

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](#)
- [JavaScript API](#)
- [Quick manual on testing](#)
- [Call flow](#)

Overview

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

Supported protocols

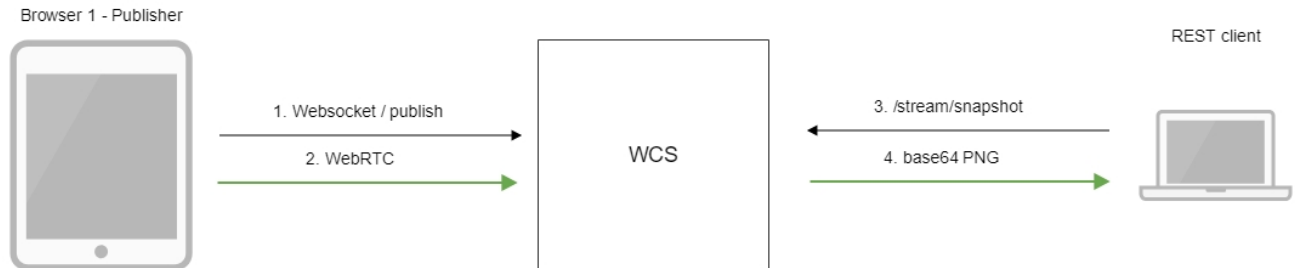
- WebRTC
- RTMP
- RTSP

Supported snapshot formats

- PNG

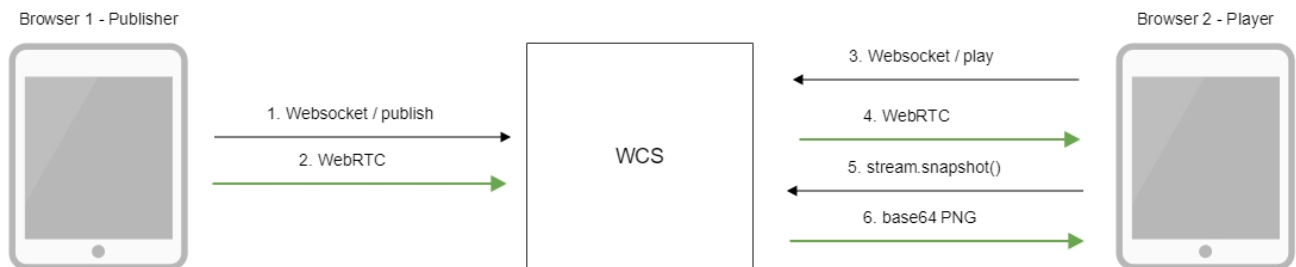
Operation flowchart

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/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	Example of the REST query	Example of the REST response body	Response statuses
<code>/stream/snapshot</code>	<pre>{ "streamName": "64966f33" }</pre>	<pre>{ "data": "iVBORw0KGGoAAAANSUhEUgAAAUAAAADwCAYAAABxLb1rAAAACXBjWMAAAAAAAAAAAQCEeRdzAAAQAE1E QVR4n0zd95Pcd37feVjhrFKwXT6fr+r+gPvh6nx1V5bvLMt3liXbkiWtrF1v105qVxu4icucwUyABAMIA gSIQ0QcBzlnzAADDIDJuadzzjmH173f7+/30216g04GiCHIZtwJeqZneJAu5 /9+XzD57tgwYIF60jo6PiSmvdfok0jo20+zPsv0NHR0TFf5v0X60jo6Jgv8 /4LdHR0dMyXef8F0jo60ubLvP8CHR0dHfN13n+Bjo60jvky779AR0dHx3yZ91+go60jY77M+y/Q0dHRMV /m/Rfo60jomC/z/gt0dHR0zJd5/wU60jo65su8 /wIdHR0d82Xef4G0jo60+TLvv0BHR0fHfJn3X6Cjo6Njvsz7L9DR0dExX+b9F+jo60iYL /P+C3R0dHTM13n/BTo60jrmY7z/Ah0dHR1t+Ue636rzGzr19Tv8jPn /S3R0dHS04z4E8HfR0dHR8XD5DaGC94/rdALY0dHxBXafAlj/wI60jofPb7foXh //edcJYEFHl4gaCd0tDPXf0+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/md3xELV16Poa0j4" }</pre>	200 - Snapshot is taken 404 - Stream not found

Parameters

Parameter name	Description	Example
<code>streamName</code>	Unique stream name	64966f33
<code>data</code>	Snapshot file encoded to base64	iVBORw0KGGoAAAANSUhEUgAAAUAAAADwCAYAAABxLb1rAAAACXBjWMAAAAAAAAAAAQCEeRdzAAAQA

Sending the REST query to the WCS server

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

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



Player



PUBLISHING

ESTABLISHED

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

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

SEND

Parameters

Headers

Body

Variables

Body content typeapplication/jsonEditor viewRaw input

FORMAT JSONMINIFY JSON

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

4. Make sure the response is received:

200 OK458.60 msDETAILS

```
{  "data": "iVBORw0KGgoAAAANSUHEUgAAAAAADwCAYAAABxLb1rAAACXBIXMMAAAAAAAAAAAQCEeRdzAAAE1EQVR4nOzd95Pcd37feVjhrFKwXT6fr+r+gPvh6nx1V5bvLMt31iXbkiWtrF1v105qVxu4iucucUyABAMIAgSIQOQcBz1nzAADDIDJuaazzjmH173f7+/302l6g046iCHIZtwjeqZnejaAU5/9+XzD57tgwYIF60jo6PiSmvdfok0jo20+zPsv0NHR0TFf5v0X60jo6Jgv8/4LdHR0dMyXef8FOjo60UblvP8CHR0dHfN13n+Bjo60jvky779AR0dHx3yZ91+go60jY77M+y/Q0dHRMV/m/Rfo60jomC/z/gt0dHR0zJd5/wU60jo65su8/wIdHR0d82Xef460jo60+TLvv0BHR0dHfJn3X6Cjo6Njvsz7L9DR0dExx+b9F+jo60iYL/P+C3R0dHTM13n/BTo60jrmjy7z/Ah0dHR1t+Ue636rzGzr19Tv8jPn/S3R0dHS04z4E8HfR0dHR8XD5DaGC94/rdALY0dHx8XafA1j/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/md3xELV16Poa0j4+G2+mY5a261sPZ2Guv6M/h0IIsNQ/kyvo+/xt/D3/tJX6K1x3+ebBwuY0dwCrtG0rg4YselUQeujFrRM27HjTGj6B+dEAMjY2J8FAQE60YnBzD1NQ4ZmamxAL+S3c8vPgJ+zDjF9uXGYfmfuaObBopYpcB2D0D7MDDB6wVfB9/jb9HhYQfP34PFZa5Hv950mUDTPBTDmDAHsegI4FhRxRj7gSmXSfHdnqExeESbrcdPp8boZaf4XAAiURULPh4A0h48fbPvveXsIZu1wWla6wdyNe4Xy+U+6H6nbj+nbkZ/Cjk0dL5Y/V583aMgbRymM+L4/nx22dKIkt41nNRBpbJzNMN2T6dw86pHHZN57FzNI1dYxnsHc+JA1PFO1Lfp6iR107RjNg3kb+r/ZMFwT+va7pUdpD12azDuqOTJXF8otaJ5QrcNHBMrnPwCJwzVZwnFyJ53eYerlpTuOVMYNCbgdnzhz0Uhi/gRYB1Fwq7RTjiEdGYD7G4H/FEAMlUiKbAg0DHg6cCuEZpM4Dz0Q2pj1kr6i0ib1v3cP4+0304t4+BeyY5tuChqK2w5Bv2m5DQeyZoJBVNsJ0twdY91THTIWPY9hM/053AI9NaTh41Th+pw1a+FT8zpsrOH6XaDTI8btmS+02KykbTpgDEkB/0IcgjfsQw1cdv0QyQAVvvdCX1b3awT4ZQ/gw6g6gDumixqK2s6ZQtNUACV+U4WakN0tgNWqR4PtBLBadQzvp1EAOXoKx49HfUURHwXvorWC43eZpsGXHX1ccrZw05FF7UAetmAAJnAIAR71cfCilk3cK+IJH8XPT/ELIJU0dLgYCEd9C2A7U8hZYfiS00IH7DSUNBS1XcZi0/bMaPZPa+4WwrvfSnn4fRYBPGGojV8zAZT4EY6fCmB/sECjvyCckXDDANbHL5MNdWLYCwAngJ0AthbA+hi2+udyAA/pETxhqASQo6eo7X5syiu/jPluFxm80dLUk4pY/h4FQUUZ/rmgEQZr2Rni6G3Nr9PipAHL8sr1IJ4CdAHYC2AngvQWw2en3XAE80V0JXv10j/KUL6Nnr5D40bQA9rhLMv2tDiCp/qK8rS/u0VTFT43+OghH8ARw1UAJnwxilrsFsRzGoeIddQL4+fr5CeBc2wabdwA6q0ujy5BpysGZLA7P5HUVUVRJYwGn6PYM/Z30mkp153jkZ6Hw2TWXHTzdrregMP56g30Ph8xYwGcjBEC7CHU7AF0sjFgvLYS7peFBkYv6ybdWgconONsB0ADsB/FIHUG2Ta+fxMpoz5HRA2Jpx2KTF75i5JPE7ba5ET+H4XbRp4VPx63ZVCPyuuofBXxFD/pLEzxifxm8fz0j8kskYMomQUNGr1gngPKo+HOazDOC92jCHjcM1swkEZZtHm/e1D+CUZqdBKWEXH6zcpD0Gzf4pzYHJ5nVNaQ50Vz5WnzfrsFFHMDNkm3bEnMMxSwbHrVmXK0GcNRUoFmWy85bSnAG85gVukwE/MEEbNA4Cvpr+2iNAMJZDOFFAMP5AKpFENhYR+Yhf5MK+Mv68E8B0ADsB/BiHsF2NAssha82RS17id8KlWwx1LUajokRw/nvqWp7w86nlnren009SndAwCSR38hwBYuRWHXc+SLER80skUcvGoKEQDNQHhrxy2wngPADQIJg0292CqKwbxh11Avj59HkM40GZ5h0xaY5SyI5RyI7ZUzjuSDft1DMjzto1520ZXHTkyi4587jsKqDbUxI9NOK75tdcD2qjvyEe+fnzsAeLCNIU0BIDohTBWAI0xc0hnyyglMgIROibw1TMoE+UA1657QSwE8B0AL/EAayJmrFSR82a4/Y8TjgKOMfBc+eadkZ3wZUXHLxqHL8r7qKET8wvN6C5EdJGfyNhSPycdBu0VukXSEhIv0gVgWRWUxVAj1/R7+kEsBPATgA7Aww9gEdp5HfNQvGzQuJ3y1XcAu++Jwd19xpAR6gEN8wPR38qfsk0JH7fDAlI5TSxsBbBkL92BhiV57T0d0geVp0AdgL4sAwQwyfxM+vxswFnaCB11lpy11fEOX+paRd0170aNdWtn/LydfFh8LGBES1+YxQ+Hv159Kmv1l8qA4nfrAAyHgXqEeTbBY02wLeicjhHreYjvwqS6wtYQ7grjbtng5XKketGXunNQ01Xtz76Z6u98hY8VhU0V9/I6bnSdo5HeS4nfGqVgX6qUAXqegzS1QqNGnu+Up1t2mEFZTOzoU3ubHRIAMuXRCA936KIIBil+cpCiAaQpgplEA41FNJKQJB+W2E8B51wng1zmArUTvfgeQR32tBFDf7xSN/M66NB1/npa5vuAd0FS12i1dv75H1/G0jWrv0EMRn8Ijv+1w4wCqCPIUuGYbIMcvGp4dwPX0JG4Xv0DUck58S1c1Psc32pzhGyrocl8qq4aLYs43f/q3ZXY5Ru6e8Ty99364ULZB15gU7dxtNS0TRS/TfM2wCm1pCu0ZNdUuEyhX739k61RU9ey6WJLDtCUmbWYBFV5Kaq6HWZNAccf2WChXCYzo+WPkM3X+ez8y8N2kqe9tItY4jdQm6HbGm4M5ULyDfJkLWIAtUIKdGmTxFXS1G1Y/hC1Qwd/PHNQVH90GeFYbKVDbikHcsjGCiKfKKGQyIL5XF0kOTIrRGIiH47K7YIN9H7d0hX8dTRaYeV17XSzB0oMRi1CD8G04fvXz0wAchmuMoKnj0aKDVXHCvEbVeFz1U4AqFwe1gDv811cP5m2VvVwVtBaURW17nTB6T3Vh61770bw4b41cMdvTAC3Hx4Tz76
```

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+zDjF9uXGYfmuAobBopYpcB2D0D7DMDB6wVfB9/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/0IcgjfSqw1cdv0QyqAVwvkdCX1b3awT4ZQ/gw6g6gDumixqK2s6ZQtNUA
CV+U4WaKN0tgNWqR4PtBLBadQzvplEAOXoKx49HfuURHwXvorWC43eZpsGXHX1ccRZw05ff7UAetmAAjnAIAR71
cfCi1k3cK+T1H8XPT/ELT1U0delYCeD9C2A7U8b7YfiS00TH7DSUNBS1Xc7i0/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:

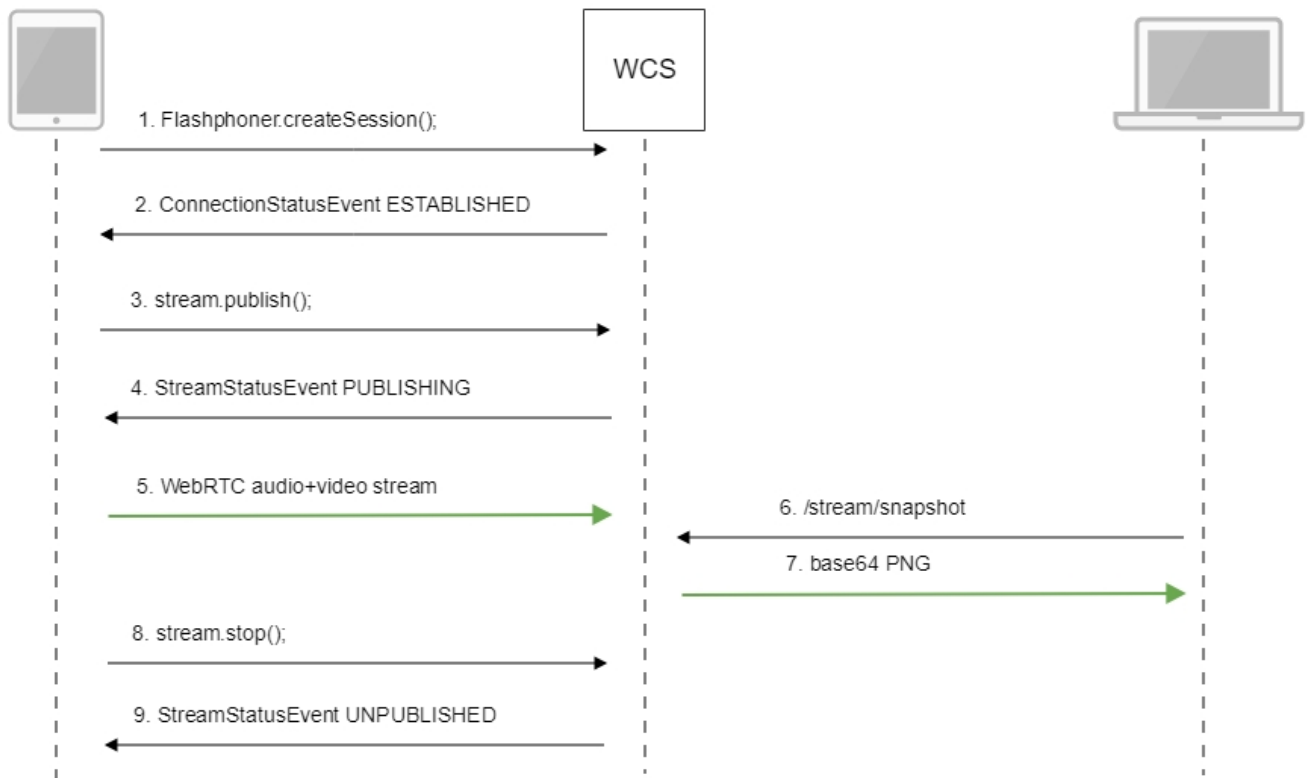


Call flow

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();

```