

Taking a PNG snapshot of the stream

- [Overview](#)
 - [Supported protocols](#)
 - [Supported snapshot formats](#)
 - [Operation flowchart](#)
- [REST queries](#)
 - [REST-methods and response statuses](#)
 - [Parameters](#)
 - [Sending the REST query to the WCS server](#)
 - [Configuration](#)
- [JavaScript API](#)
- [Quick manual on testing](#)
- [Call flow](#)
- [Automatic stream snapshot taking](#)

Overview

WCS provides a way to take a snapshot of the published stream using REST-queries as well as using JavaScript API.

Supported protocols

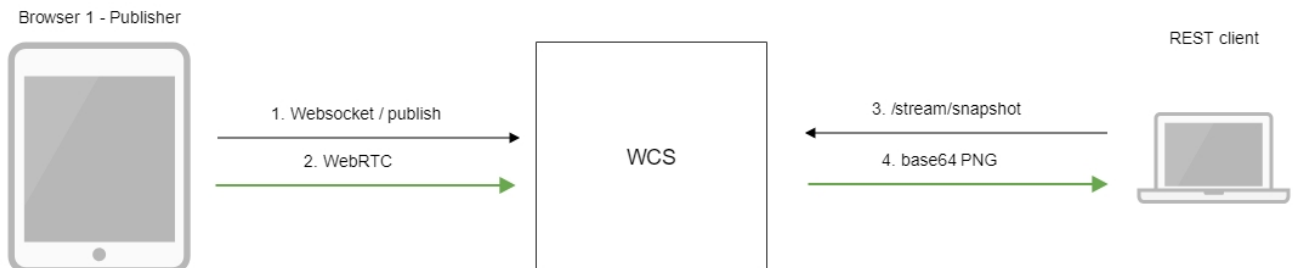
- WebRTC
- RTMP
- RTSP

Supported snapshot formats

- PNG

Operation flowchart

1: Using the REST query



1. The browser connects to the server via the Websocket protocol and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The REST client sends to the WCS the /stream/snapshot REST query.
4. The REST client receives a response with the base64-encoded snapshot of the stream.

2: Using JavaScript API



1. The browser connects to the server via the Websocket protocol and sends the publish command.
2. The browser captures the microphone and the camera and sends the WebRTC stream to the server.
3. The second browser establishes a connection also via Websocket and sends the play command.
4. The second browser receives the WebRTC stream and plays this stream on the page.
5. The second browser invokes `stream.snapshot()` to take a snapshot.
6. The second browser receives a response with the base64-encoded snapshot of the stream.

REST queries

WCS-server supports the `/stream/snapshotREST` method to take a snapshot:

A REST-query must be an HTTP/HTTPS POST request as follows:

- HTTP:<http://streaming.flashphoner.com:8081/rest-api/stream/snapshot>
- HTTPS:<https://streaming.flashphoner.com:8444/rest-api/stream/snapshot>

Here:

- streaming.flashphoner.com - is the address WCS server
- 8081 - is the standard REST / HTTP port of the WCS server
- 8444- is the standard HTTPS port
- rest-api- is the required part of the URL
- /stream/snapshot- is the REST method used

REST-methods and response statuses

REST-method	Example of the REST query	Example of the REST response body	Response statuses
/stream /snapshot	<pre>{ "streamName": "64966f33" }</pre>	<pre>{ "data": "iVBORw0KGgoAAAANSUHEUGAAAUAADwCAYAAABxLb1rAAAACXBIWXMAAAAAAAAAAQCEeRdzAAQAE1EQVR4nOzd95Pcd37feVjhrFKwXT6fr+r+gPvh6nx1V5bvLMT3liXbkiWtrFlv1O5qVxu4icucwUyABAMIAgSIQOQcBzlnzAADDIDJuadzzjmH173f7+/30216gO4GiCHIZtWjeqZnejAAu5/9+XzD57tgwYIF60jo6PiSmvdfokojo20+zPsv0NHR0Tff5v0X60jo6Jgv8/4LdHR0dMyXef8FOjo6OubLvP8CHR0dHfN13n+Bjo6Ojvky779AR0dHx3yZ91+go6OjY77M+y/Q0dHRMV/m/Rfo6OjomC/z/gt0dHR0zJd5/wU60jo65su8/wIdHR0d82Xef4GOjo6O+TLvv0BHR0fHfJn3X6Cjo6Njvsz7L9DR0dExX+b9F+jo6OiYL/P+C3R0dHTML3n/BTo6Ojrmy7z/Ah0dHR1t+Ue636rzGzr19Tv8jPn/S3R0dHS04z4E8HfR0dHR8XD5DaGC94/rdALY0dHxBXafAlj/wI6OjofPb7foXh//edcJYEFh14gaCd0tDPXf0+rjHxbNT4F/6/fr0dHxkPsNnhb+DvnNO7zYf1P7nt/8vTYf/3nzj8hvk/+B/GPt76DIfeY3dHP8jN/8g/8JHR0dD7d/9Hv/Ixb87j/Hgt/5ZzSt+6dY8D/8Eyz47T+o4Pv4a/Q9v/H7/6L1x3+e8O+m/134d/+tf/Iv8dv/9H/G7/zz/6Xs9/7ZvxS//8/+hfiDP/gD/P7v/z5+7/d+D7/7u7+L3/md3xELV16PoaOj4" }</pre>	200 - Snapshot is taken 404 - Stream not found 500 - Snapshot taking error

Parameters

Parameter name	Description	Example
streamName	Unique stream name	64966f33
data	Snapshot file encoded to base64	iVBORw0KGgoAAAANSUHEUGAAAUAADwCAYAAABxLb1rAAAACXBIWXMAAAAAAAAAAQCEeRdzAAQA

Sending the REST query to the WCS server

To send the REST query to the WCS server you need to use a [REST-client](#).

Configuration

Since build [5.2.1116](#), a maximum snapshot taking duration, including a possible server disk I/O delay, may be configured when taking snapshot via REST API. By default, maximum duration is set to 3000 ms, and 30 checks if snapshot file is ready will be performed during this interval

```
snapshot_taking_interval_ms=3000
snapshot_taking_attempts=30
```

If the snapshot file is not ready, and the interval is expired, /stream/snapshot request will return the following error

```
{
  "exception": "com.flashphoner.rest.server.exception.InternalErrorException",
  "reason": "com.flashphoner.rest.server.exception.InternalErrorException, Internal Server Error, Snapshot
response timeout, ts: 1640836780816, path: /rest-api/stream/snapshot",
  "path": "/rest-api/stream/snapshot",
  "error": "Internal Server Error",
  "message": "Snapshot response timeout",
  "timestamp": 1640836780816,
  "status": 500
}
```

JavaScript API

The snapshot method of the Stream object in WebSDK is intended to take stream snapshots. Example of use of this method can be found in the Stream Snapshot web applications that publishes a stream and take a snapshot.

[stream-snapshot.html](#)

[stream-snapshot.js](#)

1. Creating a new stream from the published stream

code:

```
function snapshot(name) {
  setSnapshotStatus();
  var session = Flashphoner.getSessions()[0];
  session.createStream({name: name}).on(STREAM_STATUS.SNAPSHOT_COMPLETE, function(stream){
    ...
  })
}
```

2. Invoking the snapshot() method

code:

```
function snapshot(name) {
  setSnapshotStatus();
  var session = Flashphoner.getSessions()[0];
  session.createStream({name: name}).on(STREAM_STATUS.SNAPSHOT_COMPLETE, function(stream){
    ...
  }).snapshot();
}
```

3. Upon receiving the SNAPSHOT_COMPLETE event, the stream.getInfo() function returns the base64 encoded snapshot

code:

```
function snapshot(name) {
    setSnapshotStatus();
    var session = Flashphoner.getSessions()[0];
    session.createStream({name: name}).on(STREAM_STATUS.SNAPSHOT_COMPLETE, function(stream){
        console.log("Snapshot complete");
        setSnapshotStatus(STREAM_STATUS.SNAPSHOT_COMPLETE);
        snapshotImg.src = "data:image/png;base64,"+stream.getInfo();
        ...
    })
}
```

4. The stream stops

code:

```
function snapshot(name) {
    setSnapshotStatus();
    var session = Flashphoner.getSessions()[0];
    session.createStream({name: name}).on(STREAM_STATUS.SNAPSHOT_COMPLETE, function(stream){
        ...
        stream.stop();
    }).on(STREAM_STATUS.FAILED, function(stream){
        setSnapshotStatus(STREAM_STATUS.FAILED);
        console.log("Snapshot failed, info: " + stream.getInfo());
    }).snapshot();
}
```

Quick manual on testing

1. For the test we use:

- the demo server at demo.flashphoner.com;
- the Chrome browser and the [REST-client](#) to send queries to the server;
- the [Two Way Streaming](#) web application to publish the stream;
- the <https://www.motobit.com/util/base64-decoder-encoder.asp> service to decode the snapshot.

2. Open the page of the Two Way Streaming application. Click "Connect", then click "Publish" to publish the stream:

Two-way Streaming

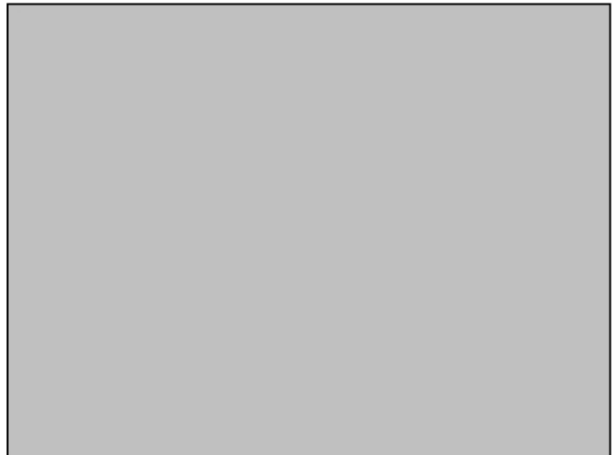
Local



abeb

Stop

Player



abeb

Play

Available

PUBLISHING

wss://p11.flashphoner.com:8443

Disconnect

ESTABLISHED

3. Open the REST-client. Send the /stream/snapshot query and pass the name of the published stream in parameters:

Method

Request URL

POST

http://p11.flashphoner.com:9091/rest-api/stream/snapshot

SEND

Parameters ^

Headers

Body

Variables

Body content type

Editor view

application/json

Raw input

FORMAT JSON MINIFY JSON

```
{  "streamName": "abeb"}
```

4. Make sure the response is received:

200 OK458.60 ms

DETAILS

```
{
  "data": "iVBORw0KGgoAAAANSUHEUgAAAAAADwCAYAAABxLb1rAAACXBBIWMAAAAAAAAAAAQCEeRdzAAAQAE1EQVR4nOzd95Pcd37feVjhrFKwXT6fr+r+gPvh6nx1V5bvLMt31iXbkiWtrF1v1O5qVxu4iCucwUyABAMIAgSIQ0QcBz1nzAADDIDJuaDzzjmH173f7+/30216g04GiCHIZtwjeqZnejaAu5/9+XzD57tgwYIF60jo6PiSmvdfoK0jo20+zPsv0NHR0TFf5v0X60jo6Jgv8/4LdHR0dMyXef8FOjo6OubLvP8CHR0dHfN13n+Bjo60jvky779AR0dHx3yZ91+go60jY77M+y/Q0dHRMv/m/Rfo60jomC/z/gt0dHR0zJd5/wU60jo65su8/wIdHR0d82Xef4G0jo60+TLvv0BHR0fHfJn3X6Cjo6Njvsz7L9DR0dExX+b9F+jo60iYL/P+C3R0dHTM13n/BTo60jrmY7z/Ah0dHR1t+Ue636rzGzr19Tvb8jPn/S3R0dHS04z4E8HfR0dHR8XD5DaGC94/rdALY0dHx8XafAlj/wI60jofPb7foXh//edcJYEFH14gaCd0tDPXf0+rjHxbNT4f/6/fr0dHxkPsNnhb+DvnN07zYf1P7nt/8vTYf/3nzj8hvk/+B/GPT76DIfey3dHP8jN/8g/8JHR0dD7d/9Hv/Ixb87j/Hgt/5ZzSt+6dY8D/8Eyz47T+o4Pv4a/Q9v/H7/6L1x3+e80+m/134d/+tf/Iv8dv/9H/G7/zz/6Xs9/7ZvxS//8/+hfIDP/gD/P7v/z5+7/d+D7/7u7+L3/md3xELV16Poaj4+G2+mYSa261sPZ2Guv6M/h0IIsNQ/kyvo+/xt/D3/tJX6K1x3+ebBwuY0dwCrtG0rg4YseIUQeujFrRM27HjTGj6B+dEAMjY2J8fAQTE60YnBzD1NQ4ZmamxAL+S3c8vPgJ+zDjF9uXGYFmfuAobBopYpcB2D0D7MDDB6wVfB9/jb9HhYQfP34PFZa5Hv950mUDTpBTDmDAHsegI4FhRxRj7gSmXSFhdnqExeESbrCpP8boZaf4XAAIURULPh4A0h48FbpVveXsIZu1wwa6wdyNe4Xy+U+6H6nbj+nbkZ/CJkm0dL5Y/V583AMgbRymM+L4/nx22dKIkt41nNRBpbJzNN2T6dw86pHHZN57FzNI1dYxnsHc+JA1PF01Lfp6iR107RjNg3kb+r/ZMFwT+va7pUdpDi2azDuqOTJXF8otaJJSQrcNHBMrnPwCJwzVZwnFyJ3SeYer1pTuOVMYNCbgdnzh0Uhi/gRYB1Fwq7RTjEdGYD7G4H/FEAMlUiKbAg0DHg6cCuEZpM4Dz0Q2pjlkr6i0ibb1v3cP4+0304t4+BeyY5tuChqK2w5Bv2m5DQeyZoJBNVs30twDy91THTIWPY9hM/053AI9NaTh41Th+pw1a+FT8zpsr0H6XaDTI8btmS+02KykBtPgDEkB/0IcgjfSqw1cdv0QyqAVvkdCX1b3awT4ZQ/gw6g6gDumixqK2s6ZQtNUACV+U4WakN0tgNWqR4PtBLBadQzvpLEAOXoKx49HFuURHwXvorWC43eZpsGXHX1ccRzW05fF7UAetMAAjjnAIAR71cfCiLk3cK+IjH8XPT/ELIJU0dGLYCEd9C2A7U8hZYfIS00IH7DSUNBS1XcZi0/bMaPZPa+4WvrvFsNn4fRYBPGGojV8zAZT4EY6fCmB/sECjvyCckXDDANbHL5MNdWLYCNAngJ0AthbA+h2+udyAA/pETxhqASQo6eo7X5qyiuJp1uFxm80dLuk4pY/h4FQUUZ/rmgEQZr2Rni6G3Nr9PipAHL8sr1IJ4CdAHYC2AngvQWw2en3XAE80VOJXv10j/KUL6Nnr5D40bQA9rhLMv2tDiCP/qk8rS/u0VTFT43+OgH8HARw1UAJnwxilrsFsRzGoeIddQL4+fr5CeBc2wabdwA6q0ujy5BpysGZLA7P5HHUVMRjYwGn6PYM/Z30mkp153jkZ6Hw2TWXHTzdregmP56g30Ph8xYwGcjBEC7CHU7AF0sJfGvLYS7peFBkYv6ybdWgconONsB0ADsB/FIHUG2Ta+fxMpoz5HRA2Jpx2KT7F5i5JPE7ba5ET+H4XbRp4VPx63ZVcPyyuoFBXxFD/plEzxiFxm8fz0j8kskYMomQUNGr1gngPKo+HOazD0C92jCHjcM1swkEZZtHm/elD+CUZqdBKWEXH6zcpD0Gzf4pzYHJ35nVNaQ50Vz5WnzfrsFFHMdNkm3bEnMMxSwbHrVmKX0GcNRUofMwy85bSnAG85gVukwE/MEEbNA4CVpr+2iNAMJZDOFFAMp5AKpFENhYR+Yhf5MK+Mv68E8B0ADsB/BiHsF2Nashha8ZRS17id8KwXlLUaJoKRw/nvqWp7w86Nren09S5DAWCSR38hwBYuRWHxC+SLEr80skUcvGoKEQDNQHMhrxy2wngPADQIjg0292CqKwbxh11Avj59HkM40GZ5h0xaY5SYi5RyI7ZUzjuSDftlDMjzto1520ZXHTkyi4587jsKqDbUxI9NOK75tdC2D2qjvyEe+fnzsAeLCNIU0BIDohTBWAI0xc0hnyyglMgIROibw1TMOE+UA1657QSwE8B0AL/EAayJmrF5R82a4/Y8TjgKOMFBc+eadkZ3wZUXHLxqHL8r7qKET8WwN6C5EdJGfyNhSPycdBu0VvKXSEHiV0gVgWRWUxVAj1/R7+kEsBPATgA7Aw9gEdp5HfMQvGzQuJ3y1XCaU++Jwd19xpAR6gEN8WPR38qfsk0JH7FdAl15TSxsBbBkL92BHiV57T0d0geVp0AdgL4sAwQwyfxM+vxswFnaCB11py11fEOX+paRd0170aNdWtn/LydfFh8LGBES1+YxQ+Hv159Kmv18qA4nfrAAyHgXqEeTbBY02wLeicjhHreYjWvqS6wTwYQ7grjbtng5XKketGXunNQd01Xtz76Z6u98hY8VhU0V9/I6bNsd0sHeS4nfGqV6x6qUAXqegzS1QqNGnu+Upl2mEFZT0zoU3ubHRIaUxRCA936KIIBi1+cpCiAaQpgplEA41FNJKQJB+W2E8B51wnglzmARUTvfgQR32tBFDf7xSN/M66NBI/npaSuAd0FS12i1dv75Hl/G0jWrv0eMRn8Ijv+lw4wCqCPIUuGyBIMcvGp4dwPX0JG4Xv0DUck58S1c1Pc32pzHgyroc18qq4aLYs43F/q3ZXY5Ru6e8Ty99364ULZB15gU7dxTNS0TRS/TfMZWcm1pCu0ZNDUueyhx7J9k61RU9ey6JlDTCumbWYBFV5Kaq6HwZNACocF2WChXCYzo+WPkM3X+ez8yg8N2kqe9tItY4jdQm6HbGm4M5ULYDfJk1WIAtUIKdGmTxFXS1G1Y/hC1Qwd/PHNQvH90GeFYbKVDBikHcscjGciKfKKGQyI1SXF0kOTIRGIiH47K7YIN9M7dHXBdTBaYeV17X5zRooNBilCD8G04fyvZQwAcbmuMoKa30aKDVYHcvFhVeFz1UA4Vfwe1qDv811cP5m2VqVwWtBa1URw17nTB6T3Vh61770hw4b41cMduTAC7Hx+Tz7G
```

5. Open the online decoder and copy the response content to the form, then click "Convert the source data":

You can use this base64 sample decoder and encoder to:

- Decode base64 strings (base64 string looks like YTM0NZomIzI2OTsmlzM0NTueYQ==)
- Decode a base64 encoded file (for example ICO files or files from MIME message)
- Convert text data from several code pages and encode them to a base64 string or a file
- **New: Try [CSS/base64 analyzer](#) and simple [Base64 decoder and encoder](#).**

The Form.SizeLimit is 10000000bytes. Please, do not post more data using this form.

Type (or copy-paste) some text to a textbox bellow. The text can be a Base64 string to decode or any string to encode to a Base64.

```
Fn/53K0dH30424E0mK00dHk8Xb5D8dC947TgALY0dHx8X8tA1j/W100j0tFb7T0Xm7/edC9TEH14gac00C0FX1
0+rjHxbNT4F/6/fr0dHxkPsNnhb+DvnN07zYf1P7nt/8vTYf/3nzj8hvk/+B/GPt76DIfeY3dHP8jN/8g/8JHR0
dD7d/9Hv/Ixb87j/Hgt/5ZzSt+6dY8D/8Eyz47T+o4Pv4a/Q9v/H7/6L1x3+e80+m/134d/+tf/Iv8dv/9H/G7/
zz/6Xs9/7ZvxS//8/+hfiDP/gD/P7v/z5+7/d+D7/7u7+L3/md3xELV16PoaOj4+G2+mYsa261sPZ2Guv6M/h0I
IsNQ/kyvo+/xt/D3/tJX6K1x3+ebBwuY0dwCrtG0rg4YselUQeujFrRM27HjTGj6B+dEAMjY2J8fAQTE60YnBzD
1NQ4ZmamxAL+S3c8vPgJ+zDjF9uXGYfmfuAobBopYpcB2D0D7DMDB6wVfB9/jb9HhYQfp34PFZa5Hv950mUDTpB
TDmDAHsegI4FhRxRj7gSmXSfhdnqExeESbrCDPp8boZaf4XAAiURULPh4AOH48FbpVveXsIZu1wwWa6wdyNe4Xy
+U+6H6nbj+nbkZ/CJkm0dL5Y/V583aMgbRymM+L4/nx22dKIkt41nNRBpbJzNN2T6dw86pHHZN57FzNI1dYxnsH
c+JA1PF01Lfp6iR107RjNg3kb+r/ZMFwT+va7pUdpDi2azDuqOTJXF8otaJSQrcNHBmRnPWcJwzVZwnFyjs3eYE
rlpTuOVMYNCbgdnnhz0Uhi/gRYBiFwq7RTjiEdGYD7G4H/FEAM1UiKbAg0DHg6cCuEZpM4DzOQ2pj1kr6i0ibB1
v3cP4+0304t4+BeyY5tuChqK2w5Bv2m5DQeyZoJBNVsJ0twDy91THTIWPY9hM/053AI9NaTh41Th+pw1a+FT8zp
srOH6XaDTI8btmS+02KykBtPgDEkB/0IcgjfSqwlcdv0QyqAVwvkdCX1b3awT4ZQ/gw6g6gDumixqK2s6ZQtNUA
CV+U4WaKN0tgNWqR4PtBLBadQzvplEA0XoKx49HfuURHwXvorWC43eZpsGXHX1ccRZw05ff7UAetmAAjnAIAR71
cfCi1k3cK+T1H8XPT/ELT1U0delYCeD9C2A7U8b7YfiS00TH7DSUNBS1XcZi0/bMaP7Pa+4UwrvFsNn4fRYBPGG
```

or select a file to convert to a Base64 string.

Выберите файл Файл не выбран

Convert the source data

What to do with the source data:

- ☐ **encode** the source data to a **Base64** string (base64 encoding)
Maximum characters per line:
- ☒ **decode** the data from a **Base64** string (base64 decoding)

Output data:

- ☐ output to a textbox (as a string)
- ☒ export to a binary file, filename:

6. Here is the snapshot we have received:

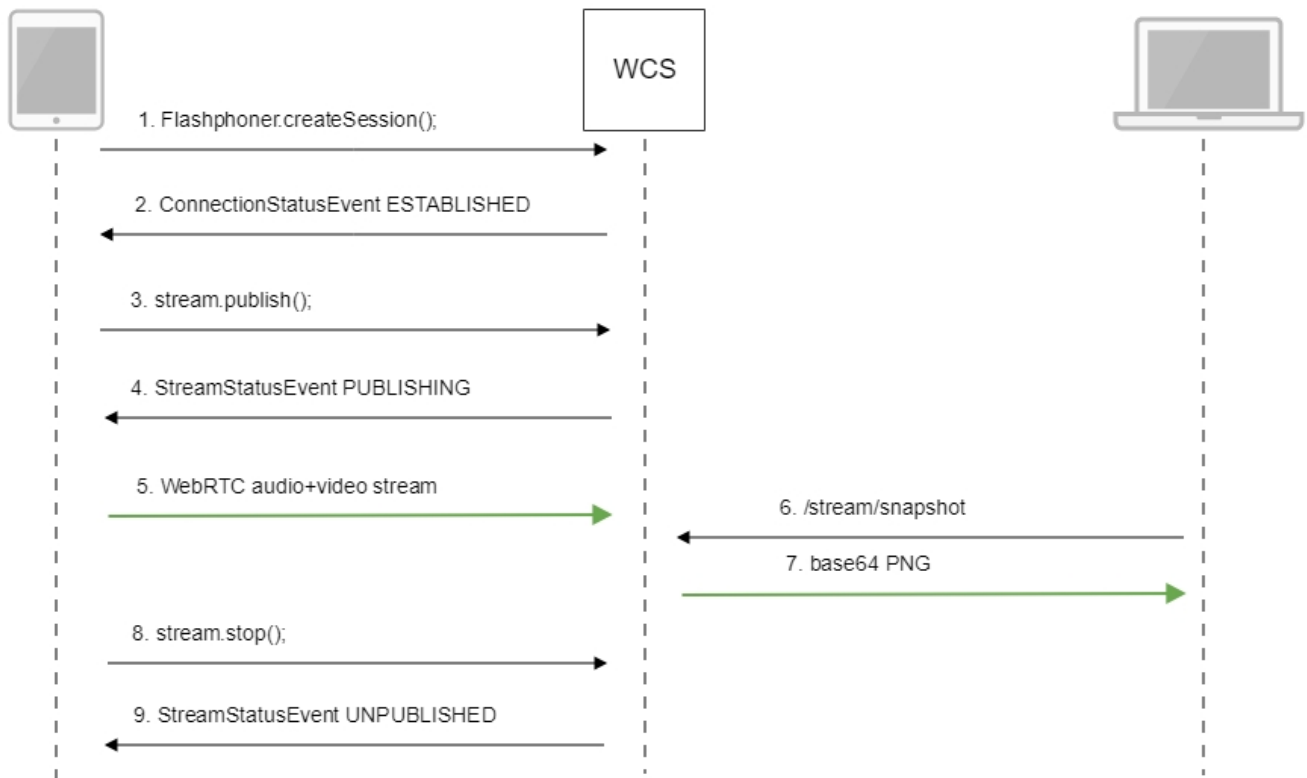


Call flow

Below is the call flow when using the Stream Snapshot example to publish the stream and take a snapshot

[stream-snapshot.html](#)

[stream-snapshot.js](#)



1. Establishing a connection to the server.

`Flashphoner.createSession();`[code](#)

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


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

ConnectionStatusEvent ESTABLISHED`code`

```
Flashphoner.createSession({urlServer: url}).on(SESSION_STATUS.ESTABLISHED, function(session){
    //session connected, start streaming
    startStreaming(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(publishStream){
    setStatus(STREAM_STATUS.PUBLISHING);
    onPublishing(publishStream);
}).on(STREAM_STATUS.UNPUBLISHED, function(){
    ...
}).on(STREAM_STATUS.FAILED, function(stream){
    ...
}).publish();
```

5. Sending the audio and video stream via WebRTC

6. Taking a snapshot of the broadcast. A new stream is created from the published one specially to take a snapshot.

stream.snapshot();`code`

```
function snapshot(name) {
    setSnapshotStatus();
    var session = Flashphoner.getSessions()[0];
    session.createStream({name: name}).on(STREAM_STATUS.SNAPSHOT_COMPLETE, function(stream){
        console.log("Snapshot complete");
        setSnapshotStatus(STREAM_STATUS.SNAPSHOT_COMPLETE);
        snapshotImg.src = "data:image/png;base64,"+stream.getInfo();
        //remove failed callback
        stream.on(STREAM_STATUS.FAILED, function(){});
        //release stream object
        stream.stop();
    }).on(STREAM_STATUS.FAILED, function(stream){
        setSnapshotStatus(STREAM_STATUS.FAILED);
        console.log("Snapshot failed, info: " + stream.getInfo());
    }).snapshot();
}
```

7. 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);
    ...
}
```

8. Receiving from the server an event confirming unpublishing the stream.

StreamStatusEvent, status UNPUBLISHED[code](#)

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

Automatic stream snapshot taking

If necessary, snapshots for every stream published of supported format can be taken automatically. This feature can be enabled with the following parameter in [flashphoner.properties](#) file

```
snapshot_auto_enabled=true
```

Snapshot pictures placement can be set with the following parameter

```
snapshot_auto_dir=/usr/local/FlashphonerWebCallServer/snapshots
```

In this folder, subfolder will be created for every stream. The subfolders name is formed from stream mediasession identifier (by default_

```
snapshot_auto_naming=mediaSessionId
```

or stream name

```
snapshot_auto_naming=streamName
```

Snapshot pictures are consistently numbered and are created periodically with the following setting

```
snapshot_auto_rate=30
```

In this case, snapshot will be created from every 30 frame.

To save disk space, snapshot pictures amount can be limited using the following parameter

```
snapshot_auto_retention=20
```

In this case, last 20 snapshot pictures will be stored in stream subfolder.

Snapshot pictures numeration will be continued if stream with same name is published.