To another RTMP server

Overview

Upon request, Web Call Server converts a WebRTC audio and video stream to RTMP and sends it to the specified RTMP server. This way you can run a broadcasting from a web page toFacebook,YouTube Live,Wowza,Azure Media Servicesand other live video services.

Republishing of an RTMP stream can be made usingREST queriesor JavaScript API.

Supported platforms and browsers

	Chrome	Firefox	Safari 11	Edge
Windows	+	+		+
Mac OS	+	+	+	
Android	+	+		
iOS	-	-	+	

Supported codecs

- Video: H.264
- Audio: AAC, G.711, Speex 16

RTMP server authentication

Supported. Specify the name and password in the URL of the server, for examplertmp://name:password@server:1935/live

Operation flowchart

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 the /push/startup query from the browser.
4. TheWCS server publishes the RTMP stream on the RTMP server at the URL specified in the query.
5. The WCS server sends the RTMP stream.

REST queries

Republishing a video stream to another server can be performed using REST queries.

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

- HTTP:http://streaming.flashphoner.com:8081/rest-api/push/startup
- HTTPS:https://streaming.flashphoner.com:8444/rest-api/push/startup

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
- /push/startup- is the REST-method used

REST-methods and response statuses

REST-method	Example of REST query body	Example of response	Response statuses	Description
/push /startup	`{ "streamName": "name", "rtmpUrl": "rtmp://localhost:1935/live", "options": {} }`	`{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t", "streamName": "rtmp_name", "rtmpUrl": "rtmp://localhost:1935/live", "width": 320, "height": 240, "muted": false, "soundEnabled": false, "options": {} }`	409 - Conflict 500 - Internal error	Creates a transponder that subscribes to the given stream and sends media traffic to the specified rtmpUrl. The name of the stream specified in the query can be the name of an already published stream or the name reserved when the SIP call was created (to send media traffic received from SIP). If a transponder for the given stream and rtmpUrl already exists, 409 Conflict is returned.
/push/find	`{ "streamName": "name", "rtmpUrl": "rtmp://localhost:1935/live", }`	`[{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t", "streamName": "rtmp_name", "rtmpUrl": "rtmp://localhost:1935/live", "width": 320, "height": 240, "muted": false, "soundEnabled": false, "options": {} }]`	404 - Transponder not found 500 - Internal error	Find transponders by a filter

/push /find_all		[{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t", "streamName": "rtmp_name", "rtmpUrl": "rtmp://localhost:1935/live", "width": 320, "height": 240, "muted": false, "soundEnabled": false, "options": {} }]	404 - Not found any transponder 500 - Internal error	Find all transponders
/push /terminate	{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t" }		404 - Not found transponder 500 - Internal error	Terminate operation of the transponder
/push /mute	{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t" }	void	404 - Not found transponder 500 - Internal error	Turn off audio
/push /unmute	{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t" }	void	404 - Not found transponder 500 - Internal error	Turn on audio
/push /sound_on	{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t" "soundFile": "test.wav" "loop": true }	void	404 - Not found transponder 404 - No such file 500 - Internal error	Insert audio from a RIFF WAV file located in the /usr/local/FlashphonerWebCallServer/media/ directory on the WCS server
/push /sound_off	{ "mediaSessionId": "eume87rjk3df1i9u14elffga6t" }	void	404 - Not found transponder 500 - Internal error	Stop inserting audio from the file

Parameters

Parameter name	Description	Example
streamName	Name of the republished stream	streamName

rtmpUrl	URL of the server the stream is republished to	rtmp://localhost:1935/live
options	Transponder options	{"action": "mute"}
mediaSessionId	Unique identifier of the transponder	eume87rjk3df1i9u14elffga6t
width	Image width	320
height	Image height	240
muted	Is sound muted	true
soundEnabled	Is sound enabled	true
soundFile	Sound file	test.wav
loop	Loop playback	false

Theoptionsparameter can be used to turn off audio or insert audio from a file when creating a transponder.

Example,

```
"options": {"action": "mute"}
"options": {"action": "sound_on", "soundFile": "sound.wav", "loop": true}
```

Sending the REST query to the WCS server

To send the REST query to the WCS server, use aREST-client.

JavaScript API

Using WebSDK you can republish a stream to an RTMP server upon creation, similar to theSIP as streamfunction. Usage example for this method is available in the WebRTC as RTMP web application.

webrtc-as-rtmp-republishing.html

webrtc-as-rtmp-republishing.js

1. When a stream is created, the method session.createStream() receives the parameter rtmpUrl that specifies the URL of the RTMP server that accepts the broadcast. The name of the stream is specified in compliance with rules of the RTMP server.

code:

```
function startStreaming(session) {
    var streamName = field("streamName");
    var rtmpUrl = field("rtmpUrl");
    session.createStream({
        name: streamName,
        display: localVideo,
        cacheLocalResources: true,
        receiveVideo: false,
        receiveAudio: false,
        rtmpUrl: rtmpUrl
        ...
    }).publish();
}
```

Republishing of the stream starts directly after it is successfully published on the WCS server.

Server configuration

When WCS creates an RTMP transponder it automatically adds a prefix to the republished stream as set in theflashphoner.propertiesfile:

```
rtmp_transponder_stream_name_prefix=rtmp_
```

If the server the stream is republished to has certain requirements to the name (Facebook,Youtube), this line must be commented out.

The option

```
rtmp_transponder_full_url=true
```

turns on a possibility to pass some request parameters to RTMP server.

Parameters passing in server URL

It is possible to pass some parameters to server. to which a stream should be republished. Parameters to pass are specified in server URL, e.g.

```
rtmp://myrtmpserver.com:1935/app_name/?user=user1&pass=pass1
```

or, if a stream supposed to be published to a specified instance of RTMP server application

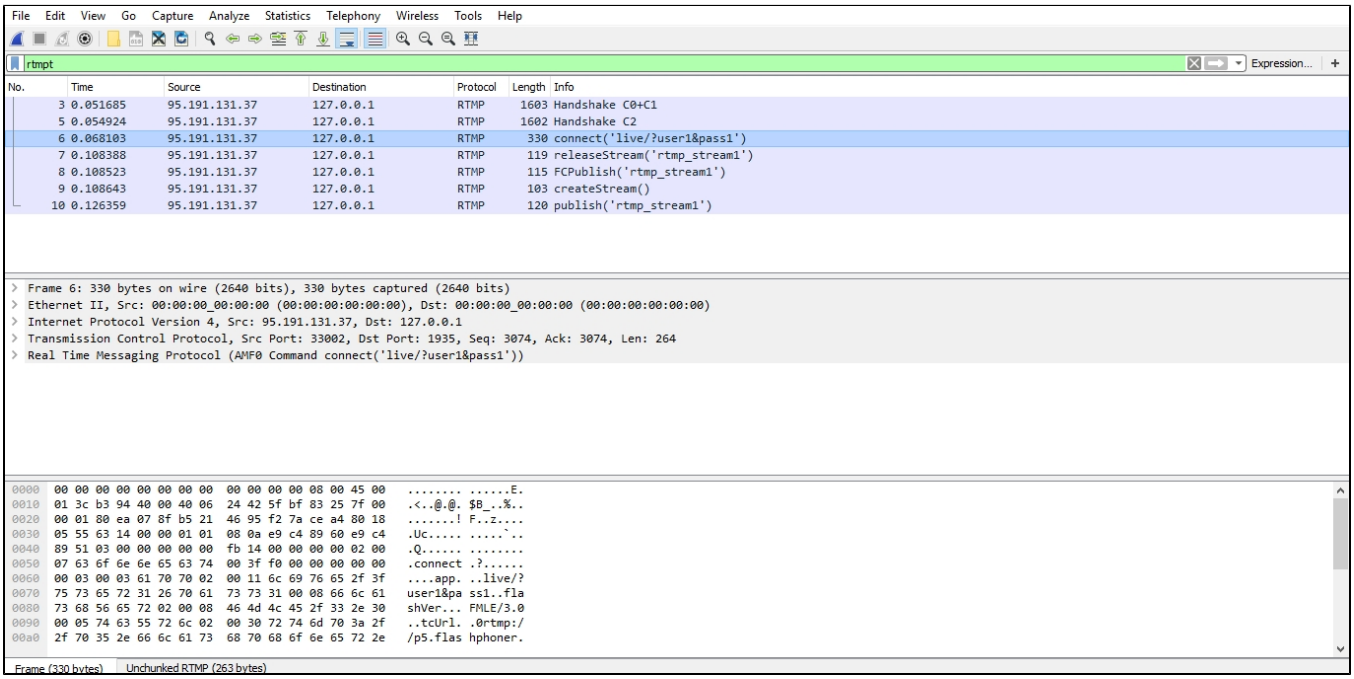
```
rtmp://myrtmpserver.com:1935/app_name/app_instance/?user=user1&pass=pass1
```

Where

- myrtmpserver.com is the RTMP server name
- app_name is the application on the RTMP server name
- app_instance is the instance name of the RTMP server application

Stream name is set in REST query /push/startup parameter 'streamName' or in corresponding stream creation option.

This is the example on RTMP connection establishing with query parameters passing



Stream name passing in server URL

In some cases, a stream publishing name shoukd be passed in the server URL. To do this, the following option must be set inflashphoner.propertiesfile

```
rtmp_transponder_full_url=true
```

Then, the URL to publish should be set in REST query /push/startup 'rtmpUrl' parameter or in corresponding stream creation option like this:

```
rtmp://myrtmpserver.com:1935/app_name/stream_name
```

or, to publish to another application instance

```
rtmp://myrtmpserver.com:1935/app_name/app_instance/stream_name
```

In this case, 'streamName' parameter or REST query /push/startup or corresponding stream creation option is ignored.

Automatic republishing to a specified RTMP server

WCS server can automatically republish all the streams published to a specified RTMP server. To activate this feature, set the next options inflashphoner.propertiesfile:

```
rtmp_push_auto_start=true
rtmp_push_auto_start_url=rtmp://rtmp.server.com:1935/
```

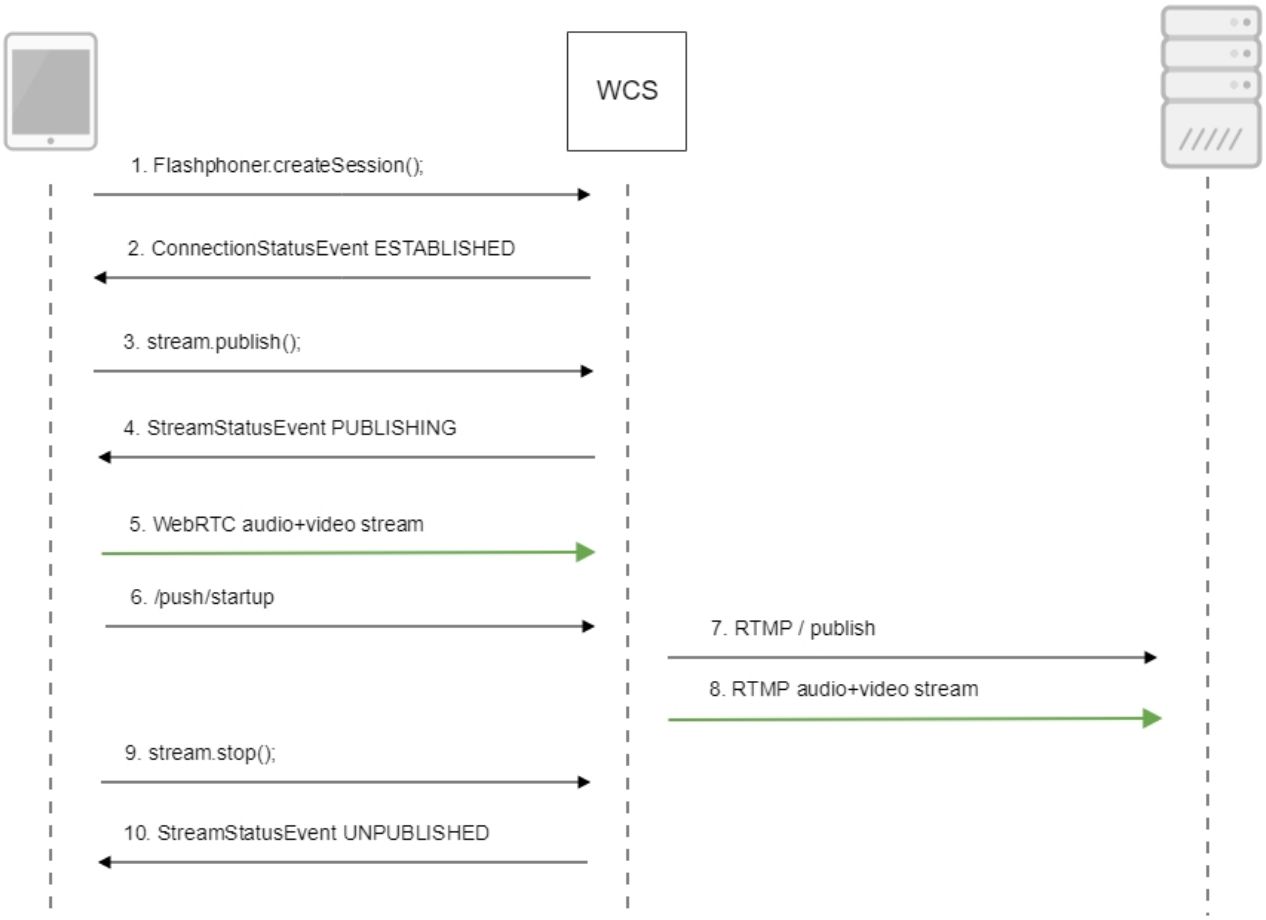
wherertmp.server.com is RTMP server name to republish all streams from WCS

Call flow

Below is the call flow when using the Two Way Streaming example to publish a stream and the REST client to send the /push/startup query:

two_way_streaming.html

two_way_streaming.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 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. Publishing the stream.

stream.publish();code

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

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

StreamStatusEvent, status 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. Sending the audio-video stream via WebRTC

6. Sending the /push/startup query

```
http://demo.flashphoner.com:9091/rest-api/push/startup
{
 "streamName": "testStream",
 "rtmpUrl": "rtmp://demo.flashphoner.com:1935/live/testStream"
}
```

7. Establishing a connection via RTMP with the specified server, publishing the stream

8. Sending the audio-video stream via RTMP

9. 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("");
}
```

10. Receiving from the server an event confirming unpublishing of the stream.

StreamStatusEvent, status 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();
```