

# Taking a PNG snapshot of the stream

## Overview

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

## Supported protocols

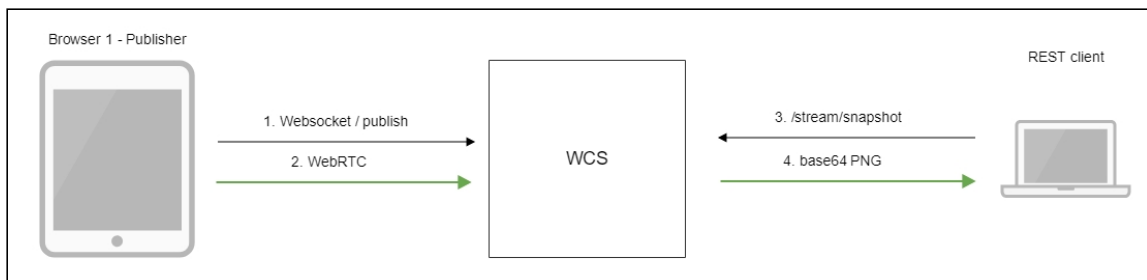
- WebRTC
- RTMP
- RTSP

## Supported snapshot formats

- PNG

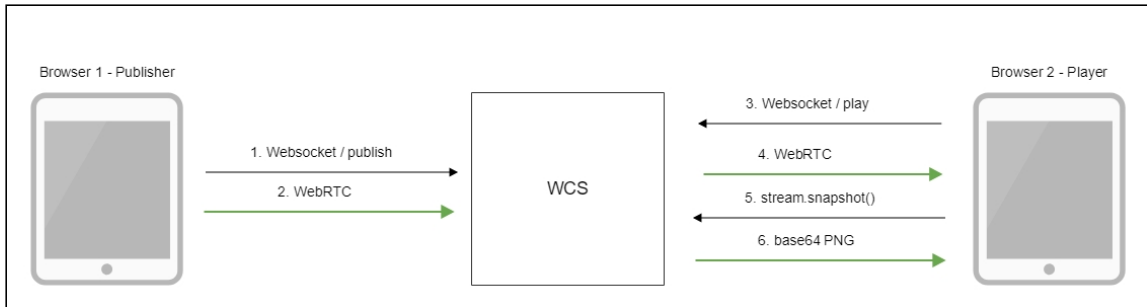
## Operation flowchart

### 1: Using the REST query



1. The browser connects to the server via the Websocket protocol and sends the `publishStream` 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 `publishStream` 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 `playStream` 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/snapshot` REST 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	Request body	Response body	Response status

REST-method	Request body	Response body	Response status
<code>/stream/snapshot</code>	<pre>{   "streamName":   "64966f33" }</pre>	<pre>{   "data":   "iVBORw0KGgoAAAANSU   hEUgAAAUAAAADwCAY   AAABxLb1rAAAACXBIWX   MAAAAAAAAAAQCEeRd   zAAAQA..." }</pre>	200 OK 404 Stream not found 500 Internal server error

## Parameters

Parameter	Description	Example
streamName	Unique stream name	<code>`64966f33`</code>
data	Snapshot file encoded to base64	<code>`iVBORw0KGgoAAAANSU hEUgAAAUAAAADwCAY AAABxLb1rAAAACXBIWX MAAAAAAAAAAAQCEeRd zAAAQA...`</code>

## 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:
2. the demo server `demo.flashphoner.com`;
3. the Chrome browser and the [REST-client](#) to send queries to the server;
4. the [Two Way Streaming](#) web application to publish the stream;
5. the <https://www.motobit.com/util/base64-decoder-encoder.asp> service to decode the snapshot.
6. 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**

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

Method POST Request URL <http://p11.flashphoner.com:9091/rest-api/stream/snapshot> SEND

Parameters

Body content type application/json Editor view Raw input

FORMAT JSON MINIFY JSON

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

8. Make sure the response is received:



9. 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 YTM0NZomlzi2OTsmIzMOntueYQ==)
- 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 below. The text can be a Base64 string to decode or any string to encode to a Base64.

```
0+rjHxbNT4F/6/fr0dHxkPsnhb+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+/d+d7/7u7+L3/md3xELV16Poa0j4+G2+mYsa261sPZ2Guv6M/h0IIsNQ/kyvo+/xt/D3/tjX6K1x3+ebBwuYodwCrtG0rg4YseLUQeuJfRm27HjTGj6B+dEAMjY2J8fAQTE60YnBzD1NQ4ZmamxAL+S3c8vPgJ+zDjF9uXGYfmfuaobBopYpcB2D0D7DMDB6wVfB9/jb9HhYQfp34PFZa5Hv950mUDTpBTDmDAHseg14FhRrJ7gSmXSFhdnqExeESbrCDp8boZaf4XAAiURULPh4A0h48FbpVveXsIZu1wWw6wdyNe4Xy+U+6H6nbj+nbkZ/Cjkm0dL5Y/V583aMgbRymM+L4/nx22dKIkt41nNRBpbJzNN2T6dw86pHHZN57FzNI1dYxnsHc+JA1PF01Lfp6iR107RjNg3kb+r/ZMFwT+va7pUdpD12azDuqOTJXF80taJ5QrcNHBmRnPWcJwzVzwnFyJ53eYer1pTOVMYNCbgdnnhz0Uhi/gRYBiFwq7RTjiedGYD7G4H/FEAM1UiKbAg0DHg6cCuEZpM4Dz0Q2pj1kr6i0iB1v3cP4+0304t4+BeyY5tuChqK2w58v2m5DQeyZ0JBVnsJ0tWdy91THTIWPY9hM/053AI9NaTh41Th+pw1a+FT8zpsr0H6XaDTI8btmS+02KykBtPgDEk8/0Icgjfsqw1cdv0QyAVvWkdCX1b3awT4ZQ/gw6g6DumiXqK2s6ZQtNUACV+U44waKN0tgNlqR4PtBLBadQzvp1EA0XoKx49HfUURHwXvorWC43eZpsGXH1ccRZw05ff7UAetMAAjnAIAR71cfcCi1k3cK+TJH8XPT/ELI3U0dglYCEd9C2A7Uj8h7YfIS00TH7DSUMBS1Xc7i0/hMaPZPa+4wvrvFsNp4fRYBP6Gg0jV8zA2T4EY6fcmB/sEJCjYcckXDDANBHL5MNdWLYCWAng30Athba+h12+udyAA/pETXhqASQ6oe47X5qy1u7p1uFXM80Luk4p/h4FQUUZ/rmgEQzr2RniI663N9PipAHL8sr1I34CdAhYQ2KAngw2en3XAE80V0JXv10j/KU16Nhr5JkzQ69rhLMv2TD1CP/qk8rS/u0VTFF43+OgH8ARw1UAJnwx1lrsFsRzGoeIdDQL4+FR5CEbc2wabdwA6q0ujy5BpysGLA7P5HHUVRJYwgn6PYH/230mk153jzK26H2TWXHTzdrmgP56g30P8hXyGcJBEc7CvH07f083kYMomQUNGR1gngPKo+H0azD0C92jCjYcmIswKEZTzHm/e1D+CUZqdBKwEXH6zcpD0GzF4pzYH35Nva050Vz5WnzfzFFHMdNm3BEnMMx5wbHrVmX0GcNR0uFmly85bSnaG85gVukwE/MEEBna4Cvpr+21NAMJZD0FFAmp5AKpFENHYR+Yhf5MK+Mv68E8B0ADsB/8IhsF2NAshas8ZRS17iD8Klwx1LUajokRw/nvqWp786nNren009SwdAMC8R38hwYuwRwHxC+5LEr80skUcvGokEQDNQHmhrxy2wngPadQIjg0292Cqkwbxh11Avj59Hk40G25h0axY5yISryI7ZUzjUSDFtLDHjzt01520ZXHTkyi45877jsKqDbUxI9NOK75tdcD2qjvyEe+fnszAeLCNIUOBIDohTBwAI0xc0hnyyg1lgR0i0w1TMOE+UA1657Q5wE88B0AL/EAayJmr58R2a+/y87jgKOMFbc+eadkZ3wXUXHxqHl8r7qkET8wN6C5ED3GfyHSpycdBUoUuXXSEHIV0VgWuRwUvAJj/1R7+kesBPATa7AWw9gEdp5HFHQvG2QuJ3yLXCau++w10xpAR6gENBPR38qfsk0JH7fdal15TSxsBbBkL92BH1v57T0d0geVp0AdGL4sAmQwyfxH+vxsuFnaCB111py11fEOX+paR0170aNdWtd/LydfH8LGBES1+Xq+Hv159Kmv18qA4nfrAayHgX8EebY02wLeicjHreYjVwv56wTwyQ7grjbtng5XkktGXunQd01Xtz76Z6u98Y8VhU009/I6bnS054E54nFGvG6qUAUqegzS1QnGnu+Upl2mEFZT0z0U3ubHRiMAUxRCA936K1IB1+cpCiAaQpplEA41FNJQ08W4E28B51wng1zmArUtvfgeQR32t8FD7X5N/M66NB1/npaSuva0dF512i1dv75H1/GOjWrv0EmRn8Ijv+1w4wCqCPIUuGvYIMcvGp4dPX0JG4Xv0DUck58S1c1Pc32pzGyrocl8qq44L543F/q3ZFXSRu687y99364ULZB15gU7dxtNS0TRS/TfMzWcmLpcU0ZNDUueYhx739k61RU9ey6WJLdtCumbwBFV5Kaq6HWZNAcocF2WChXCYzo+HPK3X+ez8yG8N2kqe9t1tY4d0m6H6m4M5ULYdfJkLWIATUIKdGmTxFXS1G1Y/hC1Qwd/PNHQvH90geFyBKVDbkHcscjGc1KFKGQyI1EXFOkOTIRGI3H47K7YIN9M7d0Xh8DTRaYeV17X5z8ocNR1iCD864FyXzQwAqchmuKq0KaQKDXHvEhVeFz1U44Vfwe1eDv81UcP5m2W6wWt8R1U17nTR6T3Vh6177D0wAhiH8dVtAC7Hx+TzJ6
```

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:

10. Here is the snapshot we have received:

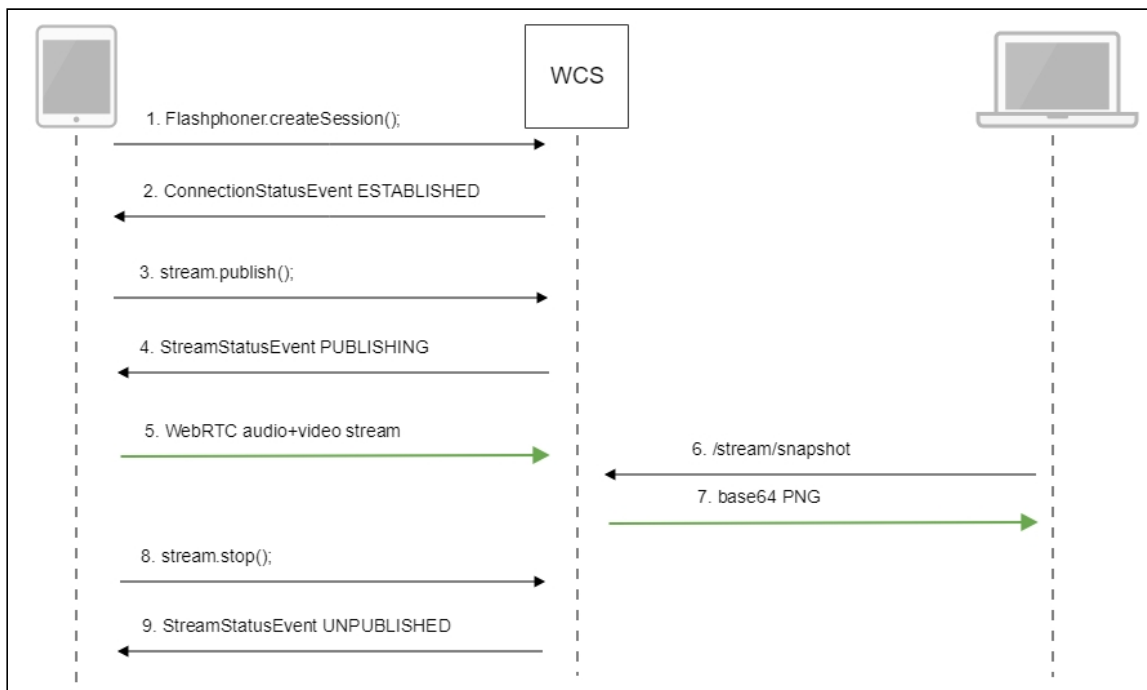


## Call flow

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

`SESSION_STATUS.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

`STREAM_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

`STREAM_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.