

Phone Video

- [Example of web phone for video calls](#)
- [Code of the example](#)
- [Analyzing the code](#)

Example of web phone for video calls

Phone Video

Connection

WCS URL: ws://localhost:8080

SIP Login: 10006

SIP Auth Name: 10006

SIP Password:


SIP Domain: sip.flashphoner.com

SIP Outbound Proxy: sip.flashphoner.com

SIP Port: 5060

Register required:

REGISTERED Disconnect



Video size: 320 x 240

Video FPS: 30

Mute Audio: off

Mute Video: off

SDP replace: 0 with 0

Statistics Video

Bytes sent: 439775

Packets sent: 545

Statistics Audio

Bytes sent: 88193

Packets sent: 1379

10008 Hangup

Code of the example

The path to the source code of the example on WCS server is:

`/usr/local/FlashphonerWebCallServer/client/examples/demo/sip/phone-video`

phone-video.css - file with styles
phone-video.html - page of the web phone
call-fieldset.html - form with fields required for connection
call-controls.html - HTML code for call controls
phone-video.js - script providing functionality for the web phone

This example can be tested using the following address:

`https://host:8888/client/examples/demo/sip/phone-video/phone-video.html`

Here host is the address of the WCS server.

Analyzing the code

To analyze the code, let's take the version of file phone-video.js with hash ecbadc3, which is available [here](#) and can be downloaded with corresponding build [2.0.212](#).

1. Initialization of the API

Flashphoner.init() [code](#)

```
Flashphoner.init();
```

2. Connection to server.

Flashphoner.createSession() [code](#)

Object with connection options is passed to the method

- urlServer - URL for WebSocket connection to WCS server
- sipOptions - object with parameters for SIP connection

```
var url = $('#urlServer').val();
var registerRequired = $('#sipRegisterRequired').is(':checked');

var sipOptions = {
    login: $("#sipLogin").val(),
    authenticationName: $("#sipAuthenticationName").val(),
    password: $("#sipPassword").val(),
    domain: $("#sipDomain").val(),
    outboundProxy: $("#sipOutboundProxy").val(),
    port: $("#sipPort").val(),
    registerRequired: registerRequired
};

var connectionOptions = {
    urlServer: url,
    sipOptions: sipOptions
};

//create session
console.log("Create new session with url " + url);
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session){
    ...
});
```

3.Receiving the event confirming successful connection

ConnectionStatusEvent ESTABLISHED [code](#)

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session){
    setStatus("#regStatus", SESSION_STATUS.ESTABLISHED);
    onConnected(session);
    if (!registerRequired) {
        disableOutgoing(false);
    }
}).on(SESSION_STATUS.REGISTERED, function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    ...
});
```

4.Receiving the event confirming successful registration on SIP server

ConnectionStatusEvent REGISTERED [code](#)

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session){
    ...
}).on(SESSION_STATUS.REGISTERED, function(session){
    setStatus("#regStatus", SESSION_STATUS.REGISTERED);
    onConnected(session);
    if (registerRequired) {
        disableOutgoing(false);
    }
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    ...
});
```

5.Receiving the event on incoming call

ConnectionStatusEvent INCOMING_CALL [code](#)

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session){
    ...
}).on(SESSION_STATUS.REGISTERED, function(session){
    ...
}).on(SESSION_STATUS.DISCONNECTED, function(){
    ...
}).on(SESSION_STATUS.FAILED, function(){
    ...
}).on(SESSION_STATUS.INCOMING_CALL, function(call){
    call.on(CALL_STATUS.RING, function(){
        ...
    });
    onIncomingCall(call);
});
```

6. Outgoing call.

session.createCall(), call.call() [code](#)

The following parameters are passed when call is created

- callee - callee SIP username
- visibleName - display name
- localVideoDisplay - <div> element, in which video from camera will be displayed
- remoteVideoDisplay - <div> element, in which video from the other party will be displayed

```
var outCall = session.createCall({
    callee: $("#callee").val(),
    visibleName: $("#sipLogin").val(),
    localVideoDisplay: localVideo,
    remoteVideoDisplay: remoteVideo,
    localVideoDisplay: localVideo,
    constraints: constraints,
    sdpHook: rewriteSdp,
    stripCodecs: "SILK"
    ...
});

outCall.call();
```

7. Answering incoming call.

call.answer() [code](#)

Object with answer options is passed to the method

- localVideoDisplay - <div> element, in which video from camera will be displayed
- remoteVideoDisplay - <div> element, in which video from the other party will be displayed

```

$("#answerBtn").off('click').click(function(){
    $(this).prop('disabled', true);
    inCall.answer({
        localVideoDisplay: localVideo,
        remoteVideoDisplay: remoteVideo,
        constraints: constraints,
        sdpHook: rewriteSdp,
        stripCodecs: "SILK"
    });
    showAnswered();
}).prop('disabled', false);

```

8. Outgoing call hangup.

call.hangup() [code](#)

```

$("#callBtn").text("Hangup").off('click').click(function(){
    $(this).prop('disabled', true);
    outCall.hangup();
}).prop('disabled', false);

```

9. Incoming call hangup

call.hangup() [code](#)

```

$("#hangupBtn").off('click').click(function(){
    $(this).prop('disabled', true);
    $("#answerBtn").prop('disabled', true);
    inCall.hangup();
}).prop('disabled', false);

```

10. Call hangup on session disconnection

call.hangup() [code](#)

```

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

```

11. Mute/unmute

currentCall.muteAudio(), currentCall.unmuteAudio(), currentCall.muteVideo(), currentCall.unmuteVideo() [code](#)

```

// Mute audio in the call
function mute() {
    if (currentCall) {
        currentCall.muteAudio();
    }
}

// Unmute audio in the call
function unmute() {
    if (currentCall) {
        currentCall.unmuteAudio();
    }
}

// Mute video in the call
function muteVideo() {
    if (currentCall) {
        currentCall.muteVideo();
    }
}

// Unmute video in the call
function unmuteVideo() {
    if (currentCall) {
        currentCall.unmuteVideo();
    }
}

```

12. WebRTC statistics collection during the call

`call.getStats()` [code](#)

```

function loadStats() {
    if (currentCall) {
        // Stats should be collected for active calls only #WCS-3260
        let status = currentCall.status();
        if (status !== CALL_STATUS.ESTABLISHED && status !== CALL_STATUS.HOLD) {
            return;
        }
        currentCall.getStats(function (stats) {
            if (stats && stats.outboundStream) {
                if (stats.outboundStream.video) {
                    $('#videoStatBytesSent').text(stats.outboundStream.video.bytesSent);
                    $('#videoStatPacketsSent').text(stats.outboundStream.video.packetsSent);
                } else {
                    $('#videoStatBytesSent').text(0);
                    $('#videoStatPacketsSent').text(0);
                }

                if (stats.outboundStream.audio) {
                    $('#audioStatBytesSent').text(stats.outboundStream.audio.bytesSent);
                    $('#audioStatPacketsSent').text(stats.outboundStream.audio.packetsSent);
                } else {
                    $('#audioStatBytesSent').text(0);
                    $('#audioStatPacketsSent').text(0);
                }
            }
        });
    }
}

```