamName": "0d40"
}
```

200 OK

97.70 ms

DETAILS

4. Open the Player web application and specify the local stream name in the Stream field, then click Start

Player



WCS URL

wss://demo.flashphoner.com:844

Stream

0d40

Call flow

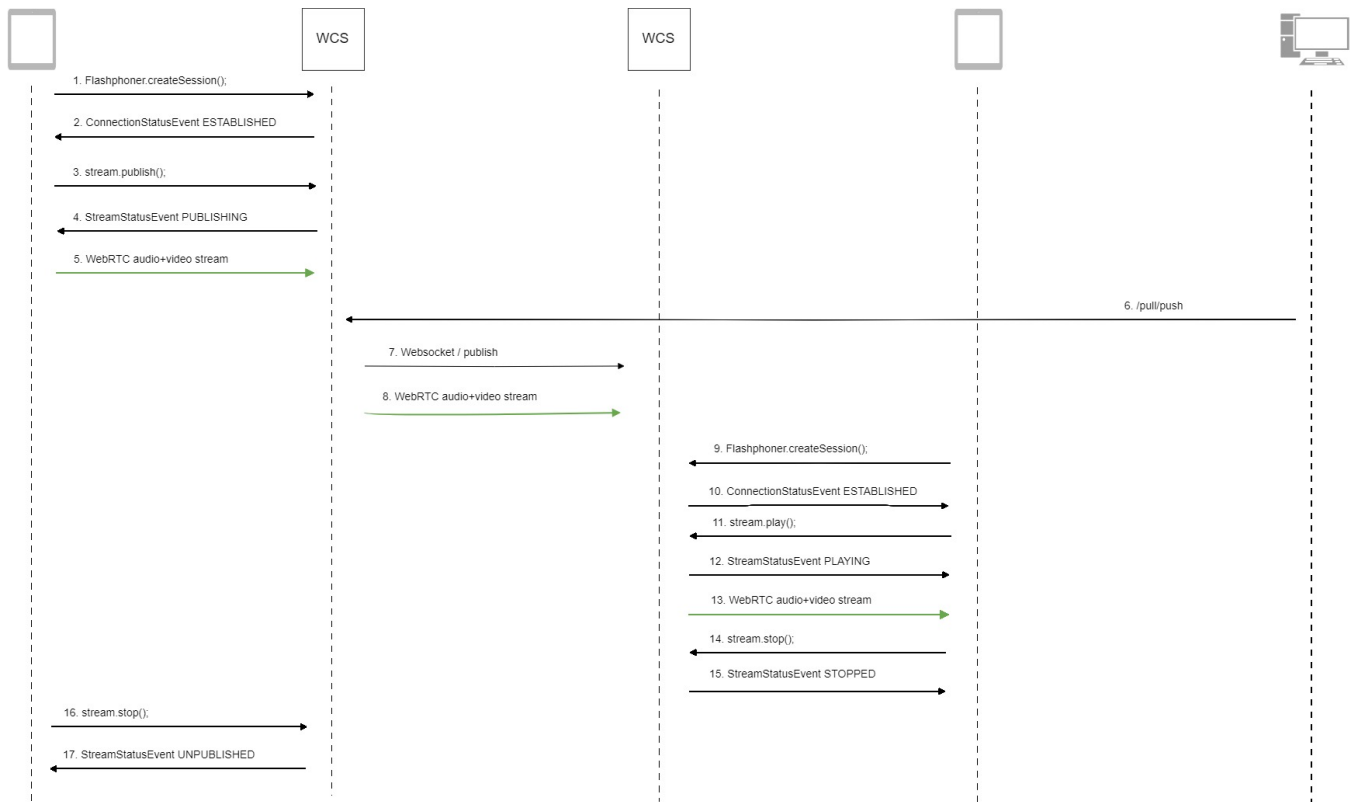
Below is the call flow when using the Two Way Streaming example to publish a stream on one WCS server and the Player example to play that stream on another WCS server.

[two_way_streaming.html](#)

[two_way_streaming.js](#)

[player.html](#)

[player.js](#)



1. Establishing a connection to the server.

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. Receiving from the server an 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 () {
    ...
});
  
```

3. Publishing the stream.

stream.publish();[code](#)

```

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

```

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

StreamStatusEvent, 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. Sending the audio-video stream via WebRTC to the server

6. Sending the /pull/push REST query to the server

7. Publishing the stream on the second server

8. Sending the audio-video stream via WebRTC to the second server

9. Establishing a connection to the second server.

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. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent 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. Requesting to play the stream.

`stream.play();`[code](#)

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

12. Receiving from the server an event that confirms successful capturing and playing of the stream.

`StreamStatusEvent`, status `PLAYING`[code](#)

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

13. Sending the audio-video stream via WebRTC

14. Stopping the playback of the stream.

`stream.stop();`[code](#)

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

15. Receiving from the server an event confirming the playback of the stream is stopped.

`StreamStatusEvent`, 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. Stopping publishing the stream.

`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. Receiving from the server an event that confirms unpublishing of the stream.

`StreamStatusEvent, 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();

```