

# Phone

Пример вебфона для аудиозвонков

## Phone Min

### Connection

WCS URL	<input type="text" value="wss://test.flashphoner.com:8443"/>
SIP Login	<input type="text" value="001"/>
SIP Password	<input type="text" value="..."/>
SIP Domain	<input type="text" value="test.flashphoner.com"/>
SIP Port	<input type="text" value="5060"/>
Register required	<input type="checkbox"/>

ESTABLISHED

Mute  off

ESTABLISHED

## Код примера

Код данного примера находится на WCS-сервере по следующему пути:

```
/usr/local/FlashphonerWebCallServer/client2/examples/demo/sip/phone
```

phone.css - файл стилей

phone.html - страница вебфона

call-fieldset.html - форма с полями, необходимыми для установления соединения

call-controls.html - элементы для управления звонками

phone.js - скрипт, обеспечивающий работу вебфона

Тестировать данный пример можно по следующему адресу:

```
https://host:8888/client2/examples/demo/sip/phone/phone.html
```

Здесь host - адрес WCS-сервера.

## Работа с кодом примера

Для разбора кода возьмем версию файла phone.js с хешем 66сс393, которая находится [здесь](#) и доступна для скачивания в соответствующей сборке [0.5.28.2753.133](#).

### 1. Инициализация API.

Flashphoner.init() [code](#)

```
Flashphoner.init({flashMediaProviderSwfLocation: '../.../.../media-provider.swf'});
```

### 2. Подключение к серверу.

Flashphoner.createSession() [code](#)

Методу createSession() передается объект с параметрами для подключения

- urlServer - URL для WebSocket-соединения с WCS-сервером
- sipOptions - объект с параметрами для SIP-соединения

```
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
};

if (authToken) {
    connectionOptions = {
        urlServer: url,
        authToken: authToken,
        keepAlive: true
    };
} else {
    connectionOptions = {
        urlServer: url,
        sipOptions: sipOptions,
        keepAlive: true
    };
}

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

### 3. Получение от сервера события, подтверждающего успешное соединение.

ConnectionStatusEvent ESTABLISHED [code](#)

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session, connection){
    setStatus("#regStatus", SESSION_STATUS.ESTABLISHED);
    $("#authToken").val(connection.authToken);
    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. Получение от сервера события, подтверждающего успешную регистрацию на SIP-сервере.

ConnectionStatusEvent REGISTERED [code](#)

```
Flashphoner.createSession(connectionOptions).on(SESSION_STATUS.ESTABLISHED, function(session, connection){
    ...
}).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. Получение от сервера события, сигнализирующего о входящем звонке.

ConnectionStatusEvent INCOMING\_CALL [code](#)

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

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

При создании передаются следующие параметры

- callee - SIP имя пользователя вызываемого
- visibleName - отображаемое имя
- localVideoDisplay - div-элемент локального дисплея (в случае Flash медиа-провайдера будет использован для диалога настроек Flash-плеера)
- remoteVideoDisplay - div-элемент для получаемого аудио
- constraints - ограничения для звонка (в данном случае параметр 'video' установлен в 'false' для аудиозвонка)

- receiveAudio - установлен в 'true' для получения аудио
- receiveVideo - установлен в 'false', чтобы не получать видео

```

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

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

outCall.call()

```

## 7. Ответ на входящий звонок.

call.answer() [code](#)

Методу передается объект с опциями для ответа

- localVideoDisplay - div-элемент локального дисплея
- remoteVideoDisplay - div-элемент для получаемого аудио

```

$("#answerBtn").off('click').click(function(){
  $(this).prop('disabled', true);
  var constraints = {
    audio: true,
    video: false
  };
  inCall.answer({
    localVideoDisplay: localDisplay,
    remoteVideoDisplay: remoteDisplay,
    receiveVideo: false,
    constraints: constraints,
    stripCodecs:"SILK"
  });
  showAnswered();
}).prop('disabled', false);

```

## 8. Завершение исходящего звонка.

call.hangup() [code](#)

```

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

```

## 9. Завершение входящего звонка.

call.hangup() [code](#)

```

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

```

## 10. Завершение текущего звонка при закрытии соединения

`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);
}
```

## 11. Включение/выключение аудио.

`currentCall.muteAudio()`, `currentCall.unmuteAudio()` [code](#)

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

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