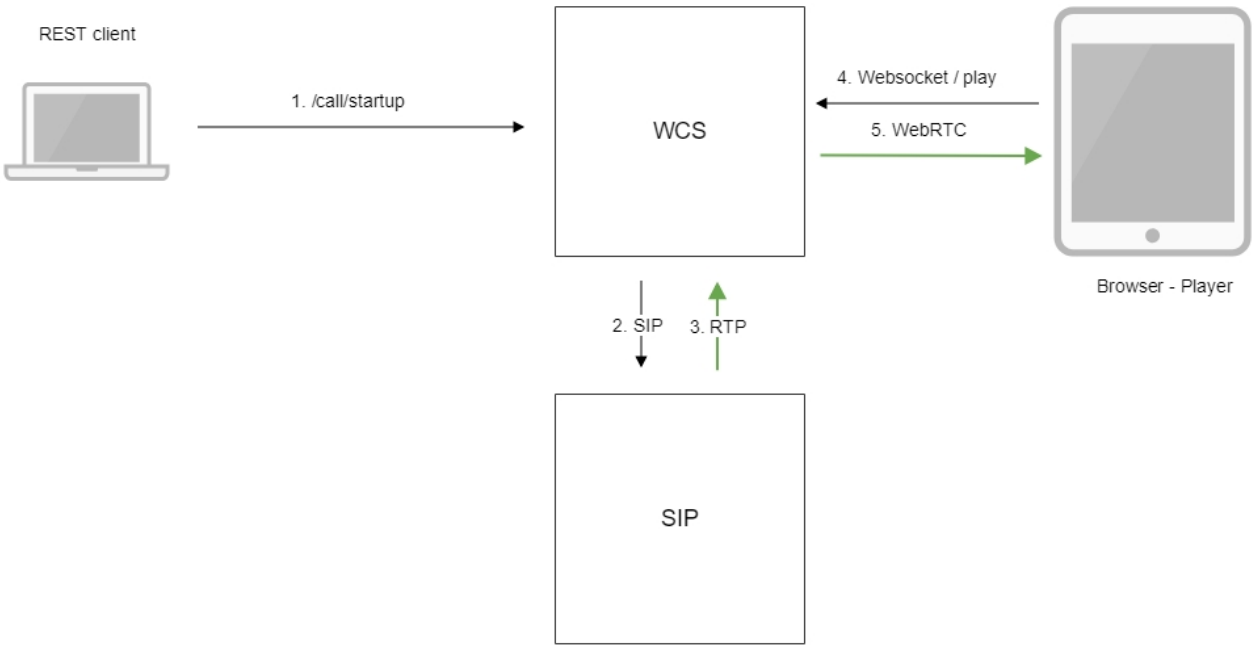
Redirecting a SIP call to a stream (SIP as Stream function)

Overview

A SIP call made through the WCS server can be captured into a stream on the server when the call is created. Then this call can be played in a browser using any method supported by WCS.
The stream captured from a SIP call can be republished to an RTMP server using the REST query /push/startup, just like any media stream on the WCS server.

Operation flowchart



1. The browser starts a call using the /call/startup REST query
2. WCS connects to the SIP server
3. The SIP server sends the RTP stream of the call to WCS
4. The second browser requests playback of the call stream
5. The second browser receives the WebRTC stream

Quick manual on testing

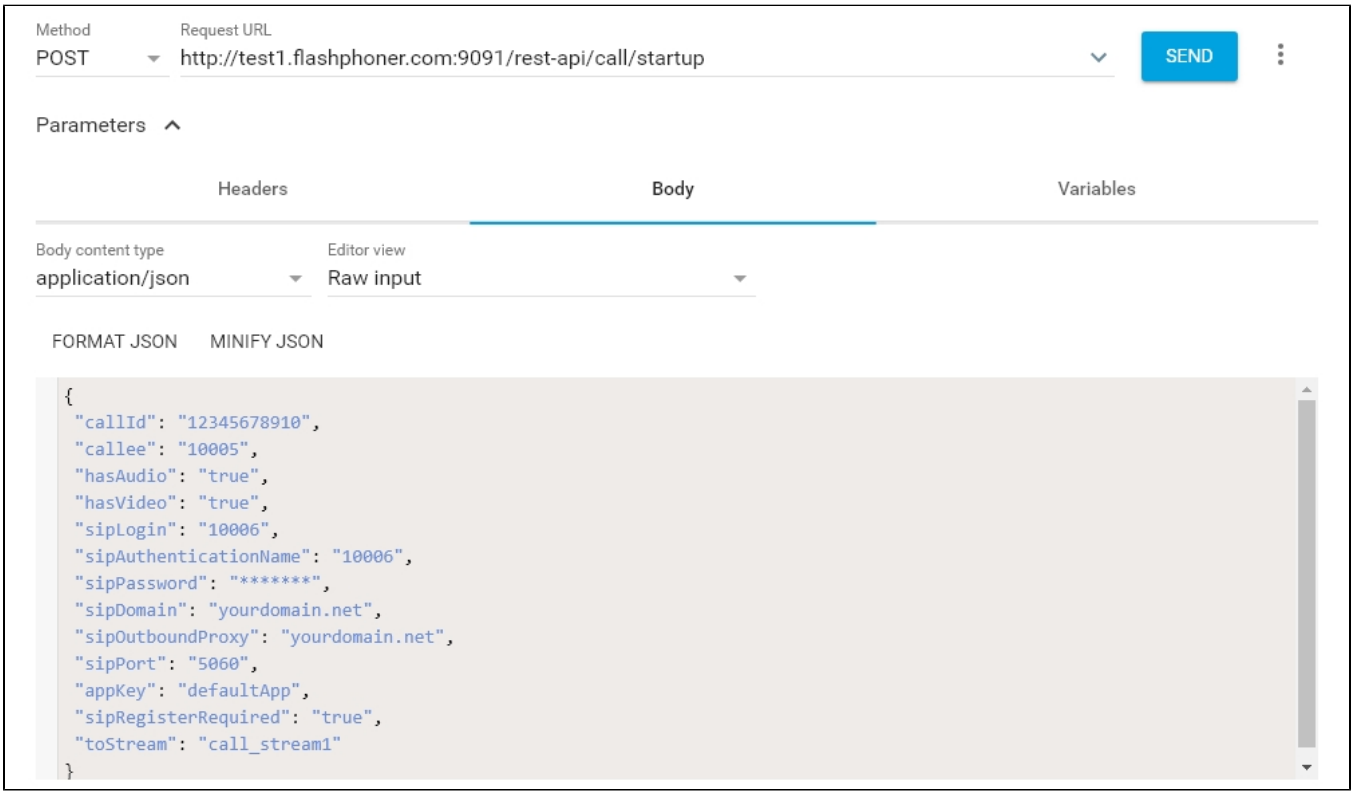
1. For this test we use:

- two SIP accounts;
- the softphone to answer the call;
- theREST-clientin the Chrome browser;
- thePlayerweb application to play the stream.

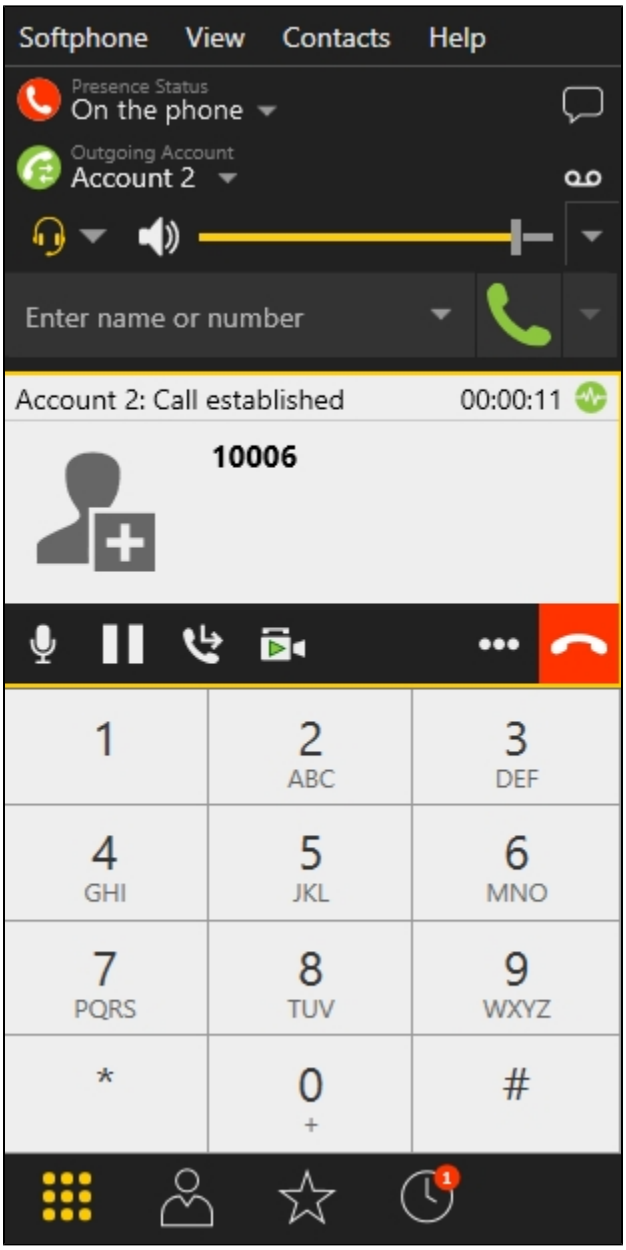
2. Open the REST client. Send the /call/startup query to the WCS server and specify the following as query parameters:

- parameters of your SIP account the call will be made from;
- the stream name to republish the call to (the toStream parameter), for example, call_stream1;

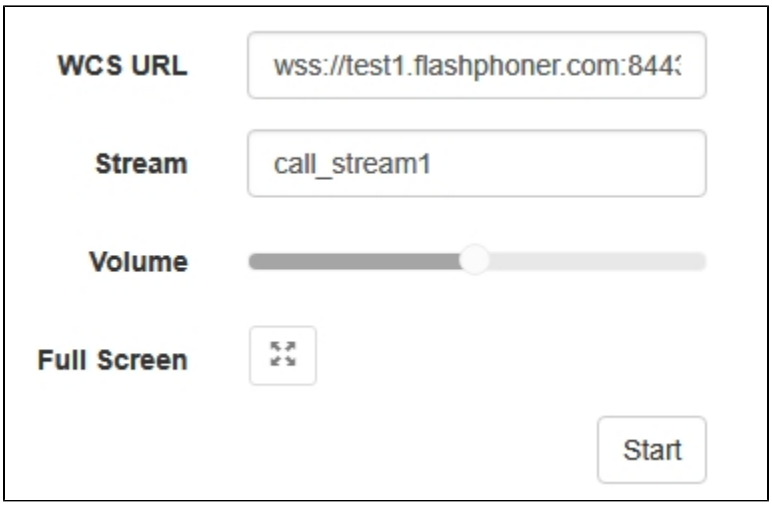
- the name of your second SIP account the call will be made to.



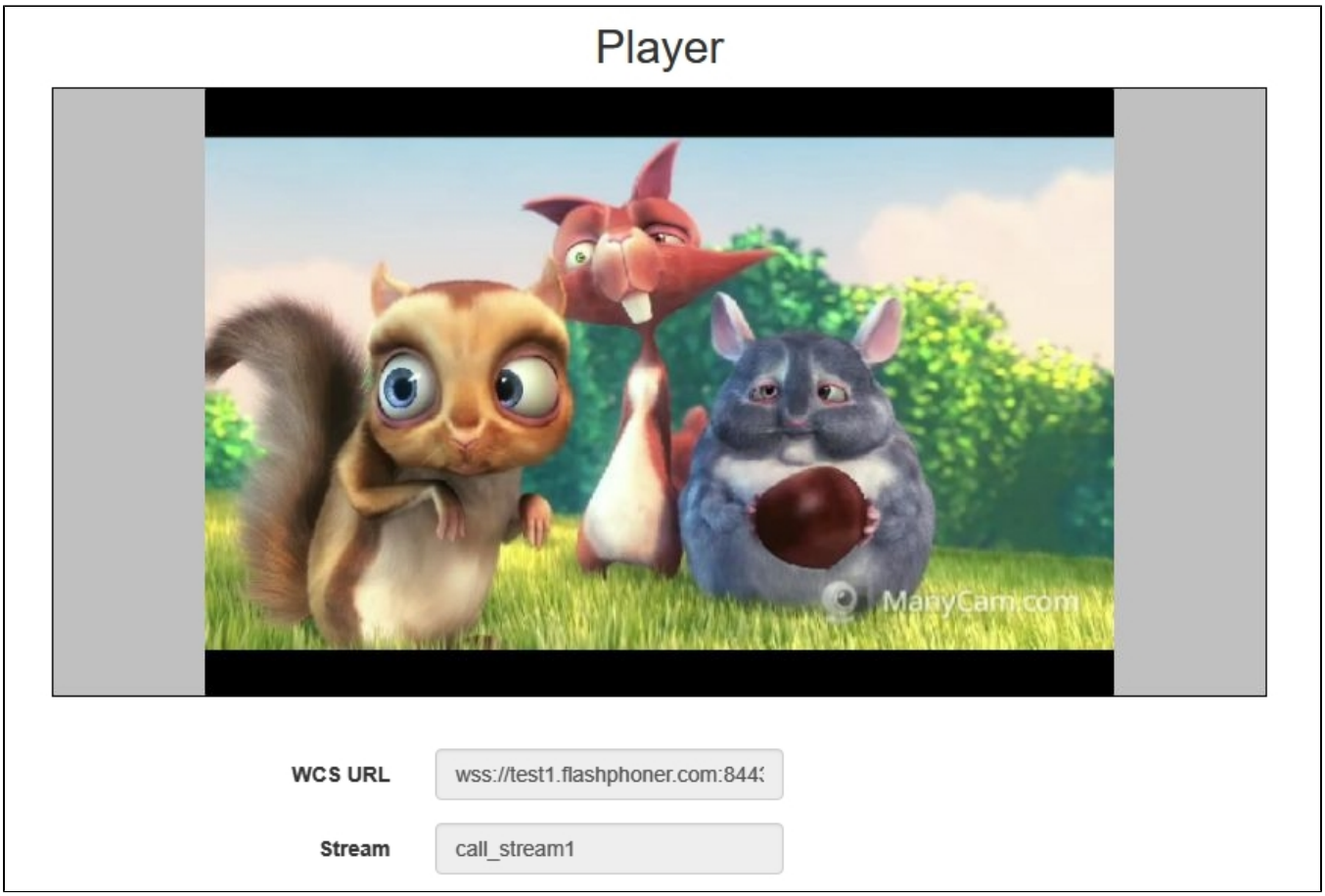
3. Receive and answer the incoming call on the softphone:



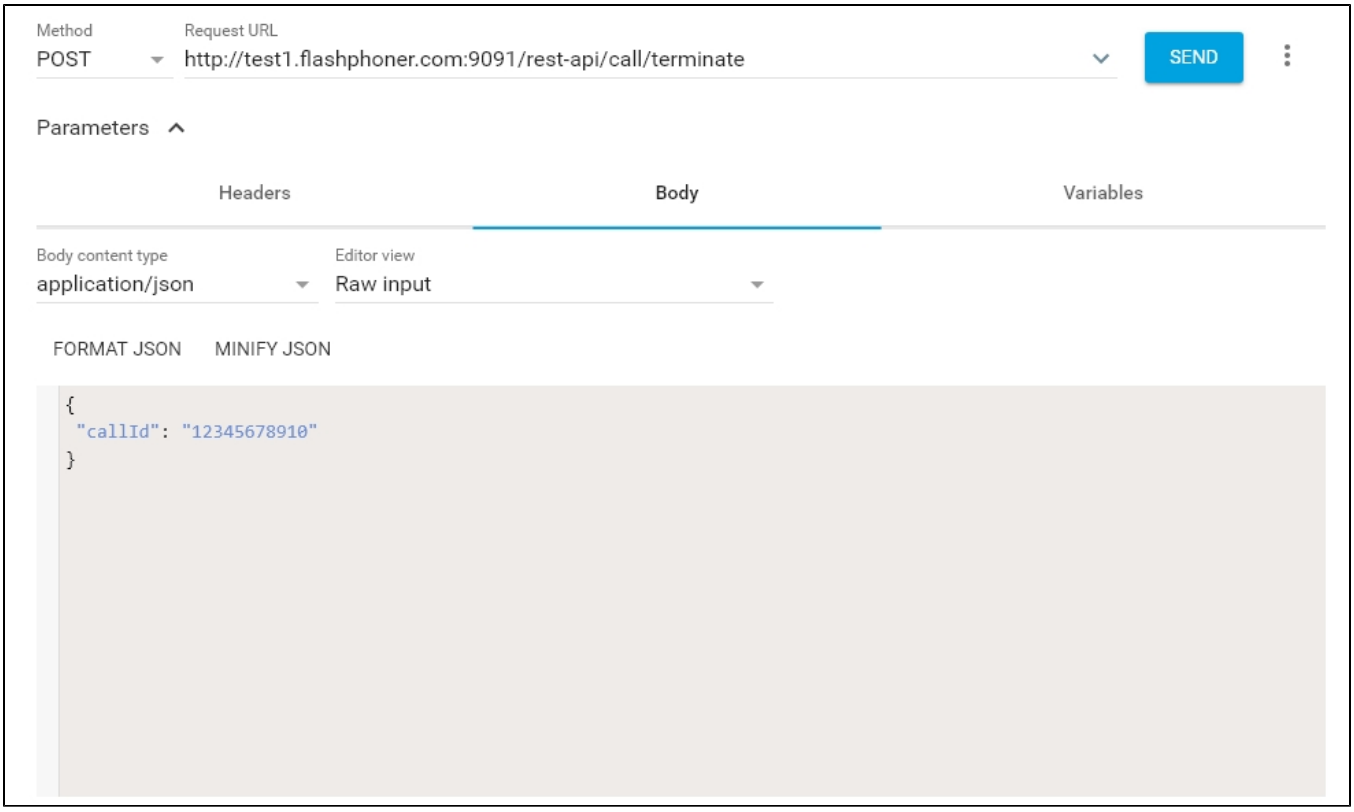
4. Open the Player web application and in the "Stream" field specify the name of the stream the call is redirected to (in our example: call_stream1):



5. Click "Play". The stream starts playing:



6. To terminate the call, send /call/terminate from the REST client to the WCS server and pass the call id in the parameters:



Call flow

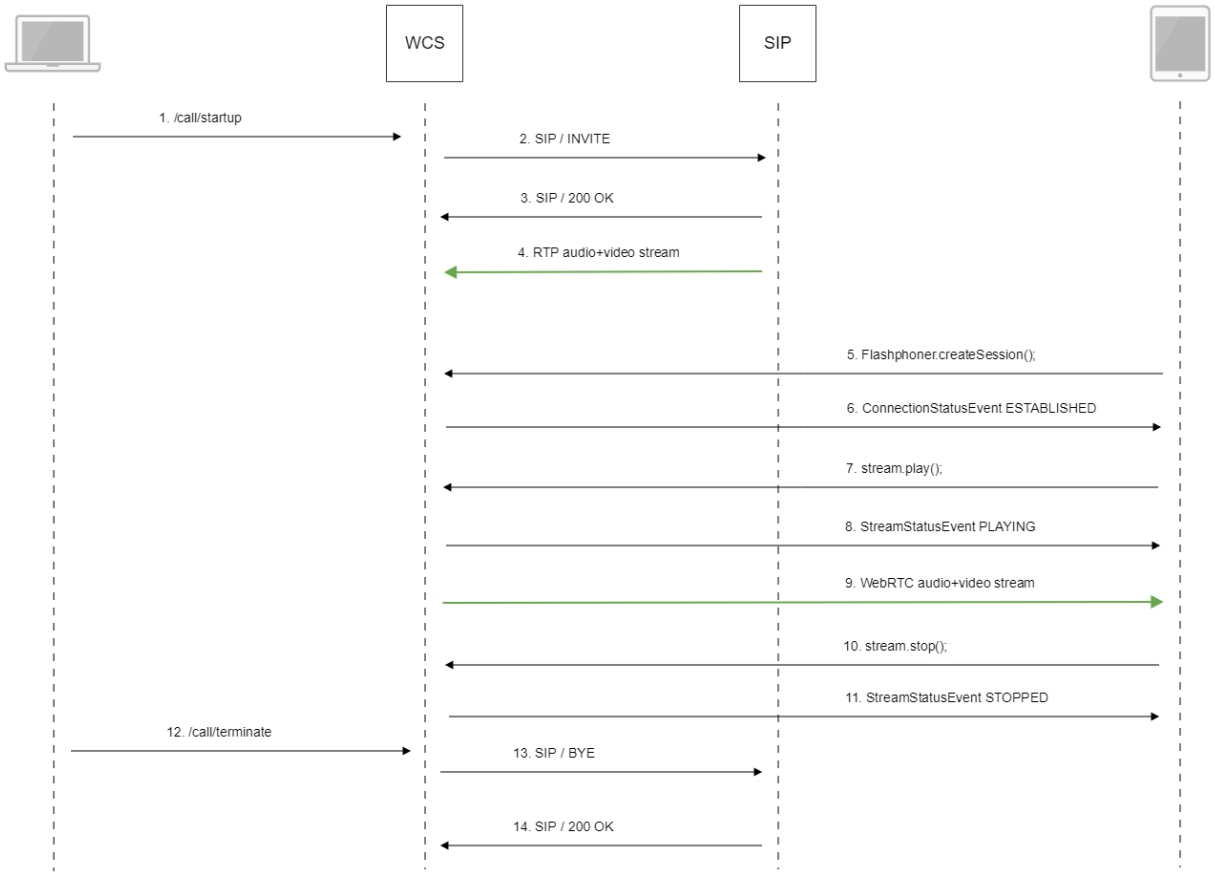
Below is the call flow when using the SIP as RTMP example to create the call and the Player example to play it

sip-as-rtmp-4.html

sip-as-rtmp-4.js

player.html

player.js



1. Sending the REST query /call/startup:

sendREST()code

```
function startCall() {
    ...
    var url = field("restUrl") + "/call/startup";
    callId = generateCallID();
    ...
    var RESTCall = {};
    RESTCall.toStream = field("rtmpStream");
    RESTCall.hasAudio = field("hasAudio");
    RESTCall.hasVideo = field("hasVideo");
    RESTCall.callId = callId;
    RESTCall.sipLogin = field("sipLogin");
    RESTCall.sipAuthenticationName = field("sipAuthenticationName");
    RESTCall.sipPassword = field("sipPassword");
    RESTCall.sipPort = field("sipPort");
    RESTCall.sipDomain = field("sipDomain");
    RESTCall.sipOutboundProxy = field("sipOutboundProxy");
    RESTCall.appKey = field("appKey");
    RESTCall.sipRegisterRequired = field("sipRegisterRequired");

    for (var key in RESTCall) {
        setCookie(key, RESTCall[key]);
    }

    RESTCall.callee = field("callee");

    var data = JSON.stringify(RESTCall);

    sendREST(url, data);
    startCheckCallStatus();

}
```

2. Establishing a connection to the SIP server

3. Receiving a confirmation from the SIP server

4. The RTP stream of the call is sent to the WCS server

5. The Browsers establishes connection to the 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();
    });
```

6. Receiving from the server an event confirming successful connection.

ConnectionStatusEvent ESTABLISHEDcode

```
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(){
    ...
});
```

7. Request to play the stream.

stream.play();code

```
stream = session.createStream(options).on(STREAM_STATUS.PENDING, function(stream) {
    var video = document.getElementById(stream.id());
    if (!video.hasListeners) {
        video.hasListeners = true;
        video.addEventListener('playing', function () {
            $("#preloader").hide();
        });
        video.addEventListener('resize', function (event) {
            var streamResolution = stream.videoResolution();
            if (Object.keys(streamResolution).length === 0) {
                resizeVideo(event.target);
            } else {
                // Change aspect ratio to prevent video stretching
                var ratio = streamResolution.width / streamResolution.height;
                var newHeight = Math.floor(options.playWidth / ratio);
                resizeVideo(event.target, options.playWidth, newHeight);
            }
        });
    }
    ...
});
stream.play();
```

8. Receiving from the server an event confirming successful playing of the stream.

StreamStatusEvent, status PLAYINGcode

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

9. Sending audio- and video stream via WebRTC

10. Stopping playing the stream.

stream.stop();code

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

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

StreamStatusEvent, status STOPPEDcode

```
    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();
```

12. Sending the /call/terminate REST query:

sendREST()code

```
function hangup() {
    var url = field("restUrl") + "/call/terminate";
    var currentCallId = { callId: callId };
    var data = JSON.stringify(currentCallId);
    sendREST(url, data);
}
```

13. Sending the command to the SIP server

14. Receiving confirmation from the SIP server

SIP as stream recording

All streams captured from SIP calls can be recorded on server. To do this, set the following parameters inflashphoner.propertiesfile:

```
sip_single_route_only=true
sip_record_stream=true
```

The following codecs are supported:

- Video: H264
- Audio: opus, PCMA (alaw), PCMU (ulaw)

Stream recording is described here in details.

Known issues

1. Stream captured from SIP call, can not be played, if RTP session is not initialized for this stream.

Symptoms: SIP stream is published on server, but can not be played.

Solution: enable RTP session initializing with the following parameter

```
rtp_session_init_always=true
```