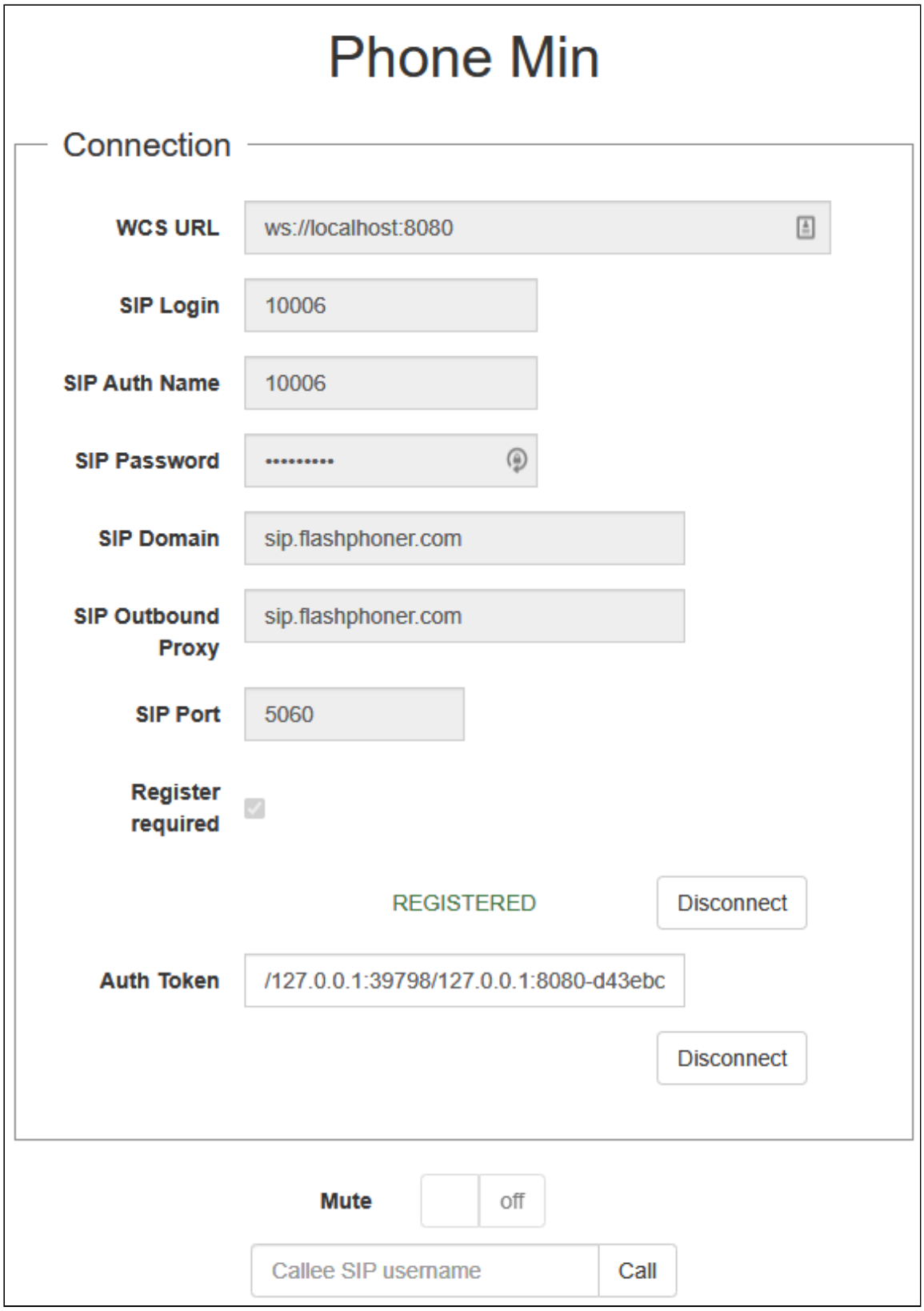
Phone

Example of web phone for audio calls



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

- phone.css - file with styles
- phone.html - page of the web phone
- call-fieldset.html - form with fields required for connection
- call-controls.html - HTML code for call controls
- phone.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/phone.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.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

```
function createSession(authToken) {
    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,
        keepAlive: true
    };

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

    //create session
    console.log("Create new session with url " + url);

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

3. Receiving the event confirming successful connection

`ConnectionStatusEvent ESTABLISHED` code

On this event, the token is stored to connect to the same session in 1 hour

```
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. Receiving the event confirming successful registration on SIP server

`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. Receiving the event on incoming call

`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. 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 for local audio
- `remoteVideoDisplay` - `div` element for remote audio
- `constraints` - constraints for the call (`video` set to `false` to make audio only call)
- `receiveAudio` - set to `true` to receive audio
- `receiveVideo` - set to `false` to receive audio only

```
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. Answering incoming call

`Call.answer()` code

Object with answer options is passed to the method

- `localVideoDisplay` - `div` element for local audio
- `remoteVideoDisplay` - `div` element for remote audio

```
$("#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. 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,
#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. Mute/unmute audio

`Call.muteAudio()`, `Call.unmuteAudio()` code

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

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