SIP calls in a WebRTC-compatible browser

Overview

Web Call Server supports audio and video calls from a browser to SIP devices, PBX servers, SIP-GSM gates, VoIP conferences and other devices supporting the SIP protocol. Therefore, a web application can work in a browser as a software phone with the support for the SIP protocol, receive and initiate voice and video calls.

Supported platforms and browsers

	Chrome	Firefox	Safari	Edge
Windows	✅	✅	❌	✅
Mac OS	✅	✅	✅	✅
Android	✅	✅	❌	✅
iOS	✅	✅	✅	✅

Supported protocols

- WebRTC
- RTP
- SIP

Supported codecs

- H.264
- VP8
- G.711
- Speex
- G.729
- Opus

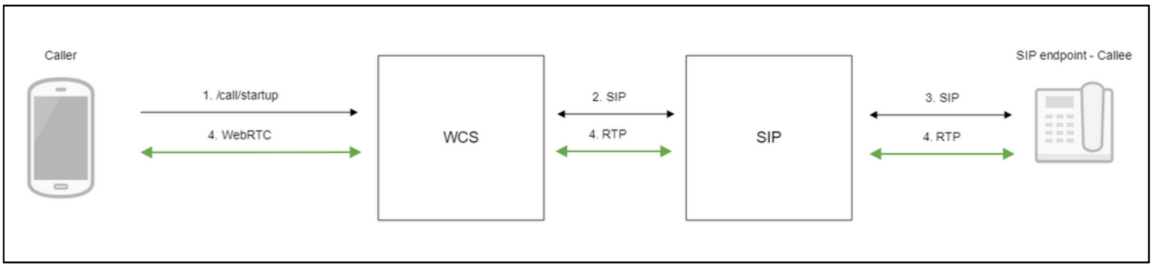
Supported SIP functions

- DTMF
- Holding a call
- Transferring a call

SIP functions are managed using the WebSDK.

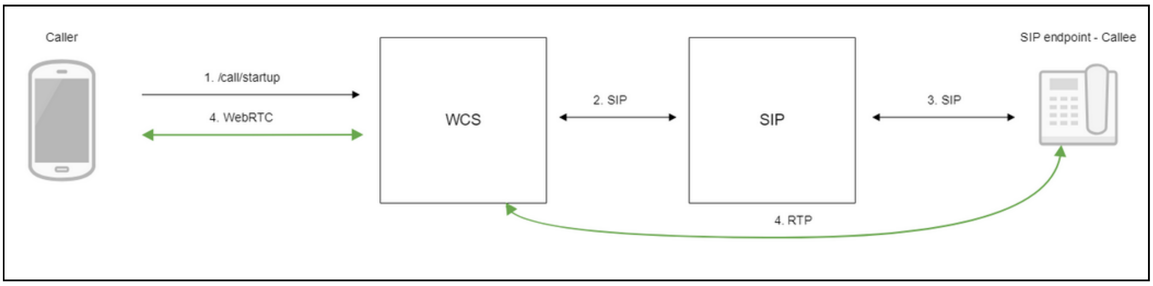
Operation flowchart

1. SIP server as a proxy server to transfer calls and RTP media



1. The browsers initiates a call using WebSDK.
2. WCS connects to the SIP server.
3. The SIP server connects to the SIP device receiving the call.
4. The browser and the SIP device exchange audio and video streams.

2. SIP server as a server to transfer calls only



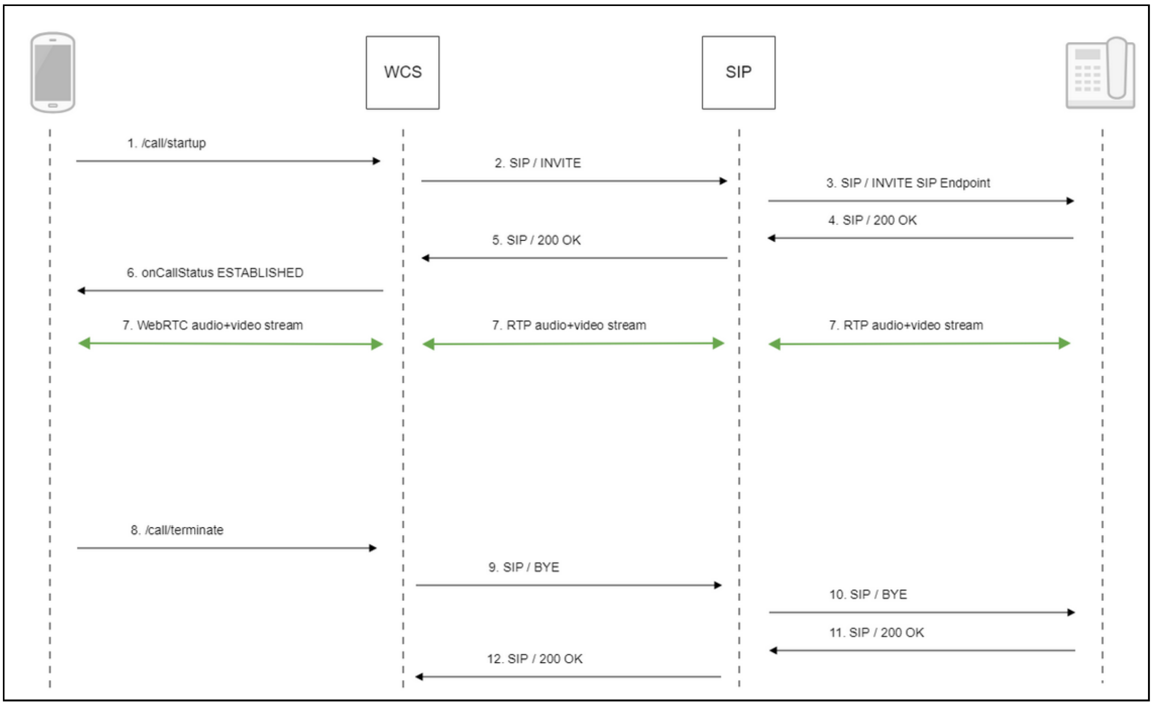
1. The browsers initiates a call using WebSDK.
2. WCS connects to the SIP server.
3. The SIP server connects to the SIP device receiving the call.
4. The browser and the SIP device exchange audio and video streams.

Call flow

Below is the call flow when using the Phone example to create a call.

phone.html

phone.js



1. Creating a call using WebSDK
`Session.createCall()`, `Call.call()` code

```
var outCall = session.createCall({
    callee: $("#callee").val(),
    visibleName: $("#sipLogin").val(),
    localVideoDisplay: localDisplay,
    remoteVideoDisplay: remoteDisplay,
    constraints: constraints,
    receiveAudio: true,
    receiveVideo: false
    ...
});

outCall.call();
```

2. Sending `SIP INVITE` to the SIP server

3. Sending `SIP INVITE` to the SIP device

4. Receiving a confirmation from the SIP device

5. Receiving a confirmation from the SIP server

6. Receiving from the server an event confirming successful connection
`CALL_STATUS.ESTABLISHED` code

```
var outCall = session.createCall({
    ...
}).on(CALL_STATUS.RING, function(){
    ...
```

```
}).on(CALL_STATUS.ESTABLISHED, function(){
    setStatus("#callStatus", CALL_STATUS.ESTABLISHED);
    $("#holdBtn").prop('disabled',false);
    onAnswerOutgoing();
}).on(CALL_STATUS.HOLD, function() {
    ...
}).on(CALL_STATUS.FINISH, function(){
    ...
}).on(CALL_STATUS.FAILED, function(){
    ...
});
```

7. The caller and the callee exchange audio and video streams
8. Terminating the call
   `Call.hangup()` code

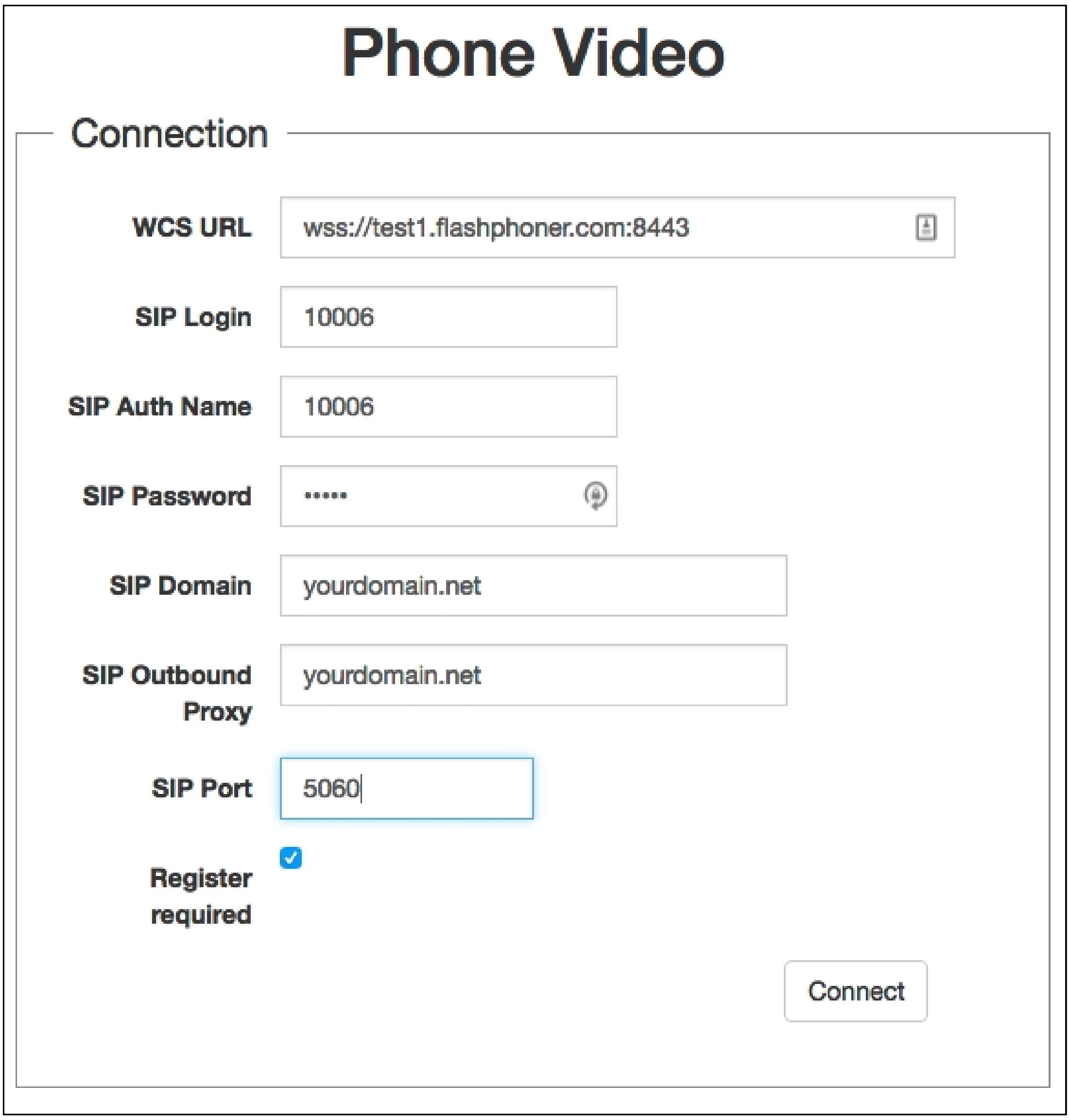
```
function onConnected(session) {
    $("#connectBtn,
#connectTokenBtn").text("Disconnect").off('click').click(function(){
        $(this).prop('disabled', true);
        if (currentCall) {
            showOutgoing();
            disableOutgoing(true);
            setStatus("#callStatus", "");
            currentCall.hangup();
        }
        session.disconnect();
    }).prop('disabled', false);
}
```

9. Sending `SIP BYE` to the SIP server
10. Sending `SIP BYE` to the SIP device
11. Receiving a confirmation from the SIP device
12. Receiving a confirmation from the SIP server

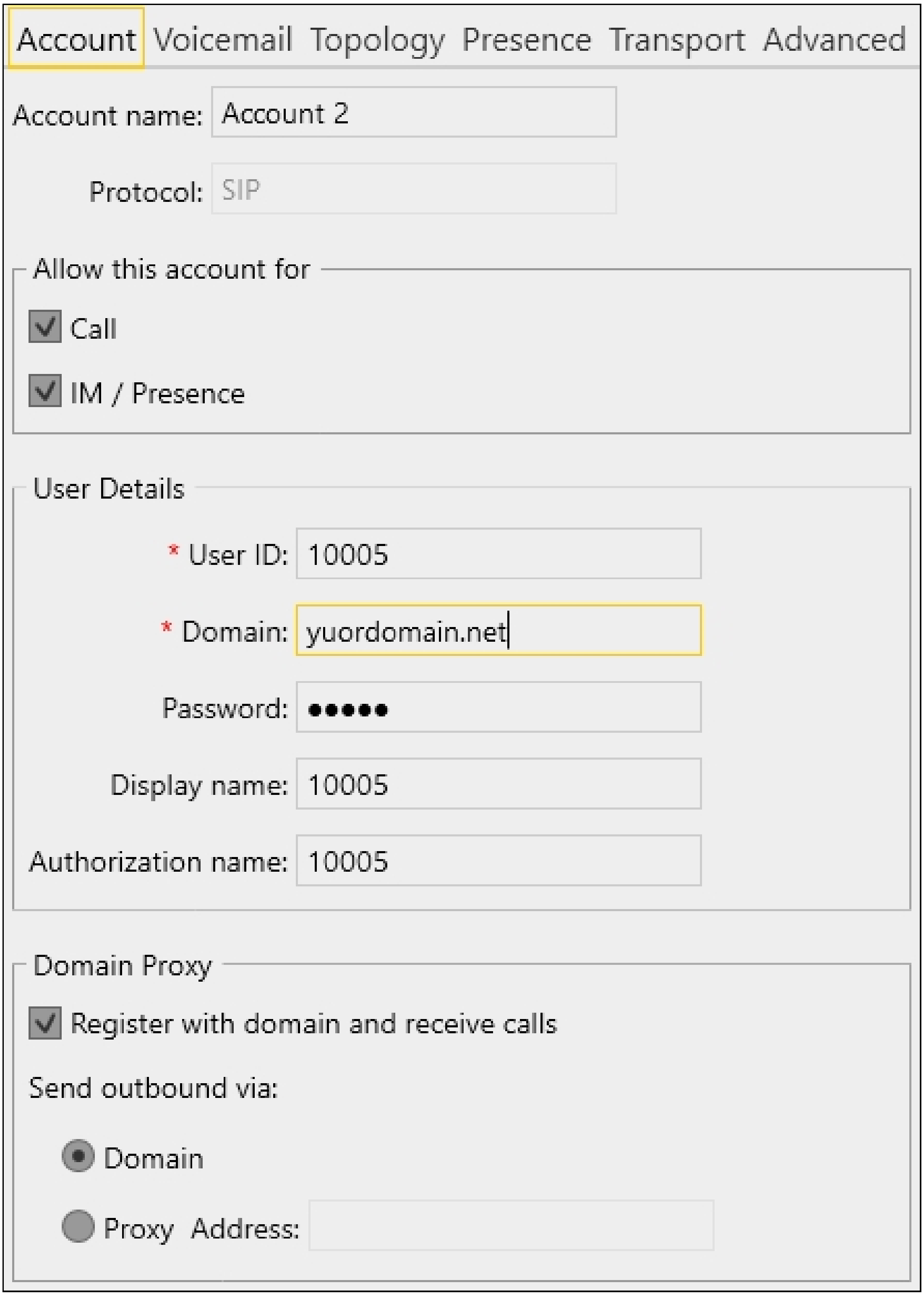
Testing

Making an outgoing call from a browser to a SIP device

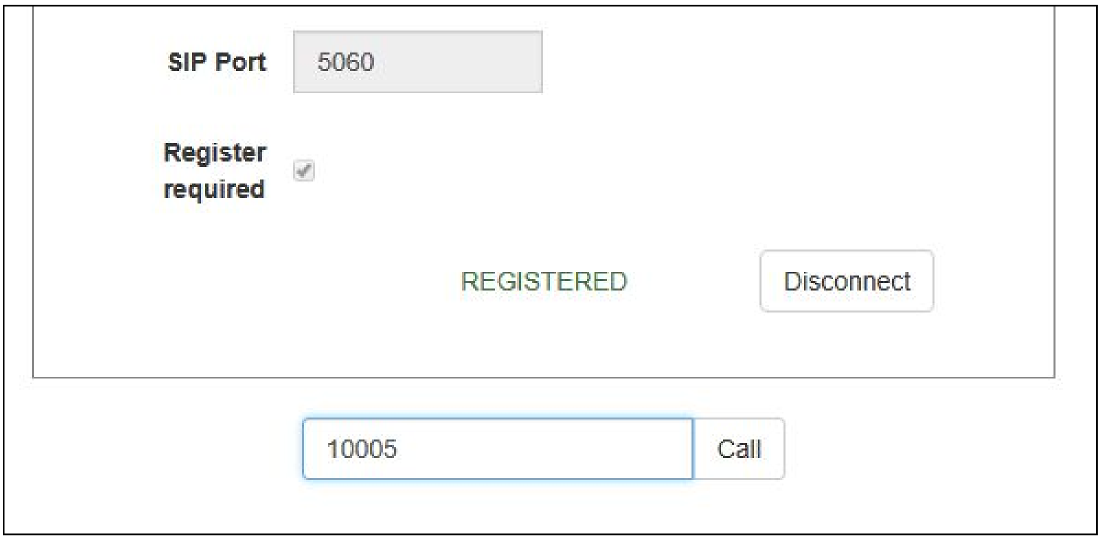
1. For the test we use:
   - two SIP accounts;
   - the Phone Video web application to make a call;
   - a software phone to answer the call.
2. Open the Phone Video web application. Enter the data of the SIP account making the call from a browser:


Phone Video
Connection
WCS URL
wss://test1.flashphoner.com:8443
SIP Login
10006
SIP Auth Name
10006
SIP Password
•••••
SIP Domain
yourdomain.net
SIP Outbound Proxy
yourdomain.net
SIP Port
5060
Register required
Connect

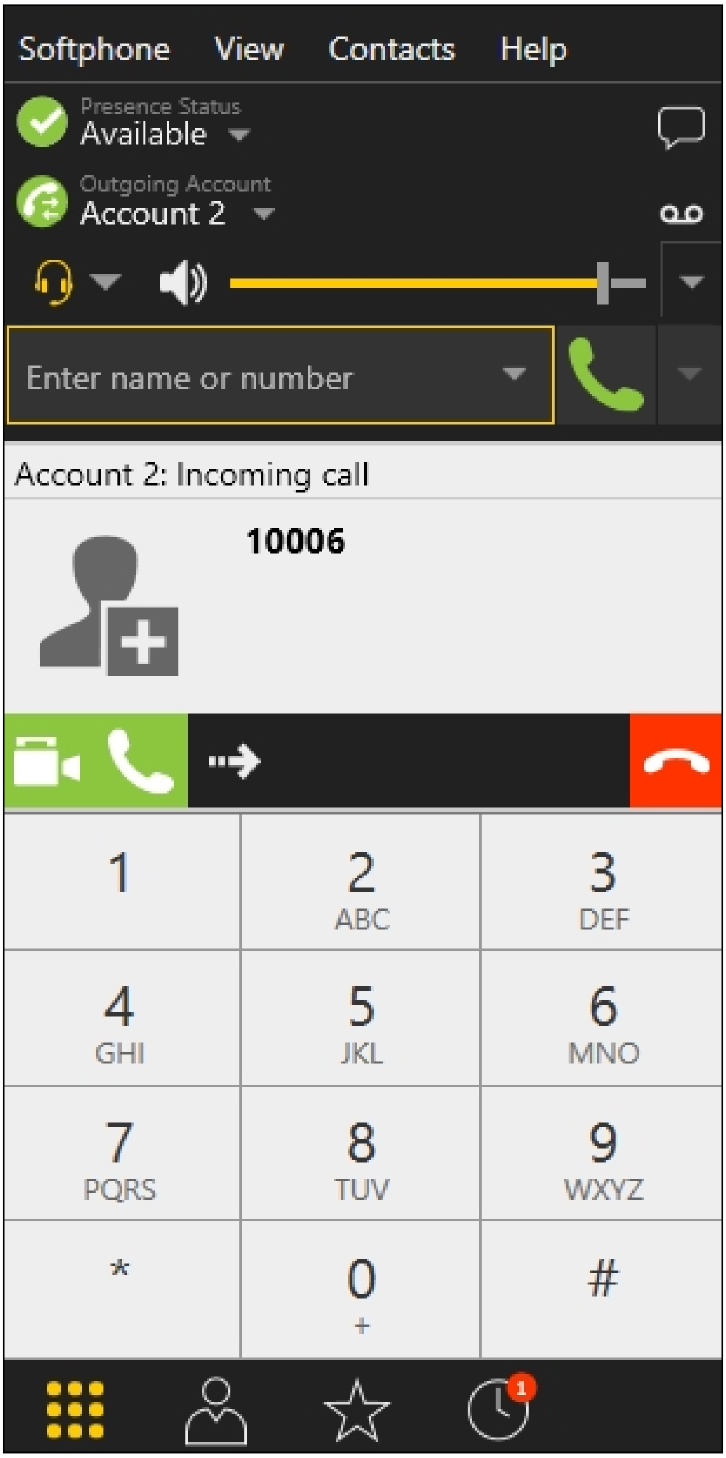
3. Run the software phone, enter the data of the SIP account receiving the call:

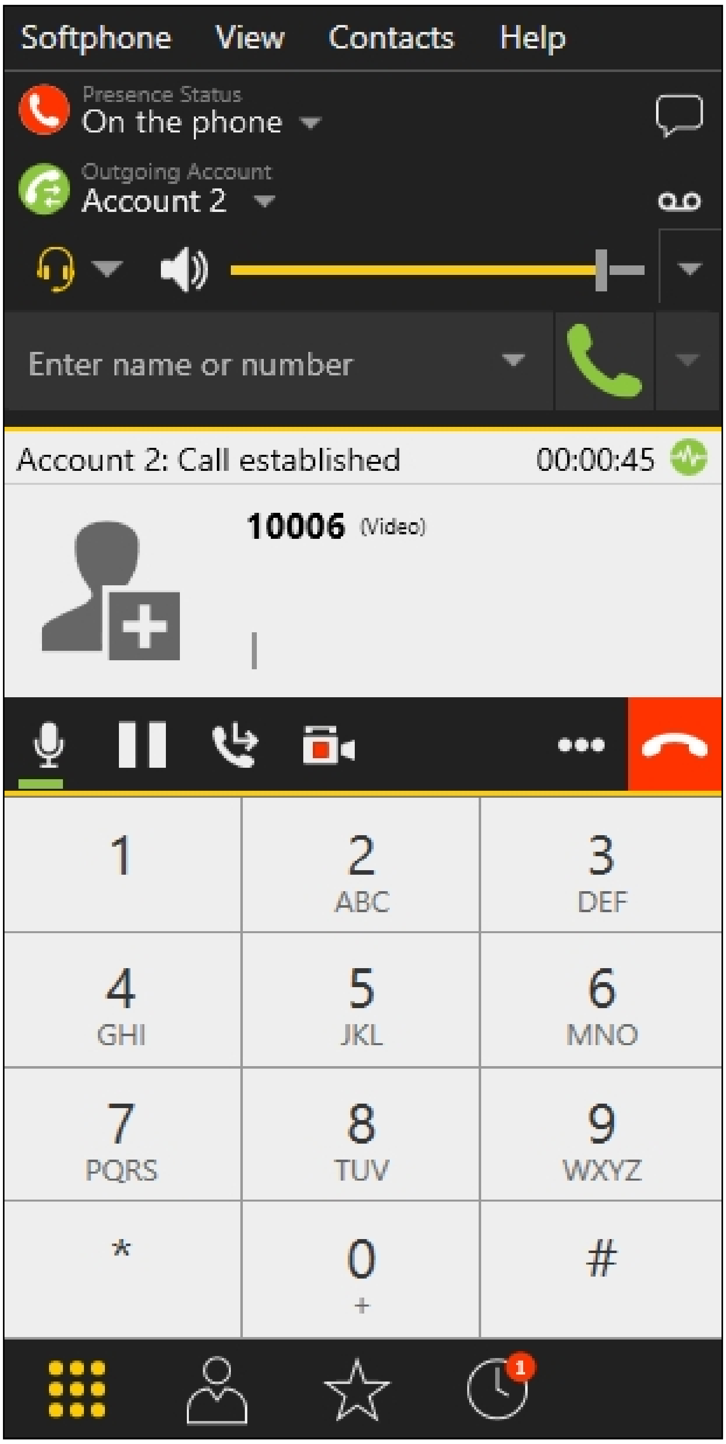


4. Click the `Connect` button in the browser. Then enter the identifier of the SIP account that receives the call and click the `Call` button:

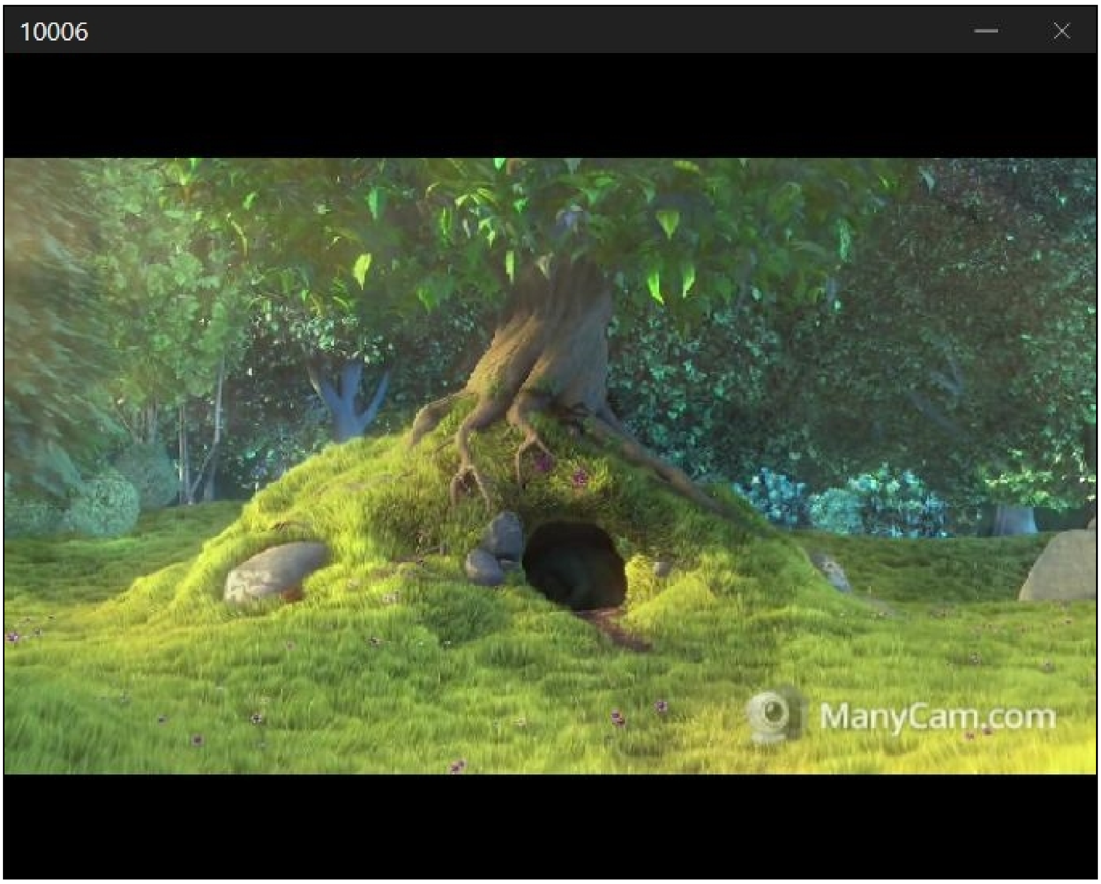

SIP Port
5060
Register required
REGISTERED
Disconnect
10005
Call

5. Answer the call in the softphone by clicking the answer a video call button:





In a separate window, the video broadcast from the browser is shown:

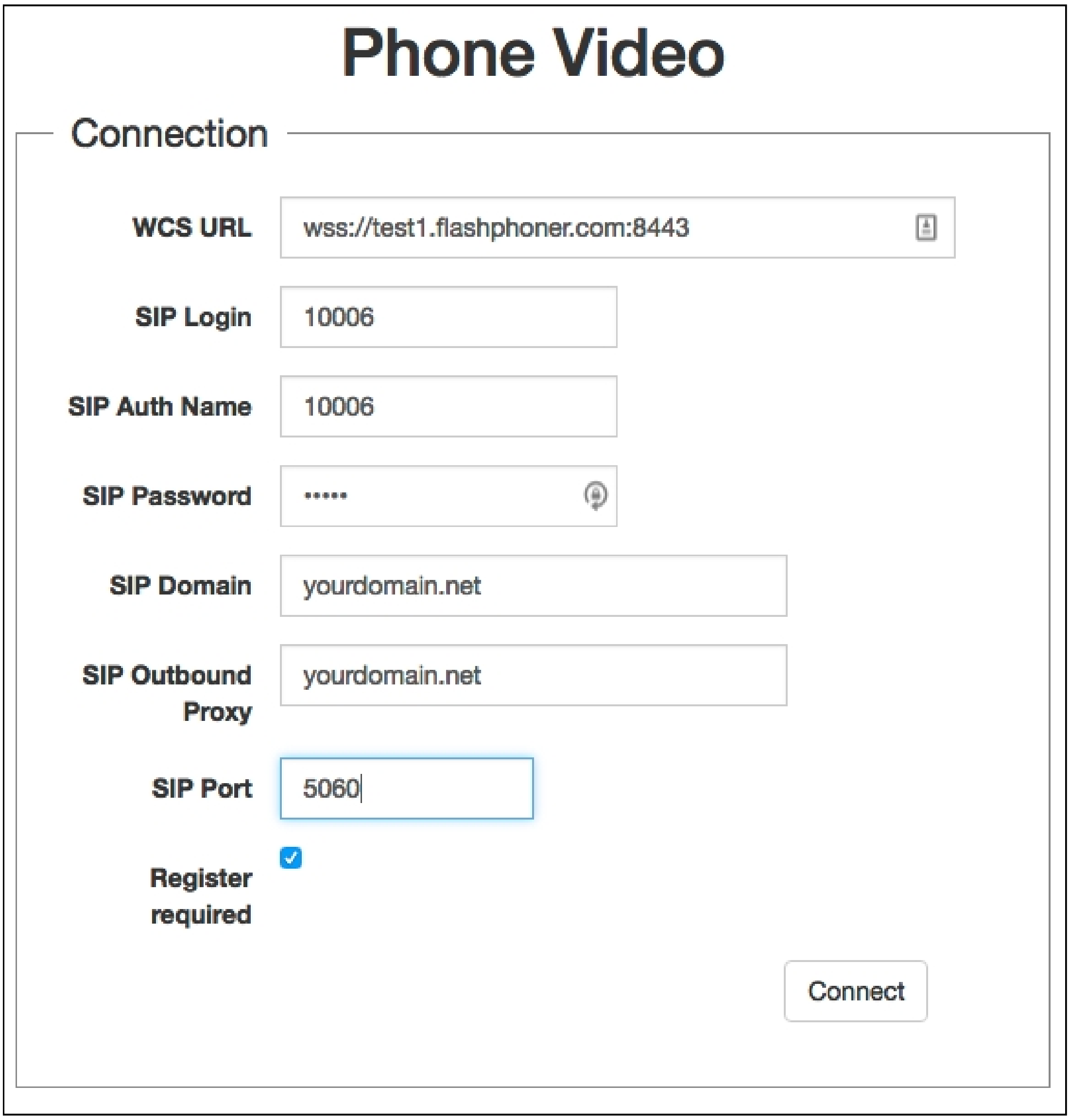


6. The browser also displays the video:

7. To terminate the call, click the `Hangup` button in the browser or in the softphone.

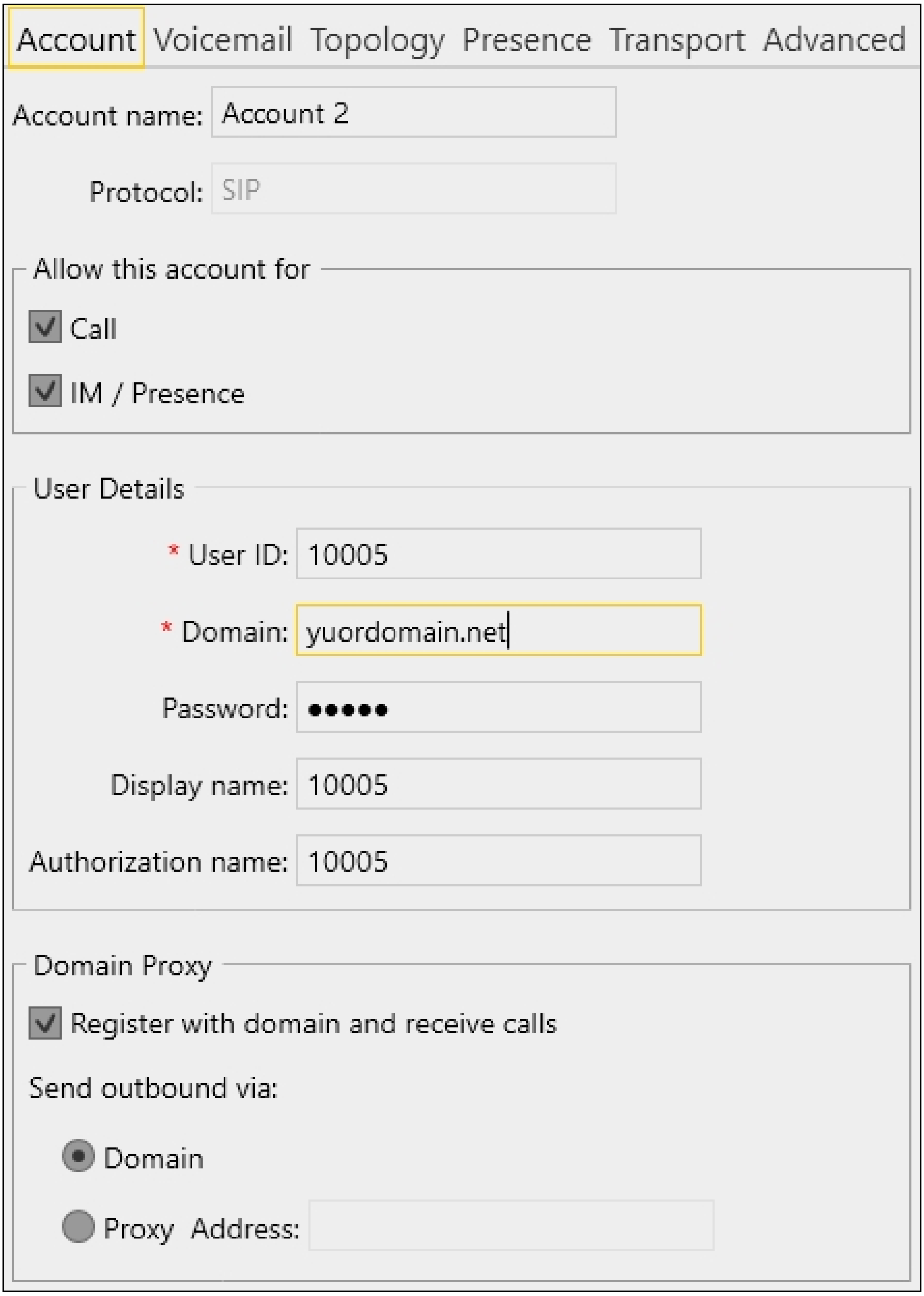
Receiving an incoming call from a SIP device in a browser

1. For the test we use:
   - two SIP accounts;
   - a software phone to make a call;
   - the Phone Video web application to answer the call.
2. Open the Phone Video web application. Enter the data of the SIP account receiving the call in a browser:

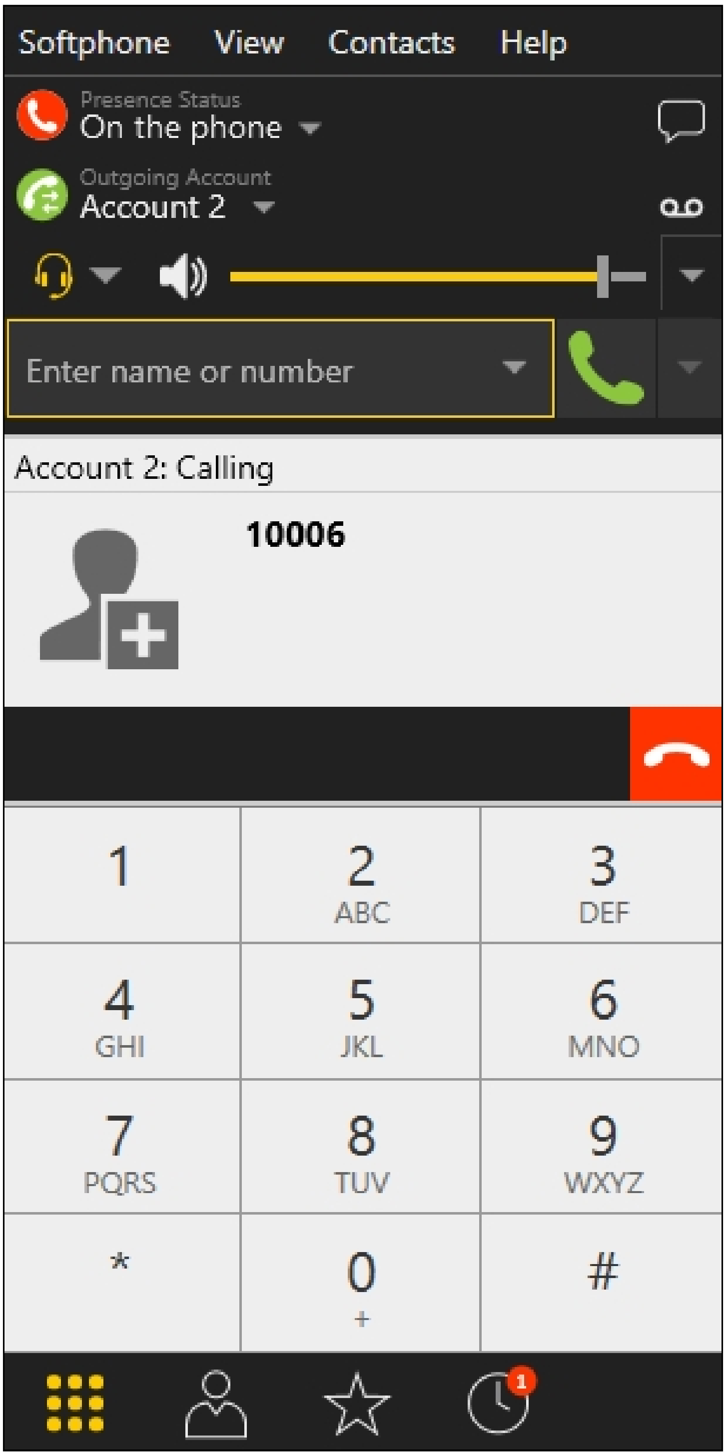


Click the `Connect` button in the browser to establish a connection to the WCS server

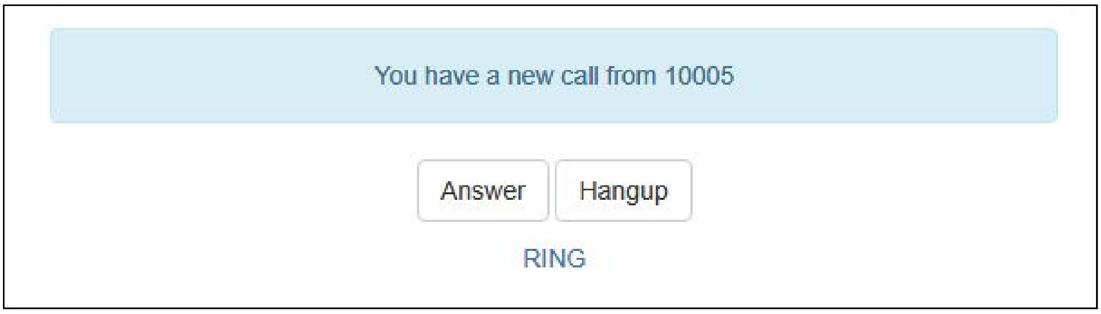
3. Run the software phone, enter the data of the SIP account making the call:



4. In the softphone enter the identifier of the SIP account that receives the call and click the `Call` button:

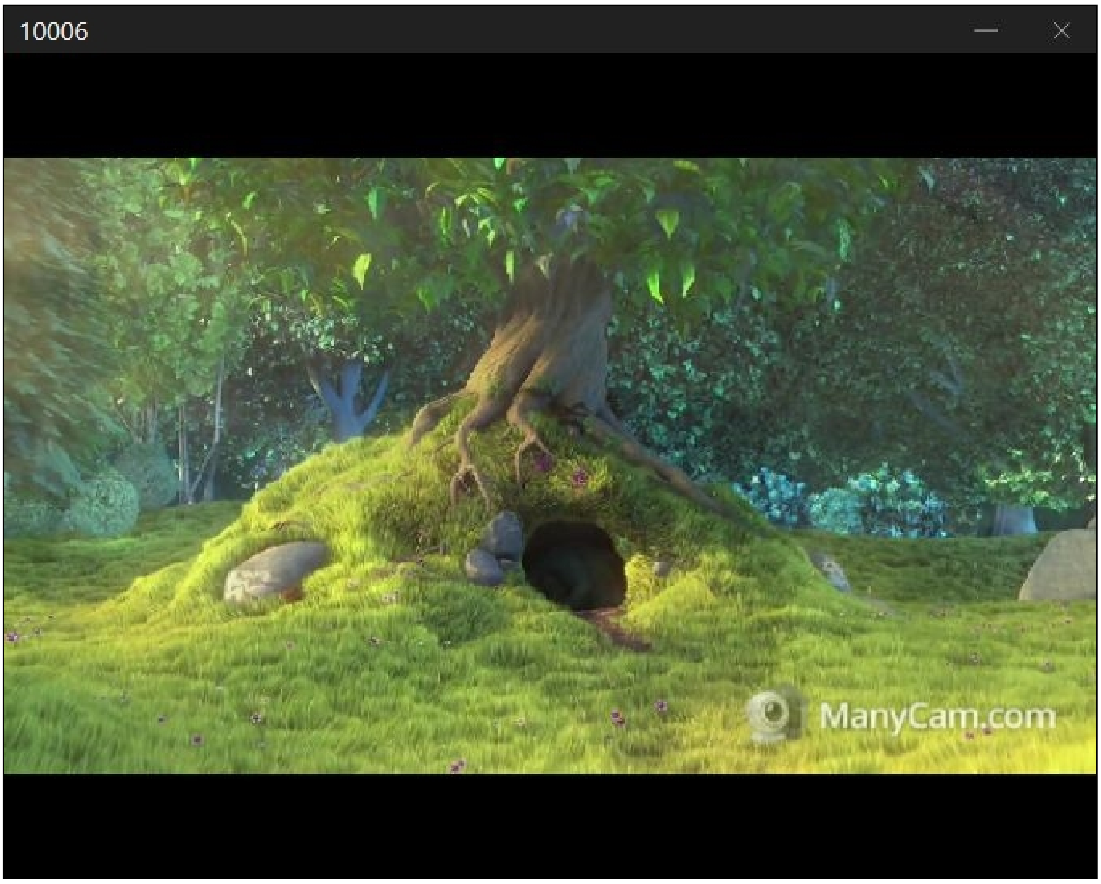


5. Answer the call in the browser by clicking the `Answer` button:



6. The browser displays the video:

7. The video broadcast from a browser also is displaing in a separate window of the softphone:

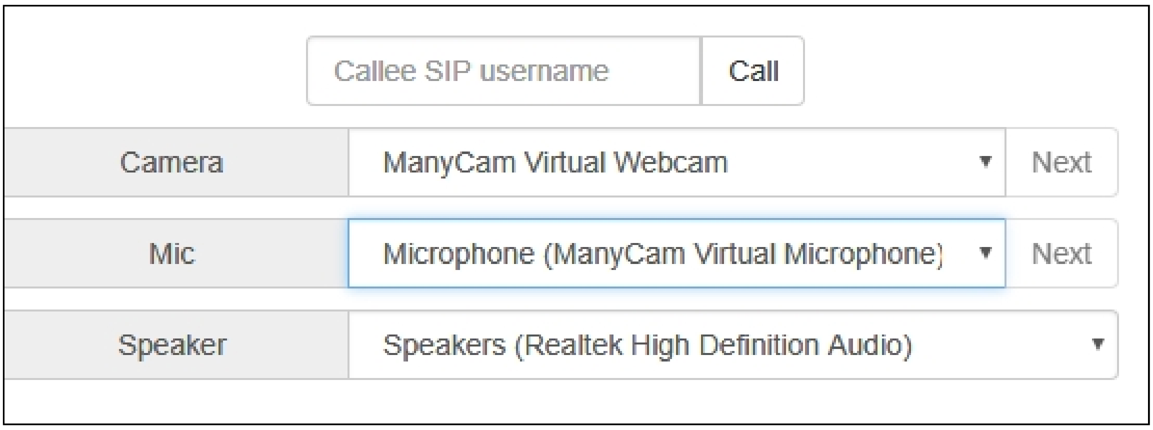


8. To terminate the call, click the `Hangup` button in the browser or the end call button in the softphone.

Camera, microphone and sound output devices management

Selection and switching input and output devices

Like a video stream capture, a camera, microphone and (in Chrome browser only) a sound output device can be selected while making a SIP call from browser. Besides, devices can be switched during a call.



1. Choosing camera, microphone and sound output device code

```
Flashphoner.getMediaDevices(null, true,
MEDIA_DEVICE_KIND.ALL).then(function (list) {
    for (var type in list) {
        if (list.hasOwnProperty(type)) {
            list[type].forEach(function(device) {
                if (device.type == "mic") {
                    ...
                } else if (device.type == "speaker") {
                    ...
                } else if (device.type == "camera") {
                    ...
                }
            });
        }

    }
    ...
}).catch(function (error) {

    $("#notifyFlash").text("Failed to get media devices "+error);
});
```

2. Switching sound output device during a call code

```
$( "#speakerList" ).change(function() {
    if (currentCall) {
        currentCall.setAudioOutputId($(this).val());
    }
});
```

3. Swithching microphone during a call code

```
$("#switchMicBtn").click(function() {
    if (currentCall) {
        currentCall.switchMic().then(function(id) {
            $('#micList option:selected').prop('selected', false);
            $("#micList option[value='"+ id +"']").prop('selected', true);
        }).catch(function(e) {
            console.log("Error " + e);
        });
    }
}).prop('disabled', true);
```

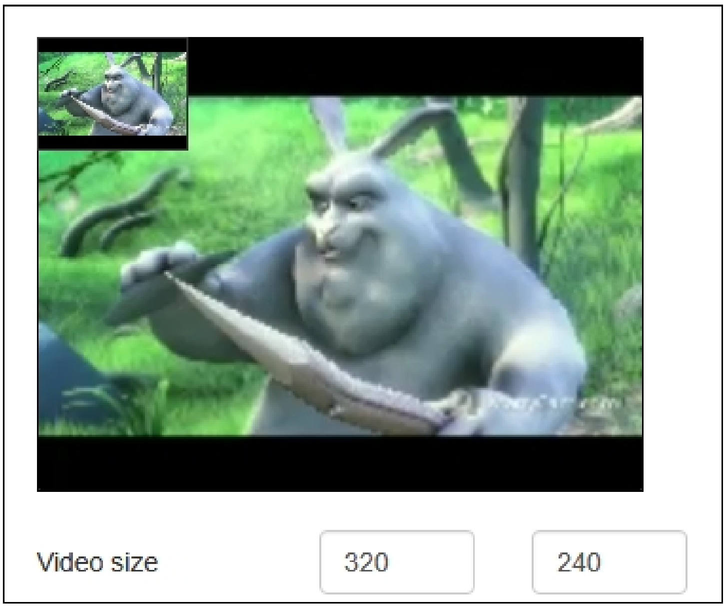
4. Switching camera during a call code

```
$("#switchCamBtn").click(function() {
   if (currentCall) {
       currentCall.switchCam().then(function(id) {
           $('#cameraList option:selected').prop('selected', false);
           $("#cameraList option[value='"+ id +"']").prop('selected',
true);
       }).catch(function(e) {
```

```
            console.log("Error " + e);
        });
    }
}).prop('disabled', true);
```

Video size setting

An outgoing video size can be specified while making a call



code:

```
function getConstraints() {
    var constraints = {
        ...
        video: {
            deviceId: {exact: $('#cameraList').find(":selected").val()},
            width: parseInt($('#sendWidth').val()),
            height: parseInt($('#sendHeight').val())
        }
    };
    if (Browser.isSafariWebRTC() && Browser.isiOS() &&
Flashphoner.getMediaProviders()[0] === "WebRTC") {
        constraints.video.width = {min: parseInt($('#sendWidth').val()), max:
640};
        constraints.video.height = {min: parseInt($('#sendHeight').val()),
max: 480};
    }
    return constraints;
}
```

Making a call without microphone and camera

In some cases, when a call supposes no two-way communication, e.g. when calling to voice menu, it is possible to make a call without using microphone and camera.

To do this RTP activity timer should be disabled with following parameter in flashphoner.properties file

```
rtp_activity_detecting=false
```

and audio and video should be turned off in outgoing call constraints for Chrome, Safari and MS Edge browsers

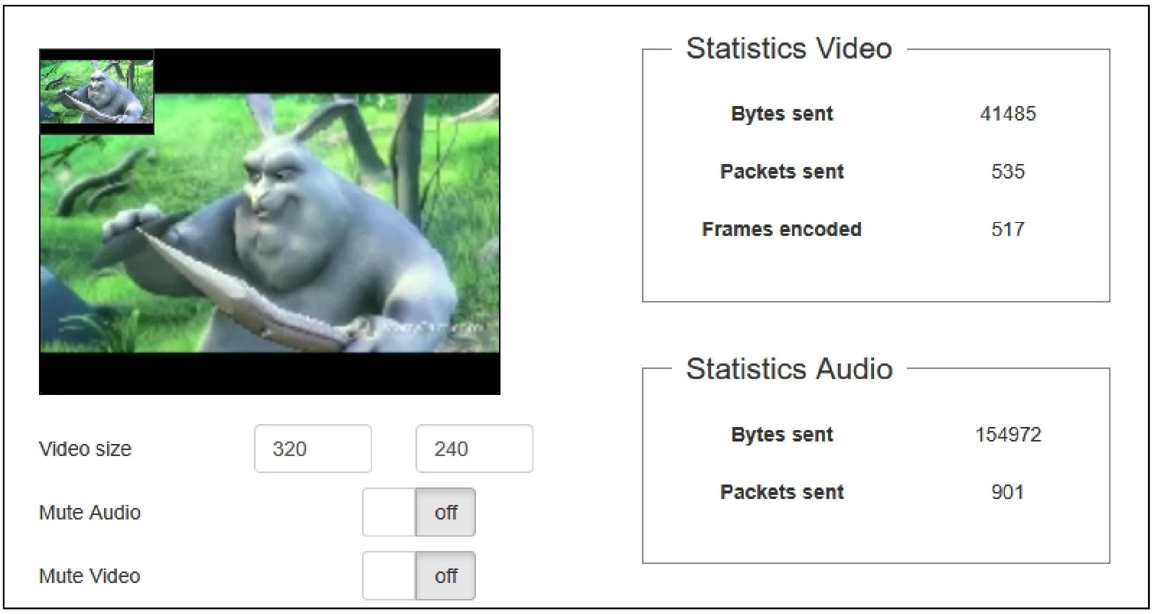
```
var constraints = {
    audio: false,
    video: false
};

var outCall = session.createCall({
    callee: $("#callee").val(),
    visibleName: $("#sipLogin").val(),
    constraints: constraints,
    ...
})
```

In addition to it, an empty audio stream should be created for Firefox browser:

```
var constraints = {
    audio: false,
    video: false
};
if(Browser.isFirefox()) {
    var audioContext = new AudioContext();
    var emptyAudioStream =
audioContext.createMediaStreamDestination().stream;
    constraints.customStream = emptyAudioStream;
}
var outCall = session.createCall({
    callee: $("#callee").val(),
    visibleName: $("#sipLogin").val(),
    constraints: constraints,
    ...
})
```

WebRTC statistics displaying

A client application can get WebRTC statistics according to the standard during a SIP call. The statistics can be displayed in browser, for example:



Note that in Safari browser audio only statistics can be displayed.

`Call.getStats()` code

```
currentCall.getStats(function (stats) {
    if (stats && stats.outboundStream) {
        if (stats.outboundStream.videoStats) {

$('#videoStatBytesSent').text(stats.outboundStream.videoStats.bytesSent);

$('#videoStatPacketsSent').text(stats.outboundStream.videoStats.packetsSent);

$('#videoStatFramesEncoded').text(stats.outboundStream.videoStats.framesEncoded)

        } else {
            ...
        }

        if (stats.outboundStream.audioStats) {

$('#audioStatBytesSent').text(stats.outboundStream.audioStats.bytesSent);

$('#audioStatPacketsSent').text(stats.outboundStream.audioStats.packetsSent);
        } else {
            ...
        }
    }
});
```

Supported codecs setting

WCS sets the codecs supported to INVITE SDP according to the following parameters in flashphoner.properties file

1. The codecs specified with `codecs` parameter are included to INVITE SDP, by default

```
codecs=opus,alaw,ulaw,g729,speex16,g722,mpeg4-generic,telephone-event,h264,vp8,flv,mpv
```

2. The codecs specified with `codecs_exclude_sip` parameter are excluded from INVITE SDP, by default

```
codecs_exclude_sip=mpeg4-generic,flv,mpv
```

3. The codecs specified by browser are excluded from INVITE SDP if this parameter is set

```
allow_outside_codecs=false
```

4. The codecs specified with `stripCodecs` parameter in client application are excluded from INVITE SDP, for example

```
var outCall = session.createCall({
    callee: $("#callee").val(),
    ...
    stripCodecs: "SILK,G722"
    ...
});

outCall.call();
```

Additional SDP parameters passing in `SIP INVITE` request and `200 OK` response

When call is made with JavaScript API, an additional parameters can be passed to control bandwith via SDP, for outgoing calls (to `SIP INVITE` request)

```
var sdpAttributes = ["b=AS:3000","b=TIAS:2500000","b=RS:1000","b=RR:3000"];
var outCall = session.createCall({
        sipSDP: sdpAttributes,
        ...
    });
```

and incoming calls (to `200 OK` response)

```
var sdpAttributes = ["b=AS:3000","b=TIAS:2500000","b=RS:1000","b=RR:3000"];
inCall.answer({
   sipSDP: sdpAttributes,
   ...
});
```

Those parameters will be added to SDP after connection information (`c=IN IP4`) and before time description (`t=0 0`):

```
v=0
o=Flashphoner 0 1541068898263 IN IP4 192.168.1.5
s=Flashphoner/1.0
c=IN IP4 192.168.1.5
b=AS:3000
b=TIAS:2500000
b=RS:1000
b=RR:3000
t=0 0
m=audio
```

SIP calls using SIP TLS signaling

SIP TLS signaling may be enabled with the following parameter

```
sip_use_tls=true
```

In this case, SIP PBX cetrificate will be checked using local system certificates storage. Therefore, a valid SSL certificate from well known CA should be installed on SIP PBX server to use SIP TLS.

SIP calls via SIP PBX server with self-signed SSL certificate

To make a SIP call via SIP PBX server with self-signed SSL certificate, this certificate should be added to local storage on the server where WCS is installed:

1. Get self-signed SSL certificate from SIP PBX server

   ```
   openssl s_client -showcerts -connect 192.168.0.153:5061
   ```

   Where

2. `192.168.0.153` - SIP PBX server IP address
3. `5061` - SIP TLS port
4. Copy certificates from the SIP server response

   ```
   Certificate chain
    0 s:/CN=pbx.mycompany.com/O=My Super Company
      i:/CN=Asterisk Private CA/O=My Super Company
   -----BEGIN CERTIFICATE-----
   ... SIP server certificate goes here
   -----END CERTIFICATE-----
    1 s:/CN=Asterisk Private CA/O=My Super Company
      i:/CN=Asterisk Private CA/O=My Super Company
   -----BEGIN CERTIFICATE-----
   ```

```
... SIP server CA certificate goes here
-----END CERTIFICATE-----
```

then add them to `pbx.crt` file. The file content should be like this:

```
-----BEGIN CERTIFICATE-----
... SIP server certificate goes here
-----END CERTIFICATE-----
-----BEGIN CERTIFICATE-----
... SIP server CA certificate goes here
-----END CERTIFICATE-----
```

5. Detect Java home path

```
readlink -f $(which java)
```

For example, if the command above returned `/usr/java/jdk1.8.0_181/bin/java`, then Java is installed to the folder `/usr/java/jdk1.8.0_181/`

6. Find Java local certificate storage file path, for example

```
find /usr/java/jdk1.8.0_181/jre/lib/security/cacerts
```

7. Import the certificates retrieved on step 2 to Java local certificate storage

```
keytool -importcert -keystore
/usr/java/jdk1.8.0_181/jre/lib/security/cacerts -storepass changeit -file
pbx.crt -alias "pbx"
```

8. Restart WCS

Connection to an existing session

Sometimes it is necessary to connect to already existing session and receive an incoming call. It is usually actual on mobile devices where websocket session is closed automatically when browser goes to background. In this case, only push notifications are available. To keep the sesssion active after disconnection, the `keepAlive` option should be set while creating the session

```
var connectionOptions = {
    urlServer: url,
    keepAlive: true,
    sipOptions: sipOptions
};
...
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session, connection){
```

```
    ...
});
```

In this case, session stays active until the following interval in milliseconds is expired (3600 seconds, or 1 hour by default)

```
client_timeout=3600000
```

This interval is periodically checked. The checking period is set in milliseconds by the following parameter (300 seconds, or 5 minutes by default)

```
client_timeout_check_interval=300000
```

A session token should be stored while creating the session

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session, connection){
    authToken = connection.authToken;
    ...
});
```

Then application may connect to the session again using this token (for example, if incoming call push notification is received):

```
var connectionOptions = {
    urlServer: url,
    keepAlive: true
};

if (authToken) {
    connectionOptions.authToken = authToken;
} else {
    connectionOptions.sipOptions = sipOptions;
}

Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED,
function(session, connection){
    ...
});
```

Known issues

1. It's impossible to make a SIP call if `SIP Login` and `SIP Authentification name` fields contain unappropriate characters

Symptoms

SIP call stucks in `PENDING` state

Solution

According to RFC3261, `SIP Login` and `SIP Authentification name` should not contain any of unescaped spaces and special symbols and should not be enclosed in angle brackets `<>`.

For example, this is not allowed by the specification

```
sipLogin='Ralf C12441@host.com'
sipAuthenticationName='Ralf C'
sipPassword='demo'
sipVisibleName='null'
```

and this is allowed

```
sipLogin='Ralf_C12441'
sipAuthenticationName='Ralf_C'
sipPassword='demo'
sipVisibleName='Ralf C'
```

2. There may be some problems with sound in SIP calls established from Edge browser

Symptoms

a) The outgoing sound is sometimes abruptly muffled, then it goes normally.

b) The incoming sound is heard only if you speak into the microphone.

Solution

Switch SILK and G.722 codecs usage off in SIP calls for Edge browser with `stripCodecs` option:

```
var outCall = session.createCall({
    callee: $("#callee").val(),
    visibleName: $("#sipLogin").val(),
    ...
    stripCodecs: "silk,g722"
    ...
});

outCall.call();
```

or with server setting

```
codecs_exclude_sip=g722,mpeg4-generic,flv,mpv
```

3. Microphone switching does not work in Safari browser.

Symptoms

Microphone is not switching using `switchMic()` WebSDK method

Solution

Use other browser, because Safari always uses sound input microphone, that is chosen in system sound menu (hold down the `Option` (`Alt`) button and click on the sound icon in the menu bar). When microphone is chosen in sound menu, Mac reboot is required.

If Logitech USB camers microphone does not work (when it is chosen in sound menu), format / sample rate changing in Audio MIDI Setup and rebooting can help.

4. Outgoing video SIP call cannot be established if INVITE SDP size exceeds MTU

Symptoms

SIP server return `408 Reques timeout` when trying to establish video SIP call, audio calls can be established successfully through the same server.

Solution

Reduce the number of codecs in the INVITE SDP so that the SDP fit into the packet size defined by MTU (usually 1500 bytes) using the following settings

```
codecs_exclude_sip=mpeg4-generic,flv,mpv,opus,ulaw,h264,g722,g729
allow_outside_codecs=false
```

Only codecs supported by both sides of the call should be left, in this case it is VP8 and PCMA (`alaw`).

5. There is no sound in browser if caller makes an audio+video call and callee responds with audio only

Symptoms

There is no sound in caller browser if caller makes an audio+video call (for instance, using Phone Video example) and callee responds with audio only (for instance, a call to IVR)

Solution

Update WCS to build 5.2.1672 and enable video frames generator

```
generate_av_for_ua=all
```

6. IVR greeting plays not from beginning if caller makes audio+video call

Symptoms

There is a gap before IVR greeting starts playing if caller makes an audio+video call (for instance, using Phone Video example)

Solution

Update WCS to build 5.2.1755 and reduce video frames generator start timeout

```
generate_av_for_ua=all
rtp_generator_start_timeout=100
```

7. An excessive video transcoding to VP8 starts when making a call between browsers

Symptoms

A video traffic is receiving from SIP server in H264 codec, but video is playing as VP8 in browser

Solution

a) Add the following parameter to WCS settings

```
profiles=42e01f,640028
```

b) Add the following parameter if above does not help

```
proxy_use_h264_packetization_mode_1_only=false
```