2026

Mizu WebPhone

Cross-Platform SIP for browsers

Mizu-WebSIPPhone is a standard based VoIP client running from browsers as a ready to use softphone or usable as a JavaScript library. Based on the industry standard SIP protocol, it is compatible with all common VoIP devices.

One software for all OS and all browsers
The ultimate Browser to SIP solution

Version 4.4
Mizutech
6/17/2026

Contents

About 3

Usage 3

Steps 3

Web Softphone 4

Click-to-call 5

Developers 5

Designers 5

Features, technology and licensing 5

Feature list 5

Requirements 6

Technical details 6

Licensing 8

Integration and customization 9

Concepts 9

User interface Skin/Design 10

Web Dialer/Softphone 11

Click to call 16

Custom solutions 19

Integration with Web server side applications or scripts 19

Integration with SIP server side applications or scripts 21

Development 21

Parameters 24

SIP account settings 26

Engine related settings 28

Call divert and other settings 45

User interface related settings 54

Parameter security 62

JavaScript API 63

About 63

Basic example 63

Functions 64

Events 76

FAQ 84

Resources 145

About

The Mizu WebPhone is a universal JavaScript SIP client library to provide VoIP capability for **all browsers** using a variety of technologies compatible with most OS/browsers. Since it is based on the open standard SIP and RTP protocols, it can inter-operate with any other SIP-based network, allowing people to make true VoIP calls directly from their browsers. Compatible with all SIP softphones (X-Lite, Bria, Jitsi others), devices (gateways, ATA's, IP Phones, others), proxies (SER, OpenSIPS, others), PBX (Asterisk, FreeSWITCH, FreePBX, Elastix, Avaya, FusionPBX, 3CX, Broadsoft, Alcatel, NEC, others), VoIP servers (Mizu, Voipswitch, Cisco, Huawei, Kamailio, others), service providers (Twilio, Vonage and others) and any SIP capable endpoint (UAC, UAS, proxy, others).

The Mizu WebPhone is truly **cross-platform**, running from both desktop and mobile browsers, offering the best browser to SIP phone functionality in all circumstances, using a variety of built-in technologies referred as "engines":

- NS (Native Service/Plugin)
- WebRTC
- Java applet
- Flash
- App
- P2P/Callback
- Native Dial

The engine to use is automatically selected by default based on OS, browser and server availability (It can be also set manually from the configuration or priorities can be changed). This multi-engine capability has considerable benefits over other naive implementations.
The webphone can be used with the provided user interface (as a ready to use **softphone** or **click to call** button) or as a **JavaScript SIP library** via its API.
The provided user interfaces are implemented as simple HTML/CSS and can be fully customized, modified, extended or removed/replaced.

Usage

The webphone is an all-in-one VoIP client module which can be used as-is (as a ready to use softphone or click to call) or as a JavaScript SIP library (to implement any custom VoIP client or add VoIP call capabilities to existing applications). You can create custom VoIP solutions from scratch with some JavaScript knowledge or use it as a turn-key solution if you don't have any programming skills as the webphone is highly customizable by just changing its numerous settings.

Setup

In short:

Download the webphone.zip and extract it to your web server (or test it from your local file system by just launching it from your desktop for example).
Open the webphone_config.js file and configure at least the serveraddress parameter.
Launch any of the example html included with the webphone, for example the index.html as from there you can access most examples.
Enter your SIP username/password (if you haven't preconfigured it in the webphone_config.js) and call any account/extension/phone number.

A quick tutorial can be found here and/or you might follow the below step by step detailed instructions:

1. Download

The package can be downloaded from here: webphone download.
It includes everything you need for a browser to SIP solution: the engines, the JavaScript API, the skins and also a few usage examples.
This public downloadable version has some demo limitations, but you can use it for any tests or integration work before to purchase your license.
If you already have a webphone license, then use the webphone package sent to you by Mizutech instead of the above link.

2. Deploy

You can find the requirements here, which needs to be fulfilled to be able to use the webphone.
There is no any web server related requirements. You can host the webphone on any web server the exact same way as you host a static page or a png images.
Unzip and copy the webphone folder into your webserver and refer it from your html (for example from your main page) or open one of the included html in your browsers by specifying its exact URL.
For example: http://192.168.1.44/webphone/samples/techdemo_example.html or https://yourdomain.com/webphone/softphone.html .

Notes:

- *You might have to enable the .jar, .exe, .zip, .swf, .dll, .so, .pkg, .dmg and .dylib mime types in your webserver if not enabled by default (these files might be used in some circumstances depending on the client OS/browser).*
- *If you wish to use (also) the WebRTC engine then your site should be secured (HTTPS with a valid SSL certificate). Modern browsers requires secure connection for both your website (HTTPS) and websocket (WSS). If your website doesn't have an SSL certificate then we can host the webphone for you for free or you can install a cheap or free certificate.*
- *More details can be found here*
- *Alternatives:*
  - *You can also test it without a webserver by launching the html files from your desktop (launch from local file system)*

- *You can also test it by using the online demo hosted by Mizutech website, but in this case you will not be able to change the configuration (you can still set any parameters from the user interface or from URL)*

3. Settings

The webphone has a long list of parameters which you can set to customize it after your needs.
You can set these parameters multiple ways (in the webphone_config.js file, pass by URL parameter or via the setparameter() API).
If you are using the webphone with a SIP server (not peer to peer) then you must set at least the "serveraddress" parameter. For more details read here.
The easiest way to start is to just enter the required parameters (serveraddress, username, password and any other you might wish) in the webphone_config.js file.

4. Integrate

The webphone can be used as a turn-key ready to use solution (just refer to it from your HTML) or as a Java-Script library to develop custom software.
There are multiple ways to use it:

- Use one of the supplied templates (the "softphone" or the "click to call") and customize it after your needs. You can place one of them as an iframe or div to your website
- Integrate the webphone with your webpage, website or web application
- Integrate the webphone with your server side application (if you are a server side developer)
- Create your custom solution by using the webphone as a JavaScript SIP library (if you are a JavaScript developer)

5. Design

If you are a designer then you might create your own design or modify the existing HTML/CSS. For simple design changes you don't need to be a designer.
Colors, branding, logo and others can be specified by the settings for the supplied "softphone" and "click to call" skins.
Mizutech can also supply a ready to use pre-customized build of the softphone skin with your settings and branding for no extra cost (ask for it).
Please note that the webphone also works without any GUI.

6. Launch

Launch one of the examples (the html files in the webphone folder) or your own html (from desktop by double clicking on it or from browser by entering the URL). You might launch the "index.html" to see the included templates.
At first start the webphone might offer to enable or download a native plugin if no other suitable engine are supported and enabled by default in your browser.
It will also ask for a SIP username/password if you use the default GUI and these are not preset.
On init, the webphone will register (connect) to your VoIP server (this can be disabled if not needed by setting the register parameter to 0).
Then you should be able to make calls to other UA (any webphone or SIP endpoint such as X-Lite or other softphone) or to pstn numbers (mobile/landline) if outbound call service is enabled by your server or VoIP provider.

Examples and ready to use solutions (found in the webphone folder):

- **index**.html: just an index page with direct links to the below examples for your convenience
- **minimal_example**.html: shortest example capable to make a call
- **basic_example**.html: a basic usage example
- **techdemo_example**.html: a simple tech demo. You might make any tests by using this html or change/extend it to fit your needs
- **softphone**.html: a full featured, ready to use browser softphone. You can use it as is on your website as a web dialer. For example you can include it in an iframe or div on your website. Change the parameters in the webphone_config.js). You can further customize it by changing the parameters or changing its design.
- **softphone_launch**.html: a simple launcher for the above to look better when launched directly (since the softphone.html is used usually in a DIV or iFrame)
- **click2call**.html: a ready to use browser to SIP click to call solution. You might further customize after your needs
- **custom**: you can easily create any custom browser VoIP solution by using the webphone java script SIP library

More details:

- Webphone file list
- Customization
- How it works?
- Quick webphone tutorial

Web Softphone

The webphone package contains a ready to use web softphone.

Just copy the webphone folder to your webserver and change the "serveraddress" setting in the in webphone_config.js file to your SIP server IP or domain to have a fully featured softphone presented on your website. You can just simply include (refer to) the softphone.html via an iframe (this way you can even preset the webphone parameters in the iframe URL) div or on demand.
Note: you might have to enable the following mime types in your web server if not enabled by default: .jar, .swf, .dll, .dylib, .so, .pkg, .dmg, .exe

The web softphone can be configured via URL parameters or in the " webphone_config.js" file, which can be found in the root directory of the package. The most important configuration is the “serveraddress” parameter which should be set to your SIP server IP address or domain name. More details about the parameters can be found below in this documentation in the “Parameters” section.

We can also send you a build with all your branding and settings pre-set: contact us.
Check here for more options.

Click-to-call

The webphone package contains a ready to use click to call solution.

Just copy the whole webphone folder to your website, set the parameters in the webphone_config.js file and use it from the click2call.html.
Rewrite or modify after your needs with your custom button image or you can just use it via a simple URI or link such as:
http://www.yourwebsite.com/webphonedir/click2call.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_callto=CALLEDNUMBER&wp_autoaction=1
You can find more details and customization options in the click to call section.

Developers

Developers can use the webphone as a JavaScript SIP library to create any custom VoIP solution integrated in any webpage or web application.
Just include the " webphone_config.js" to your project or html and start using the webphone API.

See the development section for the details.

Note: You can adjust the webphone behavior also without any development knowledge by just modifying its settings described in “parameters” section.

Designers

If you are a designer, you can modify all the included HTML/CSS/images or create your own design from scratch using any technology that can bind to JS such as HML5/CSS, Flash or others.

For simple design changes you don’t need to be a designer. Colors, branding, logo and others can be set by the settings.
See the “User Interface Skin/Design” section for more details.

Note: If you are using the built-in softphone or click2call skins, then you can adjust them also without any designer knowledge by just setting/modifying the user interface related settings in the webphone_config.js file.

Features, technology and licensing

The WebPhone is a cross-platform JavaScript SIP client library running entirely in clients browsers compatible with all browsers and all SIP servers, IP-PBX or softswitch. The webphone is completely self-hosted without any cloud dependencies, completely owned and controlled by you (just copy the files to your Web server).

Feature list

- Standard SIP voice calls (in/out), video, chat, conference and others (Session Initiation Protocol)
- Maximum browsers compatibility. Runs in all browsers with Java, WebRTC or native plugin support (WebView, Chrome, Firefox, IE, Edge, Safari, Opera)
- Includes several different technologies to make phone calls (engines): WebRTC, NS (Native Service or Plugin), Java applet, Flash, App, Native and server assisted conference rooms, calls through IVR, P2P and callback
- SIP and RTP stack compatible with all standard VoIP servers and devices like Cisco, Voipswitch, Asterisk, softphones, ATA, IP phones and others
- Transport protocols: UDP, TCP, TLS, WS, WSS. Both IPv4 and IPv6 are supported.
- Encryption: SIPS, TLS, DTLS, SRTP and end to end encryption for webphone to webphone calls even if TLS/DTLS/SRTP is not supported
- NAT/Firewall support: auto detect transport method (UDP/TCP/HTTP), stable SIP and RTP ports, keep-alive, rport support, proxy traversal, auto tunneling when necessary, ICE/STUN/TURN protocols and auto configuration, firewall traversal for corporate networks, VoIP over HTTP/TCP when firewalls blocks the UDP ports with full support for ICE TCP candidates
- Works over the internet and also on local LAN’s (perfectly fine to be used with your own internal company PBX)
- RFC’s: 2543, 3261, 7118, 2976, 3892, 2778, 2779, 3428, 3265, 3515, 3311, 3911, 3581, 3842, 1889, 2327, 3550, 3960, 4028, 3824, 3966, 2663, 6544, 5245 and others
- Supported methods: REGISTER, INVITE, re-INVITE, ACK, PRACK, BYE, CANCEL, UPDATE, MESSAGE, INFO, OPTIONS, SUBSCRIBE, NOTIFY, REFER
- Audio Codec: G.711 (PCMU, PCMA), G.729, GSM, iLBC, iSAC, SPEEX, OPUS (including wide-band HD audio)
- Video codec: H.263, H.264, H.265, VP8 and VP9 for WebRTC only
- SIP compatible codec auto negotiation and adjustment (for example OPUS – G.711, G.729 - wideband or RTC G.711 to G.729 transcoding if needed)
- Video calls and Screen-sharing (for WebRTC enabled browsers)

- Call divert: rewrite, redial, mute, hold, transfer, forward, conference and other PBX features
- Call park and pickup, auto-answer, barge-in
- Voice call recording
- IM/Chat (RFC 3428), SMS, file transfer, DTMF, voicemail (MWI)
- Multi-line support
- Multi account support
- Contacts: management, synchronization, flags, favorites, block, auto prioritization
- Presence (DND/online/offline/others) via standard SIP PUBLISH/SUBSCIBE/NOTIFY
- NS BLF (Busy Lamp Field)
- Balance display, call timer, inbound/outbound calls, caller-id display
- High level JavaScript API: web developers can easily build any custom VoIP functionality using the webphone as a JS SIP library
- Stable API: new releases are always backward compatible so you can always upgrade to new versions with no changes in your code
- Full control: perpetual life-time license self-hosted as you wish
- Integration with any website or application including simple static pages, apps with client side code only (like a simple static page) or any server side stack such as PHP, .NET, java servlet, J2EE, Node.js and others (sign-up, CRM, callcenter, payments and others)
- Phone API accessible from any JavaScript framework (such as AngularJS, React, jQuery and others) or from plain/vanilla JS or not use the JS API at all
- Branding and customization: customizable user interface with your own brand, skins and languages (with ready to use, modifiable themes)
- Flexibility: all parameters/behavior can be changed/controlled by URL parameters, preconfigured parameters, from javascript or from server side

The webphone is in continuous development with multiple new releases every year.
Only the major releases are published on the software homepage as a demo (usually once per year), but new customers always receive the latest stable release.
All versions are fully backward/forward compatible, thus upgrades doesn't require any change in existing configurations and code.
For the major new versions you can find the change list here.

Requirements

Server side:

- Any web hosting for the webphone files (any webserver is fine: IIS, nginx, Apache, NodeJS, Java, others; any OS: Windows, Linux, others). Chrome and Opera requires secure connection for WebRTC engine to work (otherwise this engine will be automatically skipped). We can also host the webphone for free if you wish on secure http. Note that the web phone itself doesn't require any framework, just host it as static files (no PHP, .NET, JEE, NodeJS or similar server side scripting is required to be supported by your webserver, however you are free to integrate the webphone with any such server side stacks if you need so)
- At least one **SIP account** at any VoIP service provider or your own SIP server or IP-PBX (such as Asterisk, Voipswitch, 3CX, FreePBX, Trixbox, Elastix, SER, Cisco and others)
- Optional: WebRTC capable SIP server or SIP to WebRTC gateway (Most VoIP servers also has built-in WebRTC support nowadays. If your VoIP server doesn't have WebRTC support, then the mizutech free WebRTC to SIP service is enabled by default. The webphone can be used also without WebRTC)

Client side:

- Any **browser** supporting WebRTC OR Java OR native plugins with JavaScript enabled (most browsers are supported)
- **Audio device**: headset or microphone/speakers

Compatibility:

- OS: Windows (XP,Vista,7,8,10,11) , Linux (most distros), MAC OSX, Android, iOS, BlackBerry, Chromium OS and others
- Browsers: Firefox, Chrome, IE (6+), Edge, Safari, Opera and others
- Different OS/browser might have different compatibility level depending on the usable engines. For example the rarely used Flash engine doesn't implement all the functionalities of the WebRTC/Java/NS engines (these differences are handled automatically by the webphone API)

If you don't have an IP-PBX or VoIP account yet, you can use or test with our SIP VoIP service.

- Server address: voip.mizu-voip.com
- Account: create free VoIP account from here or use the following username/passwords: webphonetest1/webphonetest1, webphonetest2/webphonetest2 (other people might also use these public accounts so calls might be misrouted)

Some more details can be found here.

Technical details

The goal of this project is to implement a VoIP client compatible with all SIP servers, running in all browsers under all OS with the same user interface and API. At this moment no technology exists to implement a VoIP engine to fulfill these requirements due to browser/OS fragmentation. Also different technologies have some benefits over others. We have achieved this goal by implementing different "VoIP engines" targeting each OS/browser segment. This also has the advantage of just barely run a VoIP call, but to offer the best possible quality for all environments (client OS/browser). All these engines are covered by a single, easy to use unified API accessible from JavaScript. To ease the usage, we also created a few different user interfaces in HTML/JS/CSS addressing the most common needs such as a VoIP dialer and a click to call user interface.

More details about how it works can be found here.

Each engine has its advantages and disadvantages. The sip webphone will automatically choose the "best" engine based on your preferences, OS/Browser/server side support and the enduser preferences (this can be overridden by settings if you have some special requirements): VoIP availability in browsers.

Engines

NS

Native VoIP engine implemented as a service or browser plugin. The engine works like a usual SIP client, connecting directly from client PC to your SIP server, but it is fully controlled from web (the client browser will communicate in the background with the native engine installed on the client pc/mobile device, thus using this natively installed sip/media stack for VoIP).
Available for: all browsers on Windows, Linux
Pros:

- All features all supported, native performance

Cons

- Requires install (one click installer)

WebRTC

A new technology for media streaming in modern browsers supporting common VoIP features. WebRTC is a built-in module in modern browsers which can be used to implement VoIP. Signaling goes via websocket (unencrypted or TLS) and media via encrypted UDP (DTLS-SRTP). These are then converted to normal SIP/RTP by the VoIP server or by a gateway.
Available for: all modern browsers.
Pros:

- Comfortable usage in browsers with WebRTC support because it is has no dependencies on plugins

Cons:

- It is a black-box in the browser with a restrictive API
- Lack of popular VoIP codec such as G.729 (this can be solved by CPU intensive server side transcoding)
- A WebRTC to SIP gateway may be required if your VoIP server don't have built-in support for WebRTC (there are free software for this and we also provide a free service tier, included by default)

Flash

A browser plugin technology developed by Adobe with its proprietary streaming protocol called RTMP.
Available for: old browsers.
Pros:

- In some old/special browsers only Flash is available as a single option to implement VoIP

Cons:

- Requires server side Flash to SIP gateway to convert between flash RTMP and SIP/RTP (we provide free service tier)
- Basic feature set

Java Applet

Based on our powerful JVoIP SDK, compatible with all JRE enabled browsers. Using the Java Applet technology you can make SIP calls from browsers the very same way as a native dialer, connecting directly from client browser to SIP server without any intermediary service (SIP over UDP/TCP and RTP over UDP).
Available for: java enabled browsers such as IE, Pale Moon and old Firefox/Chrome.
Pros:

- All SIP/media features are supported, all codecs including G.729, wideband and custom extra modules such as call recording
- Works exactly as a native softphone or ip phone connecting directly from the user browser to your SIP capable VoIP server or PBX (but with your user interface)

Cons:

- Java is getting deprecated by modern browsers (handled automatically by the webphone by just choosing another available engine)
- Some browsers may ask for user permission to activate the Java plugin

App

Some platforms don't have any suitable technology to enable VoIP in browsers (a minor percentage, most notably old iOS/Safari). In these cases the webphone can offer to the user to install a native softphone application. The apps are capable to fully auto-provision itself based on the settings you provide in your web application so once the user installs the app from the app store, then on first launch it will magically auto-provision itself with most of your settings/branding/customization/user interface as you defined for the webphone itself.
Available for: old browsers with no any built-in or plugin based VoIP support.
Pros:

- Covering platforms with lack of VoIP support in browsers (most notable old iOS Safari)

Cons:

- No API support. Not the exactly same HTML user interface (although highly customized based on the settings you provided for the webphone)

P2P and callback

These are just "virtual" engines with no real client VoIP stack.

- P2P means server initiated phone to phone call initiated by an API call into your VoIP server. Then the server will first call you (on your regular mobile/landline phone) and once you pick it up it will dial the other number and you can start talking (Just set the "p2p" setting to point to your VoIP server API for this to work)
- Callback is a method to make cheap international calls triggering a callback from the VoIP server by dialing its callback access number. It might be used only as a secondary engine if you set a callback access number (Just set the "callback" setting to point to your VoIP server API for this to work)

These are treated as a secondary (failback) engines and used only if no other engines are available just to be able to cover all uncommon/ancient devices with lack of support for all the above engines which is very rare. However it might be possible that these fits into your business offer and in that case you might increase their priority to be used more frequently. Alternatively it might be used also to initiate native call under low quality network conditions.

Native Dial

This means native calls from mobile using your mobile carrier network. This is a secondary "engine" to failback to if no any VoIP capabilities were found on the target platform or there is no network connection. In these circumstances the webphone is capable to simply trigger a phone call from the user smartphone if this option is enabled in the settings. Rarely used if any.

The most important engines are: NS, WebRTC and Java. The other engines are to provide support for exotic or ancient browsers maximizing the coverage for all OS/browser combinations ensuring that endusers always has call capabilities regardless of the circumstances.

API

All the above engines are covered with an easy to use unified Java Script API, hiding all the differences between the engines as described below in the "JavaScript API" section.

GUI

The webphone can be used with or without a user interface.
The user interface can be built using any technology with JS binding. The most convenient is HTML/CSS, but you can also use any others such as Flash.
The webphone package comes with a few prebuilt feature rich responsive user interfaces covering the most common usage such as a full featured softphone user interface and a click to call implementation. You are free to use these as is, modify them after your needs or create your own from scratch. For more details check the "User interface Skin/Design" section.

Licensing

The webphone is sold with life-time unlimited client license (Advanced and Gold) or restricted number of licenses (Basic and Standard). You can use it with any VoIP server(s) on your own and you can deploy it on any webpage(s) which belongs to you or your company. Your VoIP server(s) address (IP or domain name) and optionally your website(s) address will be hardcoded into the software to protect the licensing. You can find the licensing possibilities on the pricing page. After successful tests please ask for your final version at webphone@mizu-voip.com. Mizutech will deliver your webphone build within one workday after your purchase.
Release versions does not have any limitations (mentioned below in the "Demo version" section) and are customized for your domains/brand. All "mizu" and "mizutech" words and links are removed so you can brand it after your needs (with your company name, brand-name or domain name), customize and skin (we also provide a few skin which can be freely used or modified).
Your final build must be used only for you company needs (including your direct sip endusers, service customers or integrated in your software).
Title, ownership rights, and intellectual property rights in the Software shall remain with MizuTech.
The webphone is licensed per server (except the Gold version). Already licensed servers cannot be removed or changed, however, you can use domain names, thus allowing changing your server or its IP address any time later without any implications regarding the webphone license. The exception for the server limitations is the Gold version, which has a more flexible licensing, allowing also unlimited servers.
The agreement and the license granted hereunder will terminate automatically if you fail to comply with the limitations described herein. Upon termination, you must destroy all copies of the Software. The software is provided "as is" without any warranty of any kind. You must accept the software SLA before to use the webphone.
You may:

- Use the webphone on any number of computers, depending on your license
- Give the access to the webphone for your customers or use within your company
- Offer your VoIP services via the webphone
- Integrate the webphone to your website or web application (or native applications usually via a webview)
- Use the webphone on multiple webpage's and with multiple VoIP servers (after the agreement with Mizutech). All the VoIP servers must be owned by you or your company. Otherwise please contact our support to check the possibilities

You may not:

- Resell the webphone as is

- Sell “webphone” services for third party VoIP providers and other companies usable with any third-party VoIP servers (except if coupled with your own VoIP servers)
- Resell the webphone or any derivative work which might compete with the Mizutech “webphone” software offer
- Create competition for Mizutech by reselling the webphone in similar way (not tied to your servers or apps, but as a general browser VoIP client)
- Reverse engineer, decompile or disassemble or modify the software in any way except modifying the settings and the HTML/CSS skins or the included JavaScript examples

Note: It is perfectly fine to sell or promote it as a “webphone service” if that is tied to your own SIP servers. But if you sell it as webphone software which can be used with any server than you are actually selling the same as Mizutech and this is not allowed by the license.

There are the following legal ways to use the webphone:

- *you have your own SIP server(s) and the webphone will be used with these server(s) (your customers can integrate the webphone into any application or website but they will use the webphone via your VoIP server)*
- *you are building some application or website (such as a CRM, portal or game), so the webphone will be tightly integrated with your solution*
- *both of the above (webphone used via your own SIP server from within your own website or application)*
- *click to call for your webpage (or click to call service for others via your SIP server)*
- *your call center or contact center*
- *call center software offered to other companies if the webphone is deeply integrated in your callcenter software or uses your own SIP server for call termination/reception*
- *you are a VoIP service provider and wish to allow your customers to make calls from your website (without the need to install a native client)*

Contact us if you wish to use the webphone in some other way which is not covered by this license or you are in doubt with the usage: webphone@mizu-voip.com

Demo version

The downloadable demo version can be used to try and test before any purchase. The demo version has all features enabled but with some restrictions to prevent commercial usage. The limitations are the followings:

- commercial usage not allowed
- maximum ~10 simultaneous webphone instances at the same time (depending on the engine used)
- might be restricted to up to ~5 or ~10 simultaneous calls (depending on the engine used)
- will expire after several months of usage (around 2 - 5 months, depending on the release date)
- maximum ~100 sec call duration restriction
- maximum ~10 calls / session limitation (after ~10 calls you will have to restart your browser)
- will work maximum ~20 minutes after that you have to restart it or restart the browser
- can be blocked from Mizutech license service

The demo version can be used for all kind of tests or development, but it can’t be used for production. It can be used by up to 5 developers to make short calls (below 100 seconds) and the browser have to be restarted after 10 calls or 20 minutes, otherwise a “Trial” or “License” error will be displayed or printed to the browser console.

Note: for the first few calls and in some circumstances the limitations might be weaker than described above, with fewer restrictions.
All these demo limitations are removed from the licensed versions.
See the pricing and order your licensed copy from here.
On request we can also provide an one month trial without the above demo limitations.

Integration and customization

The webphone is a flexible VoIP web client which can be used for various purposes such as a dialer on your website, a click to call button for contacts or integrated with your web application (contact center, CRM, social media or any other application which requires VoIP calls).

Concepts

The webphone can be customized by its numerous settings, webphone API’s and by changing its HTML/CSS.

Deploy:
The webphone can be deployed as a static page (just copy the webphone file to your website), as a dynamic page (with dynamically generated settings) or used as a JavaScript VoIP library by web developers. You can embed the webphone to your website in a div, in an iFrame or on demand, as a module or as a separate page. The webphone settings can be set also by URL parameters so you can just launch it from a link with all the required settings specified.

VoIP platform:
All you need to use the webphone is a SIP account at any VoIP service provider or your own softswitch/IP-PBX.
Free SIP accounts can be obtained from numerous VoIP service providers or you can use our service. (Note that free accounts are free only for VoIP to VoIP calls. For outbound pstn/mobile you will need to top-up your account).

If you wish to host it yourself then you can use any SIP server software. For example FreePBX for linux or the advanced / free VoIP server for windows by Mizutech. We can also provide our WebRTC to SIP gateway (for free with the Advanced or Gold license) if your softswitch don't have support for WebRTC and you need a self-hosted solution.

Settings:
The webphone settings can be passed multiple ways as described at the beginning of the parameters chapter.
The most important parameter that you will need to set is the "serveraddress" which have to be set to the domain or IP:port of your SIP server.
Other important parameters are listed here.
More details regarding the configuration with your SIP server can be found here.

Integration:
You can integrate the webphone with your web-site or web-application:
-using your own web server API
-and/or using the webphone client side JavaScript API to insert any business logic or AJAX call to your server API
The webphone library doesn't depend on any framework (as it is a pure client side library) but you can integrate it with any server side framework if you wish (PHP, .NET, NodeJS, J2EE or any server side scripting language) or work with it only from client side (from your JavaScript).
On the client side you can use the webphone API from any JavaScript framework (such as AngularJS, React, Vue, Electron, WordPress and others) or from plain/vanilla JS or not use the JS API at all.

You can tightly integrate the web phone with your SIP server using the methods described here.

Design
You can completely change any of the included skins (click to call button, softphone), or change the softphone colors or create your user interface from scratch with your favorite tool and call the webphone API from there.

Custom application:
For deep changes or to create your unique VoIP client or custom application you will need to use the JavaScript API.
See the development section for more details.

Branding:
Since the webphone is usually used within your website context, your website is already your brand and no additional branding is required inside the webphone application itself. However the softphone skin (if you are using this turn-key GUI) has its own branding options which can be set after your requirements.
Additionally you can change the webphone HTML/CSS design after your needs if more modifications are required.

On request, we can send your webphone build already preconfigured with your preferences.
For this just answer the points from the voip client customization page (as many as possible) and send to us by email. Then we will generate and send your webphone build within one work-day. All the preconfigured parameters can be further changed by you via the webphone settings.

Of course, this is relevant only if you are using a skin shipped with the webphone, such as the softphone.html. Otherwise you can create your custom solution using the webphone library with your unique user interface or integrate into your existing website.

User interface Skin/Design

You can use the webphone with or without a user interface.

The webphone is shipped with a few ready to use open source user interfaces such as a softphone and click to call skins. Both of these can be fully customized or you can modify their source to match your needs. You can also create any custom user interface using any technique such as HTML/CSS and bind it to the web phone javascript API.

The default user interface for the softphone and other included apps can be easily changed by modifying parameters or changing the html/css. For simple design changes you don't need to be a designer. Colors, branding, logo and others can be set by the settings parameters.
Also you can easily create your own app user interface from scratch with any tool (HTML/CSS or others) and call the webphone Java Script API from your code.

In short, there are two ways to achieve your own (any kind of) custom user interface:

A. Use one of the skins provided by the webphone
   Here you also have two possibilities:
   - Quick customization changing the webphone built-in user-interface related settings (you can change the colors, behaviors and others)
   - If you are a web developer, then have a look at the html and JavaScript source code and modify it after your needs (we provide all the source code for these; it can be found also in the downloadable demo)

B. Create your own user web VoIP user interface and use the webphone as a JavaScript library for the SIP protocol from there.
   The webphone has an easy to use API which can be easily integrated with any user interface. For example from your "Call" button, just call the webphone webphone_api.js.call(number) function. Have a look at the samples folder: "minimal_example.html", "basic_example.html" or "techdemo_example.html" (You can also use the provided samples as a template to start with and modify/extend it after your needs)

Quick/Basic skin change

Just use the "colortheme" parameter to make quick and wide changes.
Then have a look at the "User interface related" parameters (described in the "Parameters" section) and change them after your needs (set logo, branding, translate and others).

We can also send you a web softphone with your preferred skin. For this just set your customization on the online designer form and send us the parameters.
We can also send you fully customized and branded web softphone with your preferences. For this just send us the customization details.
Read below if you need more customizations.

Advanced skinning

Web developers/designers can easily modify the existing skins or create their own.
For the softphone application all the HTML source code can be found in "softphone.html" file as a single-page application model.
There are a few possibilities to change the skins:

- Change the user interface related parameters as already discussed above (for example the "colortheme" and other parameters)
- You can manually edit the html and css file with your favorite editor to change it after your needs
- Or just create your design with your favorite tools and call the web sip phone API from there
- If you wish to customize the softphone skin (softphone.html), it is discussed in details here

Note: If you are using the webphone as a javascript SIP library then you can customize the "choose engine" popup in "css\pmodal.css".

The webphone can be positioned on your webpage after your needs. You can put it in a DIV or iFrame control and define any position from HTML of CSS. For example you can make it floating as described here.

If you have different needs or don't like the default skins, just create your own from scratch and call the webphone JavaScript API from your code. Using the API you can easily add VoIP call capabilities also to existing website or project with a few function calls as described in the "Java Script API" section below.

For specific needs, look at the following sections:

- Dialer skin
- Click to call skin
- Custom solution
- Floating skin
- Multi-line

Web Dialer/Softphone

With the Mizu webphone you can create a web based softphone which looks like and works like a native desktop softphone application (such as X-Lite). A web based solution has a big advantage since it is always available for endusers without the need to download and install an application and also it is much more flexible than a traditional softphone (tons of parameters, easy to build user interface with HTML/CSS and easy integration with your backend).
The webphone includes a turn-key ready softphone skin (softphone.html) which you can start using in production as-is or change it after your needs.

Possibilities

You can create any HTML/CSS web design (with call controls, contact lists, etc) and call out to the WebPhone API to add the VoIP functionality.
You can start with any custom design from scratch or you might extend the "api_example.html" to reach your goals.
More details can be found here.
There are two possibilities to create your web dialer:

- **Create your web softphone or dialer from scratch**
  You can create any HTML/CSS web design (with call controls, contact lists, etc) and call out to the WebPhone API to add the VoIP functionality.
  Start with any custom design from scratch or you might extend the "api_example.html" (or any other from the samples folder) to reach your goals.
  Basically the purpose of the webphone library is to create any web based SIP client solution and most of this documentation is about describing this process.
  Jump here to begin the development.
- **Modify the existing softphone skin**
  In this chapter below we will explore the possibilities of customizing and extending the softphone skin (softphone.html)

Softphone skin basic customization

An easier solution is to use the turn-key softphone skin shipped with the webphone (softphone.html and related css and js files). This can be used as-is (it is a ready to use turn-key solution) or further modifying after your needs.
No any development knowledge is required to deploy the web softphone on your website and you can also customize it easily via webphone parameters.

Note: You can quickly test this also from the online demo if you haven't downloaded the webphone demo package yet: https://www.webvoipphone.com/webphone_online_demo/softphone_launch.html (this will launch the very same "softphone.html" included in the download package).

If you are a JavaScript developer then you might implement even more modifications by changing the HTML, CSS or JS files (softphone.html, js\softphone files). We can also ship the webphone with ready to use customized softphone skin (with your branding and settings). For this, just answer the points on this page on your order and send the answers by email.

Here is the whole process step by step:

1) Order your webphone from Mizutech with branding (send as many customization details as you can by answering these points).
   At this point you already have a turn-key ready to use solution which can be deployed and used as-is on your website/page/application, without the need of any technical or developer knowledge.
2) Customize your softphone by adjusting the technical parameters after your needs
   This can be easily done by just modifying or adding more parameters in the webphone_config.js. At least the "serveraddress" parameter must be set.
3) Modify the user interface related parameters after your needs
   For example you can set the "colortheme" parameter to make quick and wide change about the skin appearance.
4) Optionally further customize your softphone by making more user interface changes as described here and here.
   This optional step requires some basic HTML/CSS knowledge.
5) Optionally further customize your softphone by integrating it with your client side application, integrating it with your server, integrate with server-side address book or change/add functionalities by using the phone API .
   This optional step requires some development knowledge.

The softphone skin can be positioned on your webpage after your needs. You can put it in a DIV or iFrame control and define any position from HTML of CSS. For example you can make it floating as described here.

Softphone skin advanced customization

If the above described customization options are not enough to achieve your goals, you can modify or extend the softphone HTML, CSS or JavaScript code. The softphone skin can be further customized, with new functionality, new user interface elements including new pages.

Adding a new functionality for the softphone skin usually involves the followings:

- Add some user interface element to the softphone skin. This is described in this chapter below.
- Add functionality for your user interface element. This is usually interaction with the SIP client or interaction with your app server (web service):
  - To interact with the SIP client, just use the webphone SIP API. Learn more here.
  - For the interaction with your backend you can use AJAX requests. Learn more here.

Files

All the webphone SDK library related files are located in the \js\lib\ directory.
The softphone skin related files are:

- html file: this is a Single Page Application and the HTML code is all in "softphone.html"
- css files are located in \css\ directory
- javascript files are located in: \js\softphone\ directory.

JQuery Mobile

The softphone skin is a single page application built using jQuery Mobile framework.
Note: jQuery Mobile is not maintained anymore by its original developers, however we forked a copy, which we keep up to date and optimized also for latest browser changes.

Every jQuery mobile "page" defined in softphone.html has a corresponding Javascript file in /softphone/ directory.
Regarding css files, styling, most of these css files in "css" directory are related to jQuery Mobile framework which is used to aid in the build of the Single Page application and these should not be edited/changed.
All the styling of the webphone skin is defined in "mainlayout.css" and can be customized from here. This css file should be used to change any styling.

A simple way to change the softphone design can be achieved by changing the jQuery theme: The jQuery mobile Theme Roller generated style sheet can be found in this file: "css\themes\wphone_1.0.css".
Current jQuery mobile version is 1.4.2. Using the Theme roller, you can create new styles: http://themeroller.jquerymobile.com/
The style sheet which overrides the "generated" one (in which all the customizations are defined) is "css/mainlayout.css".

For basic color scheme changes you can use the online designer form. Send us your preferred color parameters and we can send you a softphone build preconfigured with your favorite colors.

Important note: jQuery namespace within the webphone is changed from "$" to "webphone_api.$".

Pages

Pages in softphone.html are <DIV> elements with the following attribute: data-role="page".
Every jQuery mobile "page" defined in softphone.html has a corresponding Javascript file in js/softphone/ directory.
For example settings page's HTML is defined in softphone.html file in a <DIV> element with the id: page_settings. The corresponding Javascript file for the settings page is: \js\softphone_settings.js. All these "page view" Javascript files contain a few lifecycle functions which are used to initiate/load/destroy a page.

Helper containers

For your convenience, the softphone skin already contains a few <DIV> elements where you can drop any custom content.
These are place below the header on each main page (settins, dialpad, contactslist, callhistorylist).
The ID of these elements is the followings:

- extra_header_settings
- extra_header_dialpad
- extra_header_contactslist
- extra_header_callhistorylist

Of course, you can add, modify or remove any user interface element as you wish. Edit the html and CSS files after your needs.
These DIV's were added only to help beginners.

Extra customizable pages

There are 5 predefined extra pages which can be customized as required. These pages are: page_extra1, page_extra2, page_extra3, page_extra4, page_extra5.
The corresponding javascript files are: _extra1.js, _extra2.js, _extra3.js, _extra4.js, _extra5.js.

In each javascript file related to a page, there are three "lifecycle" functions predefined:

- onCreate(): This lifecycle function is called only once per session, when the user navigates to the page for the first time. This is where most initialization should go: attaching event listeners, initializing static page content, etc.
- onStart():This lifecycle function is called every time the user navigates to this page (every time the page is displayed). This is where dynamic content can be added/refreshed. For example on Contacts page we load and display the contacts list.
- onStop():This lifecycle function is called every time the user navigates away from this page. This is where you can save data that the user modified, clear dynamically added page content, etc.

Page navigation:

Page navigation can be done by calling jQuery mobile "changePage()" method:
Format: webphone_api.$.mobile.changePage("#PAGE_ID", { transition: "none", role: "page" });

Example: webphone_api.$.mobile.changePage("#page_extra1", { transition: "none", role: "page" });
To "close" a page, you can either navigate to another page, or return to the previous page by calling jQuery mobile method: webphone_api.$.mobile.back();

Page content:

The content of a page will be put in a <DIV> element with the following attribute: role="main".
Content can be added:

- statically, by defining the content in softphone.html file. For example for page_extra1 the content will be in <DIV> element with id: page_extra1_content.
- dynamically, in function "PopulateData()" defined in all javascript files corresponding to a page.

Autocomplete:

You can turn autocomplete off for your HTML input elements like this:
<input type="text" autocomplete="off" name="destination" id="destination" />

Auto login:

By default the softphone skin will stop at the login screen only if there was no previous login (missing username/password).
Otherwise it should auto-login.

If you wish to preset the username/password then you can do it multiple ways:

- Set it in the webphone_config.js
- Pass as URL query parameters https://www.youraddress.com/webphone_path/softphone.html?wp_username=xxx&wp_password=yyy
- Pass it from JavaScript using the setparameter() API
- Pass it from your server side script session variable

Add functionality to web softphone skin

The webphone is shipped with a ready to use softphone skin with full functionality, but this doesn't mean that other desired functionalities cannot be added.
This can be achieved using the webphone's API.

Empty callback functions can be found at the top of the _dialpad.js file for your convenience (onAppStateChange, onRegStateChange, onCallStateChange, etc).
You can just enter any code for these functions body and/or add more callbacks or functionality from there.

Alternatively you can capture the callbacks (and make any other API functions calls) from a separate js file if you wish, in the following way:

1) create a new javascript file, for example new.js
2) import this script right after "webphone_api.js" in softphone.html file
3) add javascript code to your new.js. Example:

```
//start interacting with the webphone only once the onAppStateChange callback has been fired with the "loaded" event.
webphone_api.onAppStateChange (function (state)
{
```

```
    //console.log('webphone '+state);
    if (state === 'loaded') //webphone loaded
    {
            //webphone loaded. You can begin interacting with the webphone API from here
            //for example you might set any extra parameters at runtime from here
            //example:
            //webphone_api.setparameter('callforwardonbusy', '12345'); //this is just a parameter set example

            //we can also set any custom callbacks here
            webphone_api.onCallStateChange(function (event, direction, peername, peerdisplayname, line, callid)
            {

                    //handle any events here. For example to catch the call connect events:
                    //console.log('Call with '+peername+' '+event);
                    if (event === 'connected')
                    {
                    }
                    //else etc..
            });
    }
});
```

You can find a similar example of webphone API usage in /samples/basic_example.html.

The full webphone API is available so you can easily interact with the softphone skin from external modules or injected code as described in the development and API chapters. Please note that the onCallStateChange and some other callbacks are not used internally by the softphone (because it uses the low level notifications), but you can use them in your code to catch webphone state changes. Just set your callback function(s) and it will work.

GUI architecture

The skin HTML is located in softphone.html.

This is a single page application that uses jQuery Mobile framework for the single page structure and also for UI elements cross browser compatibility. Every "page" in "softphone.html" has a corresponding Javasript file in "\js\softphone\" directory, for example: `"page_dialpad"` is managed by "_dialpad.js" file. Each Javascript file managing a page has three "life cycle" callback functions:

- onCreate() - called only once – it is a good practice to handle binding to all kind of events here
- onStart() – called every time the page is displayed
- onStop() – called every time the page is "closed"

Note: jQuery namespace within the webphone is changed from "$" to "j$" or to "webphone_api.$".

Page list

Below is a list of all Javascript files managing pages. These are located in \js\softphone\ directory:
"_accounts.js": manage multiple SIP accounts
"_addeditcontact.js": user interface for creating or editing contacts
"_call.js": user interface of the in call page
"_callhistorydetails.js": call history details page
"_callhistorylist.js": call history list page
"_message.js": messaging page
"_messagelist.js": messaging inbox list page
"_contactdetails.js": contact details page
"_contactslist.js": contacts list page
"_filetransfer.js": file transfer page
"_filters.js": simple prefix rewrite page
"_internalbrowser.js": web view used as a browser
"_logview.js": page for viewing internal app log/trace
"_newuser.js": new user registration page
"_dialpad.js": main dial pad page
"_settings.js": settings/login page
"_startpage.js": it's a page displayed once or on every start; can by a URL or text; usually used for privacy policy
"themes.js": contains all the definitions of the different color themes that can be applied to the skin

Menu

Every page has a menu which can be accessed when the menu button in top-right corner is clicked. This click event triggers the CreateOptionsMenu() function in the corresponding page javascript file. In CreateOptionsMenu() function the menu is assembled every time it is displayed to the user.
Then in MenuItemSelected() function the selected menu item will be handled.

Navigation

Navigation between pages can be accomplished with the jquery mobile changePage() method. For example, to navigate to settings page the following command will be used:

```
webphone_api.$.mobile.changePage("#page_settings", { transition: "pop", role: "page" });
```

Styling

All the skin related CSS files are located in "\css\softphone\" directory. Here all the CSS files are jQuery mobile generated style sheets, except "mainlayout.css". All the styling is done here.
All element dimensions are set proportionally with percentages if possible, or using "em" unit of measure, to assure that the skin user interface resizes well on any screen size and any resolution.

Skin start up

The main page launched is every time the settings page. If the user has already entered login credentials before, the skin will automatically login and display the dial pad page.

Common tasks

The softphone skin is open sourced (plain html/css/js source code is included in the webphone package) and it can be freely changed by any web developer.
A few common tasks are described below for your convenience:

How to add a new page

The ready to use skin which is delivered with the webphone package is built as a single page application using jQuery mobile framework (actually only a few things are used from jQuery mobile, the rest is plain html/javascript).
Follow the steps below to create a new page:

1) Create new html element with attribute **data-role="page"** in softphone.html file. You can use "page_extra1" as template.
2) Create the equivalent JavaScript file, which manages the behavior of the new page. You can use the "_extra1.js" file as template.
3) Add the three "life cycle" callback Javascript functions in init.js
    a. webphone_api.$(document).on('pageinit
    b. webphone_api.$(document).on('pageshow'
    c. webphone_api.$(document).on('pagebeforehide'

The new page is ready.

How to change the login page

The login page is managed in "_settings.js" file. Basically, it displays only the main login settings: serveraddress, username, password. There is also a traditional style login page which can be enabled with the "useloginpage" parameter: -1=auto (default), 0=no, 1=only at first login, 2=always.

How to change the main page (dial pad tab)

The main dial pad tab consists of the "page_dialpad" id defined in softphone.html and the associated javascript file: _dialpad.js.

How to add a new setting

First establish the setting parameter name, which should be in lowercase, for example "newsetting".
Open stringres.js file and define the new setting title, description and short description (short description is optional) with the following format:

```
sett_display_name_mysetting: 'My settings',
sett_comment_mysetting: 'My new settings does the following',
sett_comment_short_mysetting: '',
```

If you want this setting to appear on the user interface, then add the "newsetting" to the comma separated list stored in String variable "settOrderWebphone" and "settOrderWebphoneWebRTCFlash" located in _setings.js file. Also, the order in this list will define the order in which the settings will be displayed. For a setting to appear in "Basic settings", the parameter name should be added to the slit in variable "basicSettings".
To actually define a new setting, go to "InitializeSettings()" function in common.js file add the new settings using SettingItem() function.
The following line of code will add the new settings:

```
SettingItem ("newsetting", "", "1", GROUP_LOGIN, "", "", "1", "");
```

SettingItem function has the following parameters:

- (String) parameterName: the name of the parameter.
- (String) value: the current value of the parameter. I can be empty string.
- (String) type: the type of the settings:
  - "0" = (boolean) enabled/disabled
  - "1" = text input
  - "2" = dropdown list
- (String) group: it belongs to this group/submenu:
  - "20" = displayed on login page
  - "0" = basic settings list
  - "1" = SIP settings
  - "2" = Media settings
  - "3" = Call divert settings
  - "4" = General Settings
  - "8" = Video settings
- (String) allNames: a comma separated list of the display names of possible values. Used only for dropdown list type settings, otherwise empty string.
- (String) allValues: a comma separated list of the actual parameter values. Used only for dropdown list type settings, otherwise empty string.
- (String) isdefault: the value here should always be "1"
- (String) defaultValue: the default value of the parameter

How to remove a setting

Go to "InitializeSettings()" function in common.js file and comment out/delete the "SettingItem()" function call for that parameter that is to be removed. Also search for setting parameter name in "settOrderWebphone" and "settOrderWebphoneWebRTCFlash" and "basicSettings" variable and remove it from there if found.

How to add a new menu item

In every javascript file that manages a page (located in \js\softphone\), the Menu is created in "CreateOptionsMenu ()" method. Add a menu with the following line of code:

```
webphone_api.$(menuId).append('<li id="ITEM_ID"><a data-rel="back">TITLE</a></li>').listview('refresh');
```

In MenuItemSelected() function handle the action for the added menu based on itemId.

How to remove a menu item

In every javascript file that manages a page (located in \js\softphone\), the Menu is created in "CreateOptionsMenu ()" method. Just comment out or delete the "webphone_api.$`(menuId).append()`" line for the item to be removed.

How to add a header/footer

For every page there is already added a header and a footer where custom content can easily be displayed.
The content to be displayed here can be text or HTML and can be specified with the following parameters in webphone_api.js: "header" and "footer".

Click to call

Click-to-call (also known as click-to-talk, webcall or click-to-dial) is a simplified form to make a call when the enduser just needs to click on a button or a hyperlink to request an immediate connection to another webphone, softphone or PSTN (landline/mobile), without the need of configuration to be set by the enduser.
You can use the webphone SIP library to implement your custom click-to-call solution or use one of the skin templates for click to call. Click to call buttons or links can be embedded into any webpage (sales page, blogs, social media, others) and also in email (click to call from email signature).
You can also set auto-dial feature when the call is initiated automatically without enduser interaction, using the callto and autoaction parameters.

There are multiple ways to achieve click to call or clicktodial functionality with the sip webphone:

1) Click to call from HTML link via URL parameters (single line copy-paste code into your html)
2) Use the webphone library directly (via it's API) to implement any custom click to call solution (for JavaScript developers)
3) Use the click2call template as-is. A click to call skin is shipped with the webphone (click2call.html and related css/js files) as a turn-key click to call solution which can be used without any development knowledge
4) Modify the click2call template by changing its settings and skin after your needs)

Details:

1) Click to call from HTML link

You can pass any setting as URL parameter and the webphone (and the included templates) can be easily parametrized to act as a click to call solution:
http://www.yourwebsite.com/webphonedir/click2call.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_callto=CALLEDNUMBER&wp_autoaction=1

A working example with the click to call skin:
https://www.webvoipphone.com/webphone_online_demo/click2call.html?wp_serveraddress=voip.mizu-voip.com&wp_username=webphonetest1&wp_password=webphonetest1&wp_callto=testivr3&wp_autoaction=1

This will launch the click to call page and will initiate the call automatically. You can find more examples for URL parameters here.

You can also use any other skins for click to call. For example here is with the softphone skin:
https://www.webvoipphone.com/webphone_online_demo/softphone.html?wp_serveraddress=voip.mizu-voip.com&wp_username=webphonetest1&wp_password=webphonetest1&wp_callto=testivr3&wp_autoaction=1

2) Use the webphone library directly

You can easily create your custom click to call solution from scratch by using the webphone as a library/SDK.
Just create a HTML button after your needs and call the webphone_api.call(number) function from there.
This is recommended for JavaScript developers and in this way you can create any custom web click to call solution after your needs.
Have a look at the click2call_example.html in the samples folder for a very basic click to call implementation (you can use this as a starting point for your custom development).
See the Development and the API sections about how to work with the web phone SDK.

3) Use the Click2Call template as-is

The webphone package contains a ready to use click to call solution (click2call.html and related JS/CSS files).
Just copy the whole webphone folder to your website, set the parameters in the webphone_config.js file and use the click2call.html as-is on your webpage or directly.

Configure it after your needs by adjusting (changing and/or adding) the settings in the “Configuration parameters” section in the “click2call.html” file. See the parameters settings for the full list of adjustable settings.

Note: You can quickly test this also from the online demo if you haven’t downloaded the webphone demo package yet:
https://www.webvoipphone.com/webphone_online_demo/click2call.html

4) Modify the Click2Call template

You can completely customize the click2call example for your needs (change any settings, change the html/css/javascript, use your custom button image).

This click2call template/skin consists of a few different files:

- \click2call.html
- \css\click2call\click2call.css
- \js\click2call\click2call.js

In the HTML file there is short html code which defines a click to call button:
This is the single line of HTML code which you will include in your webpage where you want the click to call button to be displayed. For the call button to work, you will also have to include (preferably in the <head> section of your web page) the related css (in this case click2call.css), Javascript (click2call.js) and the webphone API file: webphone_api.js.
The Configuration parameters in click2call.html are put there just for convenience. These same parameters can be just as easily set in webphone_config.js file where the other webphone parameters are set.
In order for the click to call skin to work, a valid SIP account and a destination number need to be configured with the following API parameters :

- serveraddress: yoursipdomain.com your VoIP server IP address or domain name
- username: SIP account username
- password: SIP account password
- callto: destination number to call (the click2call can be also modified to load this at run-time if there are multiple target numbers)

(You can also fine-tune your solution by setting/changing any of the webphone parameters)

The click-to-call buttons color, width, height, radius or the displayed text can be customized after your needs from the click2call.js file using the variables from the "Customizations" section.
You can even add a background image, by setting your image (jpg or png format) path for the call_button_color variable in the following way:
var call_button_color = 'url(IMAGE_PATH/YOUR_PHOTO.png)';

Integrate into web page

Below is an example code and steps to integrate the click to call button into your web page:

Put the below code in your web page's <head> section:
<link rel="stylesheet" href="css/click2call/click2call.css" />

```
<script src="webphone_api.js"></script>
<script src="js/click2call/click2call.js"></script>
<script>
    /**Configuration parameters*/
    webphone_api.parameters['serveraddress'] =  '';
    webphone_api.parameters['username'] = '';
    webphone_api.parameters['password'] = '';
    webphone_api.parameters['md5'] =  '';
    webphone_api.parameters['realm'] = '';
    webphone_api.parameters['callto'] = '';
    webphone_api.parameters['autoaction'] = 1;
    //Add any other parameter here

</script>
```

Copy this html element in you page, where you want the click to call button to show up:

```
<div id="c2k_container_0" title=""><a href="tel://CALLTO" id="c2k_alternative_url">CALLTO</a></div>
```

Customize the button

The provided click to call example button can further be customized with customization parameters located at the beginning of click2call.js file in js\click2call\ directory:

```
/** Customizations */
var call_button_text = 'Call';     // text displayed on call button
var hangup_button_text = 'Hangup';     // text displayed on hangup button
var call_button_color = '#43b61b';     // call button color
var hangup_button_color = '#e83232';    // hangup button color
var status_text_color = '#ffffff';     // color of displayed status messages
var button_width = 135;              // button width in pixels
var button_height = 42;              // button height in pixels
var button_radius = 5;              // button corner radius in pixels, higher values will result in a round button
var chatwindow_title = 'Chat';         // chat window header-title; can be text or html
var chatwindow_default_state = 0;      // default state of the chat window: 0=open, 1=collapsed
```

The styling can be further customized from click2call.css located in css/click2call/ directory.

```
/**For floating button*/
var float_button = false;            // if set to true the button will float over the content of the webpage
var float_distance_from_top = -1;      // distance in pixels from the top of the page. -1 means disabled
var float_distance_from_left = -1;     // distance in pixels from the left of the page. -1 means disabled
var float_distance_from_bottom = -1;    // distance in pixels from the bottom of the page. -1 means disabled
var float_distance_from_right = -1;     // distance in pixels from the right of the page. -1 means disabled
```

Customize the behavior

The click to call example can further be customized by editing the js\click2call\click2call.js Javascript file.
All the API functions found in webphone_api.js can be used to achieve the desired functionality.
For example if you choose to add your own custom incoming call popup, you can do this in click2call.js file using the webphone_api.onCallStateChange() function where you receive call events. For chat message events use the webphone_api.onChat() function. Find more details about the API functions in the documentation and in webphone_api.js.

Use as a chat window

The click to call button can also be used as a chat window. This is controlled by the autoaction parameter (1 means call, 2 means chat).
The chat window can also be opened by accessing the menu and selecting the Chat item.
The menu can be accessed by right clicking or by long clicking on the button.

Floating button

This click to call button can also be used as a floating button on a web page. The floating related configurations can be found in click2call.js file located in js/click2call/ folder.
To enable floating, set the "float_button" config to true and specify two direction coordinates for the floating. For example to have a floating button on the top right corner of your page, located from 100 pixels from the top and 10 pixels from the right:

```
var float_button = true;
var float_distance_from_top = 100; // floating 100 pixels from the top of the page
var float_distance_from_left = -1;
var float_distance_from_bottom = -1;
var float_distance_from_right = 10; // floating 10 pixels from the right of the page
```

The floating parameters also apply to the chat window in the same way as for the click to call button.

Note: If somehow your changes will not have effect in the web browser, you just need to clear the cache.

Multiple instances
To add more than one click to call button to a page, include the script part in the <head> section once, and copy the container increasing the id index number for every instance.
ex:

```
<div id="c2k_container_0" title="55555"></div>
<div id="c2k_container_1" title="66666"></div>
<div id="c2k_container_2" title="77777"></div>
<div id="c2k_container_3" title="88888"></div>
```

These id indexes must be unique and increasing.
The callto parameter can be set as the title attribute of the element.

Load on demand
You can also load the sip web phone on demand as explained here.

Auto-call
If you wish to make a call automatically, then just initialize the webphone as described above and
-either set also the "autoaction" parameter to 1
-or make the call with the webphone_api.call(number) API from the onAppStateChange() callback "loaded" event or from the onRegStateChange () callback "registered" event.

The autoaction API parameter controls the behavior of the click to call button and can have the following values:
0: Nothing (default) - it will display the click to call button and will place a call whet the user clicks it.
1: Call - it will immediately place a call, once page is loaded in which the click to call button is integrated.
2: Chat - a chat window will be displayed for the user
3: Video Call - it will immediately place a video call, once page is loaded in which the click to call button is integrated.

Note:
- *Even if you initiate a call form the onAppStateChange "loaded" event and the webphone is not yet registered -and it needs to register-, then it will handle registration first then it will initialize the call automatically.*
- *If your IP-PBX doesn't require registrations, then just set the "register" setting to 0.*

More details about the click to call functionality can be found here and in this wiki article.

Custom solutions

The most common use case for the webphone is a dialer (softphone user interface) or as a click to call solution (simple click to call button user interface).
For these we got you covered with ready to use solutions shipped with the webphone (see the softphone.html and the click2call.html files).
However this doesn't mean that the webphone usage is restricted only for these. You can create any kind of solution using the webphone. A few examples are listed here.
All you need is a little HTML knowledge to create your desired user interface and some JavaScript knowledge to use the webphone API from your HTML/JS.
You might implement your solution by extending the softphone skin or by creating a new solution from scratch.
See the webphone development section for more details.

Integration with Web server side applications or scripts

This section is mostly for server side developers. If you have more JavaScript skills then we recommend to just use the JavaScript API directly as described in the development section.

First of all it is important to mention that the webphone doesn't have any server side framework dependencies. You can host it on any webserver without any framework (.PHP, .NET, Node.Js ot others installed).

The webphone is running entirely on the client side (in the user browser as a browser sip plugin) and can be easily manipulated via its JavaScript SIP API, however you can easily integrate the webphone with any server side application (web applications or web service) or script, be it .NET, PHP, Node.Js, J2EE or any other language or framework even if you don't have JavaScript experience. Just create a HTTP API to catch events such as login/call start/disconnect and drive your app logic accordingly.

The most basic things you can do is to **dynamically generate the webphone parameters** per session depending on your needs. For example if the user is already logged-in, then you can pass its SIP username/password for the webphone (possibly encoded).
For this, just generate the webphone_config.js dynamically or pass the parameters in the URI.

For a tighter integration you will just have to **call into your server from the webphone**.
This can be done with simple XMLHTTP /AJAX or websocket requests against your server HTTP API, then process the events in your server code according to your needs. The requests can be generated using the built-in HTTP API events or you can just post them yourself from your custom JavaScript code using websocket or ajax requests. Usually these requests will be made from callback events which are triggered on web phone state machine changes, but you are free to place ajax request anywhere in your code such as click on a button.

Example:
For example if you need to save each call details (date, caller, called, duration, others) into a server side database, then just define a "oncalldetails" or similarly named API in your server side application which can be called via simple HTTP request in one of the following ways:

1. Using the built-in HTTP API integration capabilities:
   Just set the scurl_onincalldisconnected to your HTTP API. For example: https://mydomain.com/myapi/oncalldetails/ (or wherever your API can be called). This method is very convenient if you are a server side developer with no JavaScript knowledge as you don't need to touch any JavaScript to implement this. See the details here.

2. Using custom AJAX requests:
   Use the onCdr() API to setup a callback which will be triggered after each call.
   Send an AJAX request (XMLHttpRequest or jQuery get or post) to your application server with the CDR details.
   (You can pass the details in HTTP GET URL parameters or in HTTP POST body in your preferred format such as clear text, json, xml or other).

You will need an API on your web server to handle the websocket or AJAX HTTP GET/POST requests from the webphone.
This can be done by implementing a web service or a simple script in any server side framework on language (such as .ASP .NET, PHP, Node.js or others).
You can push/request anything from the webphone and do anything with the data (store in a database such as MS SQL, MySQL or PostgreSQL) or answer with some content to be displayed on the webphone user interface.
For example on incoming calls (cached with the onCallStateChange callback or preset with the scurl_onincallsetup parameter) you might look-up the caller id (caller number) in your database and return details about the caller (such as full name, interests, etc).
You can do similar things for all other important events such as on phone start, on chat, on call setup, etc and perform various actions on your server side such as processing/transforming/storing the data received by the webphone and/or returning answers to be processed or displayed by the webphone.

Note: For auto-provisioning from a server side application, you can create an API to return all the webphone parameters (settings) and set the "scurl_setparameters" setting to this API URL.

Custom HTTP API integration

You can integrate the webphone with your server code using your custom HTTP requests (AJAX/fetch) API URI's. These are called "Action URL's", so you can implement various server-side actions when something happens on the SIP client (such as on incoming call, on call finished, and others). You can implement these API's in your existing server-side framework, using any language or tools (ASP.NET, PHP or any others).
Just set one or more of the following settings to point to your server application HTTP API entries which will be called automatically as the webphone state machine changes:

- **scurl_onstart**: will be called when the webphone is starting
- **scurl_onoutcallsetup**: will be called on outgoing call init
- **scurl_onoutcallringing**: will be called on outgoing call ring
- **scurl_onoutcallconnected**: will be called on outgoing call connect
- **scurl_onoutcalldisconnected**: will be called on outgoing call disconnect with call details (CDR)
- **scurl_onincallsetup**: will be called on incoming call
- **scurl_onincallringing**: will be called on incoming call ring
- **scurl_onincallconnected**: will be called on incoming call connect
- **scurl_onincalldisconnected**: will be called on incoming call disconnect with call details (CDR)
- **scurl_oninchat**: will be called on incoming instant message
- **scurl_onoutchat**: will be called on outgoing instant message
- **scurl_setparameters**: will be called after onAppStateChange started event and can be used to provision the webphone from server API. The answer should contain parameters as key/value pairs, ex: username=xxx,password=yyy
- **scurl_displaypeerdetails**: will be called at the beginning of incoming and outgoing calls to return details about the peer from your server API (like full name, address or other details from your CRM). It will be displayed at the location specified by the "displaypeerdetails" parameter. You can return any string as clear text or html which can be displayed as-is.

Special keywords will be replaced by the webphone at runtime. The keywords are listed here.
For example: scurl_onoutcallsetup: https://mydomain.com/myapi/?user=USERNAME&called=CALLEDUMBER
(Your API will be called each time the webphone user makes an outgoing call and the parameters in uppercase will be replaced at runtime in the same way as described for the links setting)

For API request, the webphone will try to use following techniques (first available): AJAX/XHTTP, CORS, JSONP and websocket (if available).
Make sure that there is no any cross-domain request restriction: you might need to set the Access-Control-Allow-Origin HTTP header on your WEB server/service to * or to the target domain if you are not hosting the webphone on the same domain.

You can achieve similar functionality by just sending AJAX request from the onCallStateChange callback.
Using these scurl variables is just a different approach to help users with little JavaScript knowledge.

Integration with third party systems and CRM's

Many custom callcenters are using the webphone as their VoIP client module to implement voice/video call capabilities.

You can use the webphone SIP library to implement your custom click-to-call solution, use one of the skin templates for click to call or use the softphone.html as-is integrated into your CRM.

The VoIP web sip phone browser plugin doesn't depend on any framework and can be integrated with any system or CRM with JavaScript binding support. Usually you just need to include the webphone_api.js, set the basic parameters in the webphone_config.js (such as serveraddress/username/password to be used, but these can be passed also by the API) and just use the call() function to make an outgoing calls. Incoming calls are handled automatically and with a few more API calls you can easily implement features such as call transfer, conference, chat, dtmf or video call.

For example here is tutorial for Salesforce webphone integration in case if you are interested in this platform or some details about VoIP callcenter integration.

Consult your CRM documentation to find the details about integration third-party general modules (or even better if it has an interface specific for third party phone integrations). Contact us if you need help with any integration.

Using the webphone with various server side frameworks

The webphone doesn't require any server side web framework however you are free to use your own favorite framework, language and libraries to interact with the web phone such as PHP, ASP.NET, J2EE, NodeJS, Perl, C++, Python, Ruby on Rails, Express.js, Django or any other framework.

This is useful if you have a server application and you need certain details from the webphone (such as call detail records to be inserted into your database) or you wish to push some data to the webphone (settings depending on the enduser, incoming caller details, etc).

By default the webphone has nothing to do with server side framework (it is a pure client-side library) so you can deploy it like any other javascript library.

Then you can add server side binding in various ways:

- if you don't have any JavaScript experience, just use the action URL's mentioned above
- otherwise you can interact with your server side service with AJAX calls (for example you can notify your server about phone state change by triggering an xhttp request from the from the onCallStateChange callback) or send call the details after hangup from the onCdr callback.
- you can use also any other technique for the communication between the webphone and your server such as WebSocket, HTTP polling or use a library such as Socket.IO technique (These might be necessary only if you wish to exchange a lot's of data between the webphone and your server application. Otherwise simple HTTP API requests are just fine)

In short: All you need to do is to create some API's in your server application to be called from the webphone, usually via AJAX XHTTP requests.

Examples:

We don't have too much framework specific examples to show as the webphone is a client-side library and the server side integration is just optional and the usage is really straightforward as described above:

- ASP.NET: a very basic example from one of our customer can be downloaded from here
- A basic demonstration/example about using the webphone with PHP can be found here.
- Integration with various frameworks can be also done via WebView as described here.

Integration with SIP server side applications or scripts

This is a more straightforward topic then the previous Web application integration because here we are restricted to the SIP protocol capabilities.

Actually the main purpose of the whole webphone library is to interact with a SIP server.

For a tighter integration you can take full advantage of the SIP protocol using the following tools:

- Your SIP server capabilities (configure the webphone extensions like any other SIP endpoint, taking full advantage of your softswitch (or proxy/gateway/other SIP device) capabilities such as voicemail, server side call forward, call queue, etc
- Webphone parameters: you can fully customize the webphone behavior by changing the parameters
- Webphone API: fully control the interaction between the webphone and your SIP server using the webphone API. Beyond basic functionality, you can find useful functions such as dtmf or presence which can be used for specific interaction with your SIP server in a standard way. For example, during a call you can exchange additional details with your SIP server using SIP INFO, DTMF or even chat/presence notifications.
- You can also consider using a side-channel to exchange information with your server using alternative methods such as HTTP API or websocket. For this you will need a server side application and the details described in the previous chapter can be applied.

In case if you have some built-in application or script then a preferred method to exchange information between the server and the webphone is the usage of **extra SIP headers** (which can be easily sent/extract on your SIP server and also by the webphone). For more details see the customsipheader parameter and the setsipheader/getsipheader API's.

Integration with the SIP server API is also possible by using the links parameters. This is useful for things like balance display, server side contact list, SMS, recharge and others.

Development

This section is for JavaScript developers. You can use this webphone also without any JavaScript skills:

- *If you don't have any programming skills: customize and use the included turn-key templates (for example the softphone.html or click2call.html) on your website.*
- *If you are a server-side developer not comfortable with JS: take advantage of the server integration capabilities*

Developers can use the webphone as an SDK (JavaScript SIP library) to create any custom VoIP solution, standalone or integrated in any webpage or web application.

Setup

First of all you should download and deploy the webphone on your webserver (copy the webphone folder to your web server).
You can also launch it from local file system on your dev environment (works with some limitations), but the best is if you use it from a secure (HTTPS) webserver.

Notes:
Requirements:
Basic HTML and JavaScript knowledge is required to work with the webphone to create custom solutions.
You will also need a web server and a SIP account. See more details here.

Settings:
Once you deployed the webphone, you will have to adjust the settings. At least the "serveraddress" parameter must be set.
The library parameters can be preconfigured in webphone_config.js, changed runtime from JavaScript, passed by URL parameters or set dynamically by any server side script such as PHP, .NET, java servlet, J2EE or Node.js.

Frameworks:
The webphone doesn't require any extra client or server side framework (it is a client side VoIP implementation which can be used from simple JavaScript) however you are free to use your own favorite framework or libraries to interact with the web phone (for example use with jQuery on the client side or integrate into your PHP/.ASP/.NET/J2EE/NodeJS or other server side framework or use it straight without any frameworks involved).

Demo:
The downloadable demo version has some limitations to disable commercial usage, however if your development/test process is affected by these then you can request a trial from mizutech with all demo limitation removed.

API

The public JavaScript API can be found in "webphone_api.js" file, under global javascript namespace "webphone_api".
Just include the "webphone_api.js" to your project or html and start using the webphone API.

The API reference can be found here.
Make sure to call any API function only after the webphone completely initializes itself (the onAppStateChange callback was fired with the "loaded" event).

Many functionalities can be controlled by both the parameters and by the API. When you need some persistent global behavior then you might use the parameters, otherwise use the API if you need more control (for example toggle a function at runtime or apply it only in some circumstances such as for a single call only or manage multiple simultaneous calls differently). For example you can use the voicerecupload parameter if you need all calls to be recorded or use the voicerecord() API to record only some calls (or part of the calls). Another example is the customsipheader parameter vs the setsipheader() API.

Work-flow

Follow these steps to get started:

1 Download and deploy the webphone
3. Include the webphone to your project by adding/importing the webphone_api.js to your project
   For HTML add the following line: <script src="webphone_api.js"></script>
4. Create any user interface in HTML/CSS (or any other framework capable to bind to JavaScript) if you need something from scratch.
   Otherwise you can use, modify or start with one of the html included in the webphone package: the softphone.html, the click2call.html or the examples in the samples folder.
5. Configure the webphone via its parameters. This can be done in multiple ways (but it is enough if you use one method only):
   a. by using webphone_api.setparameter() API from JavaScript once the webphone is loaded
   b. by setting static parameters in the webphone_config.js file
   c. other methods described here
6. Use the webphone API from your user interface controls (such as webphone_api.register () from your CONNECT button or webphone_api.call () from your CALL button)
7. Handle the webphone state machine (call state) by using the onCallStateChange and other callbacks (modify your user interface based on the webphone state, such as displaying "Connected" when the webphone is registered or displaying a "Incoming call from X. Accept/Reject" message on incoming calls)
8. Consult the parameters, the API, the examples in the samples folder and other parts of this documentation for more details

Usage

There are 3 basic things which you have to do while working with the SIP SDK:

1. Set parameters: There is a long list of parameters that you can set to configure the webphone after your needs (but only a few really important).These can be set statically in the webphone_config.js files, dynamically from your code using the setparameter API or with other methods as described at the beginning of the parameters chapter.
2. Call API functions: You can interact with the webphone by calling its API. For example you can initiate a call with the call function.

3. Process the events: Handle the callback notifications according to your application logic. The most important is the onCallStateChange callback from where can update your user interface from there as required.

Note:
You can use the webphone also without point 2 and 3, bye just configuring a html shipped with the webphone (for example the softphone.html).
Also even if you use point 2 (API), for very simple apps you might not need to watch for events at all (For example to make blind calls to a destination).

Simple example

A minimal implementation (a simple VoIP call) can be achieved with less than 5 lines of code on your website. See the minimal_example.html (found in the webphone package samples folder).
Example:

```
<head>
        <!-- Include the webphone_api.js to your webpage -->
        <script src="webphone_api.js"></script>
</head>
<body>
<script>
        //Wait until the webphone is loaded, before calling any API functions
        webphone_api.onAppStateChange(function (state)
        {
            if (state === 'loaded')
            {
                //Set parameters (Replace upper case worlds with your settings)
                //Alternatively these can be also preset in your webphone_config.js file or passed as URL parameters
                webphone_api.setparameter('serveraddress', SERVERADDRESS);
                webphone_api.setparameter('username', USERNAME);
                webphone_api.setparameter('password', PASSWORD);
                //See the "Parameters" section below for more options

                //Start the webphone (optional but recommended)
                webphone_api.start();

                //These API calls below actually should be placed behind separate functions (button clicks)
                //Make a call (Usually initiated by user action, such as click on a click to call button. Number can be extension, SIP username, SIP URI or mobile/landline phone)
                webphone_api.call(NUMBER);

                //Hang-up (usually called from "disconnect" button click)
                webphone_api.hangup();

                //Send instant message (Number can be extension, SIP username. Usually called from a "send chat" button)
                webphone_api.sendchat(NUMBER, MESSAGETEXT);
            }
        });
        //You should also handle events from the webphone and change your GUI accordingly (onXXX callbacks)
</script>
</body>
```

Other examples

See the html files in the webphone/samples folder for more examples.

- a very simple but functional basic example can be found in the webphone package: basic_example.html
- as a better example, see the tech demo page (techdemo_example.html / techdemo_example.js) or the api_example.html
- click2call.html is a ready to use click to call implementation
- softphone.html implements a fully features browser softphone (this can be found in the webphone root folder, not in the samples folder). You might choose to extend this skin instead of creating your own from scratch (in case if your goal is to create a complex dialer based solution).

You can also try the same examples from our online demo.

You are free to use/modify any of these files and adjust it after your needs or create your own solution from scratch.
For beginners we would recommend to start with the "basic_example.html ", "api_example.html " or "techdemo_example" and modify/improve it after your needs.
If you need something ready to use, than just use the "softphone.html" (this can be also customized with the many webphone parameters or by changing the html/css/js files)

Advanced functions

Most of the traditional VoIP functionalities (in/our calls, chat, call divert) can be handled very easily with the webphone, however some advanced features might require special care if you wish to interact with them.

Lots of things can be achieved by the webphone parameters, without the need of any programming effort.
Here are some examples for advanced usage:

- settings/auto-provisioning: it can be done easily with the setparameter API but you might have special needs which would require to pass the parameters in a special way. See the beginning of the parameters section for the possibilities and some more in the FAQ.
- multiple lines: handled automatically but you might handle it explicitly if required for your project
- low-level engine messages: this is only for advanced users and rarely needed to intercept these messages. You might use the onEvent callback for this but it is recommended to use the other high level events such as the onCallStateChange to handle the web SIP state machine
- low-level interaction with the native engines: if somehow you have some extra requirements which is not available with this high-level API then you might use the low-level jvoip API with the NS and Java engines
- dtmf, call transfer, hold, forward: we took special care to make these as simple to use as possible so all of these can be handled by a single API call
- voice call recording: just set the voicerecupload parameter or use the voicerecord API from Java Script
- conference: handled automatically by default via a single API call but optionally you might implement some specific user interface to display all parties
- parameter encryption/obfuscation: usually not required since you are working with them in the otherwise secure user session, but if you wish to use it then it is described here
- video: you must provide a html element where the video have to be displayed and manage this GUI accordingly:
- chat, sms: these also requires some extra user interface to send the messages and display the call history
- manipulating SIP messages: requires some VoIP/SIP skills if you need to interact this way with your VoIP server and you can use the setsipheader/getsipheader API's
- create a native application by using the webphone from a WebView

Note: all of these are also implemented in the "softphone" skin which is included with the webphone so you might use/modify this skin if you need a complete softphone like solution instead to develop your own from scratch (if you don't have specific requirements which can't be handled by customizing the softphone skin)

For more details regarding custom development, see the "JavaScript API" section below in this documentation.

Parameters

The webphone parameters can be used to fully customize the webphone behavior including SIP settings, media settings, user interface settings, for example parameters like the SIP server domain, authentication, called party number, autodial and many others.

Most of the settings are optional except the "**serveraddress**" (but also this can be provided at runtime via the API).
The other important parameters are the SIP user credentials (**username**, **password**) and the called number (**callto**) which you can also preset (for example if you wish to implement click to call) however these are usually entered by user (and optionally can be saved in local cookie for later reuse).

The webphone parameters can be set in multiple ways (statically and dynamically) to allow maximum flexibility and ease the usage for any work-flow.
Use one (or more) of the following methods for the webphone configuration:

- Preset the settings in the "**webphone_config.js**" file. This should be used for parameters which are expected to not change at runtime, such as the serveraddress parameter.
- Use the **setparameter()** API call from JavaScript. For most parameters the setparameter function must be called from the onAppStateChange callback "loaded" event. Never sooner (API functions, including the setparameter, works only after the "loaded" event was triggered in onAppStateChange).
- Changing the parameters by using the webphone_api.parameters['paramname'] = 'value' format is NOT recommended because parameters changed in this way can't be overwritten later by the setparameter() API.
- Webpage **URL** query string (The webphone will look at the embedding document URL at startup. Prefix all keys with "wp_". For example &wp_username=x or any other parameter specified in this documentation)
- Via the scurl_setparameters settings which can load the parameters from your server side application (This will be called after the onAppStateChange "started" event and can be used to provision the webphone from server API. The answer should contain parameters as key/value pairs, ex: username=xxx,password=yyy)
- Cookies (prefix all keys with "wp_". For example wp_username)
- Web session variables (PHP example)
- SIP signaling (sent from server) with the x-mparam header (or x-mparamp if need to persist). Example: x-mparam=loglevel=5;aec=0
- Auto-provisioning: the browser phone is also capable to download it's settings from a config file based on user entered OP CODE (although this way of configuration is a little bit redundant for a web app, since you can easily create different versions of the app –for example by deploying it in different folders- already preconfigured for your customers, providing a direct link to the desired version instead of asking the users to enter an additional OPCODE)
- User input: You can let the user to modify the settings. For example to enter username/password for SIP authentication (For example using the softphone skin most of the settings can be specified by the users at login/settings screens, which might overwrite server side settings loaded from the webphone_config.js file if any)
- Also see here and here

Any of these methods can be used or they can be even mixed.

A quick and easy way to start is to just set all the required parameters in the webphone_config.js file. For example:

```
webphone_api.parameters = {
        serveraddress: 'voip.mizu-voip.com', //your SIP server domain:port or IP:port
        username: 'webphonetest1',  //the username is usually specified by the enduser and not need to be set globally here
        password: 'webphonetest1',  //the password is usually specified by the enduser and not need to be set globally here
        displayname: 'John Smith', //optional display name, usually specified by the enduser and not need to be set globally here
        brandname: 'BestPhone', //your brand name
        rejectonbusy: true, //will reject incoming call if user already in call
        ringtimeout: 50000, //disconnect the call after 50 sec on no answer
        loglevel: 5 //enable detailed logs
        //no comma is needed after the last parameter
};
```

Usually you might set some parameters in the above webphone_config.js file (the common static/global parameters applicable for all users which doesn't need to be changes such as the serveraddress or proxy address), then you might use one of the other methods to specify instance specific parameters (for example ask the user credentials at login screen and set it as the username/password parameters with the setparameter API).

Note:

- For a basic usage you will have to set only your VoIP server ip or domain name ("serveraddress" parameter).
  The SIP username/password are asked from the user with the default skins if not preconfigured.
  The rest of the parameters are optional and should be changed only if you have a good reason for it.
- Some parameters (username/password, displayname) are usually set by the user via some user interface (using the setparameter() API), however in some situation you might hardcode them on the server side webphone_config.js file. For example if you have some static IVR service and the caller user identity doesn't matter.
- All parameters can be passed as strings and will be converted to the proper type internally by the webphone browser plugin.
- Most parameters are saved by the webphone (in web storage and cookies) and reused at next sessions (but you can always overwrite already stored settings or clear them by setting the value to 'NULL' or use different profiles).
- Don't remove or comment out already set parameters because the old value might be already cached by the browser webphone. Instead of this you should just set to 'NULL'/ 'DEF' or its default value. Details here.
- If you have changed the parameters in the webphone_config.js then you might change its jscodeversion parameter where you include it in your project, to avoid any caching and force a re-download by the browser. For example: <script src="webphone_config.js?jscodeversion=1234"></script>
- Prefix parameter name with "ucfg_" if it should prefer client side settings (otherwise server side settings defined in the webphone_config.js will overwrite the client settings). Example: ucfg_aec: 2
- You can also clear the settings with the delsettings API or force the webphone to forget old settings with the resetsettings parameter. Details here.
- Parameters can be also encrypted or obfuscated. See the "Parameter security" section for the details.

Important settings

The most important parameters are the followings:

- serveraddress (your SIP server domain or IP:port, usually set statically in the webphone_config.js)
- username/password (SIP account details, usually entered by the endusers and passed to the webphone with the setparameter API or by URL)

Other important parameters you might need to set are the followings:

- sipusername (sometimes required for authentication or for Caller-ID purposes)
  If your SIP server requires a different auth vs user id, then you can set the user id with the username parameter and the authorization username with the sipusername. You can also use it to configure a Caller ID (different from the auth user name) if your SIP server accepts a different username/sipusername.
  With other words:
  - *The username will be used as user id (extension number / Caller–ID)*
  - *The sipusername parameter will be used as the authentication username*
  - *If only one is set, then it will be used for both the above purposes*
- displayname (usually from user input –enduser full name)
- proxyaddress (only if you have an outbound proxy different from your SIP server address)
- register (you might set it to 0 if no registration is not required)
- other sip account related settings (change any only if needed for your goals)
- engine priority (only if you prefer a specific engine for some special reason, otherwise the "best" possible engine will be used automatically)
- webrtcserveraddress (if you have your own WebRTC gateway or your softswitch has built-in WebRTC capabilities, set this to your websocket listener URL)
- codec and prefcodec (no any changes are recommended for these since the webphone is capable to auto negotiate the "best" possible codec to use, but you might change it if you have some special needs)
- other engine related settings (change any only if needed for your goals)
- callto and autoaction (in case if you wish to make an action at startup which otherwise can be done also with the API)
- other call divert related settings such as autoaccept, callforwardonbusy and others (you might use the API instead for these tasks)

- user interface related settings if you use the softphone skin (the softphone.html, included with the webphone)

There is no need to any "fine-tuning" of the parameters.
The webphone has a long list of configurable settings (listed below) for maximum flexibility, however it is not recommended to change any parameter unless strictly required to achieve your project goals. All parameters has meaningful optimal value by default or it is auto-negotiated regarding server/network/environment and unnecessarily altering the settings might result in sub-optimal behavior.

SIP account settings

Credentials and other basic SIP parameters:

serveraddress

(string)
The address of your SIP server (domain or IP + port), usually the same with your SIP domain.
It can be specified as IP address or as A or SRV domain name.
Specify also the port if your server is not using the default 5060; in this case append the port after the address separated by colon.
Examples:

mydomain.com (this will use the default SIP port 5060)
sip.mydomain.com:5062
10.20.30.40:5065
10.20.30.40 (this will use the default SIP port 5060)

This is the single most important parameter for the webphone (along with the username/password but those can be also entered by the user).
Default value is empty.

Note:

- *Sometime you might have to set also the SIP proxy address and the transport parameters (depending on your server side requirements). Read here for more details.*
- *If you are using a (sub)domain as your SIP server address, then we recommend to always set it's DNS A record also (not just the SRV record) for full WebRTC compatibility, since browsers might not ask for SRV records.*
- *If your server address is different from your SIP domain, then you might set the SIP domain as the realm parameter.*
- *If you SIP server doesn't allow UDP for the SIP signaling then set the transport parameter accordingly.*

username

(string)
This is the SIP account user name (User ID / Extension ID / Caller-ID).
Default value is empty.

Note:

- *Technically speaking, this is the user part of the Address of Record (AOR), sent in the "From" and "Contact" SIP headers.*
  It is used for authentication (if no sipusername parameter is set) and as A number/Caller-ID for the outgoing calls. Other users can make call to this endpoint by dialing this value. Some SIP service providers use a globally routable telephone number (DID) as the SIP username.
- *The username/password parameters are usually supplied by the user (via some user interface and then calling the setparameter API), however in some cases you might just set it statically in the webphone_config.js file (when caller user credentials doesn't matter). See more here. In case if you set it "statically" in the webpone_api.js and you are using the softphone skin (softphone.html) the username will not be asked from the enduser at login.*
- *Even if you don't need a username and/or your server accepts all calls without authentication, you must set the username to some value: the "anonymous" username might be used in this case.*
- *If you set the username setting to "Anonymous" then the username input box will be hidden on the "softphone" skin settings and login screens.*
- *If you wish to set a separate caller-id you can use this parameter to specify it and then use the sipusername parameter to specify the username used for authentication as specified in SIP standards. However please note that most SIP server can treat also the below mentioned "displayname" parameter as the caller-id so the usage of separate username/sipusername is usually not necessary and confusing. See more details here.*
- *If the showusername parameter is set to 2 then both the username (Caller-ID) and the auth username input will be displayed on the softphone skin login page.*
- *For maximum compatibility with third-party software and services, avoid using special and non-ASCII characters.*

sipusername

(string)

This is the username used for SIP digest authentication (Auth User Name or Authorization Name).
It is an optional setting which should be set only if your SIP server requires a different username for authentication.

If your SIP user name and auth user name are different, the proceed like this:

- *Set the webphone username to the SIP user name (also known as extension number or the user part of the AOR)*
- *Set the webphone sipusername parameter to the Auth user name (also known as Auth ID or PIN in some systems)*

If only one of the above are set (username or sipusername) then the same will be used for both purposes (both as user id and authorization user name).
If you are using the softphone skin (the softphone.html) then you might set the showusername parameter to 2 in the webphone_config.js to show both the username (Caller-ID) and the sipusername (Auth username) input on the login screen. Otherwise you can configure them separately only from the SIP account settings.

You need to set this if only the username parameter is not the same with the auth user name (when you need to use a different caller id and username for authentication).

- *If this is not set, then the username parameter will be used for both the caller-id (public-identity sent with the From SIP header) and authorization (private identity sent with the Authorization SIP header).*
- *If this is set, then this parameter will be used for authorization and the username parameter will be used for caller ID.*
- *If you expect the @ character (full SIP URI) in the sipusername, then set the handleusernameuri parameter to 0 to prevent extracting the domain from it on the softphone skin*

Default value is empty.

See more details here and here.

password

(string)
This is the SIP account password used for authentication.
Default value is empty.

Note:

- *Make sure to never hardcode the password in html or set it via insecure http. See more details here about security.*
- *You can use the webphone also without password (if not using via server or your server doesn't authenticate the users). In this case you can set the password to any value since it is supposed that it will not be required for calls or registrations.*
- *Usually you ask the password from the endusers and set it at runtime with the setparameters API. In case if you set it "statically" in the webpone_api.js and you are using the softphone skin (softphone.html) the password will not be asked from the enduser at login.*
- *For the softphone user interface you can also set hidepassword and pwdautocomplete parameters.*
- *If your IP-PBX accept blind registrations and/or calls then the value of the password doesn't matter (it will not be used anyway)*
- *If you set the password setting to "nopassword" then the password input box will be hidden on the "softphone" skin settings and login screens.*
- *If your IP-PBX doesn't require registrations or you are not using any server then you should set the "register" setting to 0.*
- *SIP passwords are always case sensitive.*

displayname

(string)
Optional SIP display name.
Specify default display name used in "from" or "contact" SIP headers.
This is often sent to peer as Caller ID when placing a call (if not overwritten on the server side).
Default value is empty (which means that only the "username" field will be sent to the peers).

Note:
This has nothing to do with the text displayed by the webphone skin. This is a SIP parameter to specify the display name sent to the others with call setup.
See more details here.

realm

(string)
Optional parameter to set the SIP realm (logical SIP domain) if not the same with the serveraddress.
Rarely required. (Only if your VoIP server has different realm setting as its domain and it strictly enforces that realm for authentication)
Default value is empty. (By default the serveraddress will be used without the port number)

proxyaddress

(string)
Outbound SIP proxy address (Examples: mydomain.com, proxy.mydomain.com:5065, 10.20.30.40:5065)
Leave it empty if you don't have a stateless proxy. (Use only the serveraddress parameter)
Default value is empty.

Note:
Set to "NULL" if you already set it before (to a wrong value) and wish to clear it.
If the showproxyaddress parameter is set to 2 then the proxy address input will be displayed on the softphone skin login page.

register

(number)
With this parameter you can set whether the softphone should register (connect) to the sip server.
0: no (the webphone will not send REGISTER requests)
1: auto guess (yes if username/password are preset, otherwise no)
2: yes (and must be registered before to make calls)
Default value is 1.

Related parameters: autostart (start webphone with page load), appengine_startat (native engine launch), startsipstack (NS engine specific), autologin (softphone skin specific), mustconnect (if set to true the webphone will not allow any calls before registered; default is false), regtimeout (register no answer timeout for the WebRTC engine in milliseconds; default is 6000; 0 means no retry).

registerinterval

(number)
Registration interval in seconds (used by the re-registration expires timer).
Default value is 120 or 300 depending on the circumstances.

This is important for SIP servers to find out unexpected termination of the webphone application or webpage such as killing the browser, power loss or others (so the server will know that the client is no longer alive if this time is expired, but no new re-registration were received from the client).

Note: we don't recommend to set the re-register interval below 30 seconds (it just causes unnecessary server load; below 30 seconds most of the SIP servers will not decide anyway and some servers doesn't accept such short re-registration periods). Also you should not set it longer than 3600 (one hour).

Engine related settings

Library, engine, transport, SIP, media, DTMF, presence and related settings:

engine priority

By default the webphone will choose the "best" suitable engines automatically based on OS/browser/server support. This algorithm is optimized for all OS and all browsers so you can be sure that your users will have the best experience with default settings, however, if you wish, you can influence this engine selection algorithm by setting one or more of the following parameters:

- **enginepriority_java**
- **enginepriority_webrtc**
- **enginepriority_ns**
- **enginepriority_flash**
- **enginepriority_app**
- **enginepriority_p2p**
- **enginepriority_accessnum**
- **enginepriority_nativedial**
- **enginepriority_otherbrowser**

Possible values:

0: Disabled (never use this engine)
1: Lower (decrease the engine priority)
2: Normal (default)
3: Higher (will boost engine priority)
4: Highest (will use this engine whenever possible)
5: Force (only this engine will be used)

Example:
-if you wish to prioritize the NS engine, just set: enginepriority_ns: 3
-if you wish to prioritize the WebRTC engine, just set: enginepriority_webrtc: 3
-if you wish to avoid WebRTC engine, just set: enginepriority_webrtc: 1

-if you wish to force the WebRTC engine only, just set: enginepriority_webrtc: 5

Best practices:
-You should not change the engine priorities unless you have a good reason (one of the main strength of the webphone is automatic optimal engine selection based on circumstances, so there should be rare cases when you might need to adjust this manually)
-Even if you have a favorite engine, you should not force it or disable the others. Just set your favorite engine priority to 3 or 4. This way even endusers which doesn't have a chance to run your favorite engine might be able to make calls with other engines.
-Even if for some reason you don't like an engine, don't entirely disable it, just lower its priority (set its enginepriority to 1 instead of 0). This means that the webphone will pickup the engine only if there is no -any other alternatives and having the webphone working with an unwanted engine is much better then not working at all
-It is recommended to set engine priority from static settings (for example in the webphone_config.js file) and not at runtime. In case if you have to change the engine priorities at runtime using the setparameter() API, then you must do it only from the onAppStateChange "loaded" event, not sooner or later.

The engines also have a built-in default priority number assigned, which can range from 0 to 100 and can be changed with the enginedefpriority_ENGINENAME settings.
Default values:

enginedefpriority_java: 32
enginedefpriority_webrtc: 20
enginedefpriority_flash: 13
enginedefpriority_ns: 30
enginedefpriority_app: 10
enginedefpriority_p2p: 5
enginedefpriority_callback: 5
enginedefpriority_natividial: 3

You should not touch these values!

Note:

- *You can disable the NS engine for linux by setting the linnsengine parameter to false.*
- *You can disable the NS engine for MacOS by setting the macnsengine parameter to false.*
- *You can disable the linux NS engine for windows by setting the winlinmacnsengine parameter to false (this is for test only and defaults to false).*
- *You can disable the NS engine for Windows only if you completely disable the NS engine by setting enginepriority_ns to 0.*
- *Special case: If the enginepriority_webrtc is set to 4 and the enginepriority_ns is set to 3, then the webphone will always checks first if the NS engine is already installed and will use it instead of WebRTC (but if not already installed, then it will prioritize WebRTC over the NS engine)*

webrtcserveraddress

(string)
WebSocket server URL for WebRTC.
Optional setting to indicate the domain name or IP address of your websocket service used for WebRTC if any (your server address and websocket listen port).
The following URI format can be used:

protocol:address:port/path

Where:

The protocol can be ws (unsecure websocket) or wss (secure websocket)
The address can be an IP address or a (sub)domain name
The port is where the websocket is listening and it must be specified if not 80
The path part must be set if your server require it (For example Asterisk requires the "ws" path). Check your server documentation for the exact format.

Examples:

ws://mydomain.com
ws://10.20.30.40:5065
wss://subdomain.mydomain.com/anypath
wss://asterisk.mydomain.com:8089/ws
wss://sip.mydomain.com:8080

Default value is empty (which means auto service discovery and if no webrtc service found then the mizu webrtc service can be used if accessible).
Note:

- You can verify if the address you have set is correct by trying a simple websocket connect with some tool.
- Latest browsers require secure websocket (wss), otherwise media (call recording) permissions will be denied. You will need to install an SSL certificate for your WebRTC server for this and set this parameter with the domain name (not IP address). This is needed only if your VoIP server is WebRTC capable or you have your own WebRTC to SIP gateway. Otherwise no changes are required.
- Multiple servers can be set separating the addresses by comma. In this case the webphone will use the closest by default (with fastest response time) and it is capable to failover to other server/gateway

- By default the webphone might use our WebRTC-SIP service if your SIP server has a public IP (accessible over the internet) AND the webphone auto selects the WebRTC engine AND you haven't set your own websocket listener for webrtc. This is a free service tier offer by us for your convenience, but its usage is optional and the webphone works also without this service.
  More details about webrtc can be found in the FAQ: here and here.

rtmpserveraddress

(string)
Optional setting to indicate the address (domain name or IP address + port number) of your flash service if any (flash media + RTMP) which might improve the SIP availability in some exotic or outdated browsers where the other engines such as WebRTC, Java or NS are not available.
If not set, then the mizu flash to sip service might be used (rarely used in normal circumstances and it can be also disabled if for some reason you always wish to avoid it).
Format: yourdomain.com:rtmpport
Example: 10.20.30.40:5678
Default value is empty.

stunserveraddress

(string)
STUN server address in address:port format (RFC 5389)
You can set to "NULL" to completely disable STUN. Usually STUN is not required at all, however in some circumstances (mostly depending on your SIP server and outbound trunk capabilities) it might help to avoid unnecessary RTP routing (direct peer to peer RTP routing bypassing your server).
Examples:

11.22.33.44:3478
mystunserver.com:3478
NULL

By default (if you leave this setting unchanged) the webphone will use the Mizutech STUN servers (unlimited free service for all webphone customers). You can change this to your own STUN server or use any public server if you wish.
Note:
If you set an incorrect STUN server, then the symptoms are extra delays at call setup (up to "icetimeout").
Instead of using the stunserveraddress parameter, you might use the ice parameter to configure both stun and turn together.

turnserveraddress

(string)
TURN server address in address:port format (RFC 5766)
You can set to "NULL" to completely disable TURN.
Examples:

11.22.33.44:80
mystunserver.com:80
NULL

If your server is capable to route the RTP, then TURN is usually not required at all.
TURN is used mostly only with WebRTC and might be useful only if the webphone cannot send the media directly to the peer (which is usually your VoIP server) and your server doesn't support TCP candidates. For example if all UDP is blocked or only TCP 80 is allowed or you need peer to peer media via TURN relay.
By default (if you leave this setting unchanged) the webphone can use the Mizutech TURN servers. If you wish, you can deploy your own TURN server using the popular open source coturn server. The MizuTech WebRTC to SIP gateway also has its own built-in TURN server and there is no need to set this parameter.

Note: Instead of using the turnserveraddress parameter, you might use the ice parameter to configure both stun and turn together.

turnparameters

(string)
Any TURN URI parameter. *Required only if the turnserveraddress is set.*
Example: transport=tcp

turnusername

(string)
Username for turn authentication. *Required only if the turnserveraddress is set.*

turnpassword

(string)
Password for turn authentication. *Required only if the turnserveraddress is set.*

Note: in case if you are using the mizu server or gateway then don't bother securing the turn password as the server will use also other mechanisms to reject unauthorized TURN requests.

ice

(string)
Instead of using the above STUN and TURN parameters, you can use this ice parameter in the following format:
ice: ' [{ url:"stun:stun.l.google.com:19302"}, { url: "turn:user@myturnserver.com", credential: "myTurnPassword"}] '
or
ice: ' [{ url:"stun:stun.l.google.com:19302"}, { url: "turn:myturnserver.com:PORT", username: "myTurnUsername", credential: "myTurnPassword"}] '

In case if you don't use WebRTC or you are using mizutech WebRTC-SIP gateway (MRTC) then there is no need to set any ice/stun/turn parameter as they are discovered automatically (service provided by the gateway). Otherwise the default in this case is:
ice: '[{ url:"stun:rtc.mizu-voip.com:8090"}, { url:"turn:rtc.mizu-voip.com:80?transport=tcp","username":"mzturnusr","credential":"***"}]'

In case if you configured the webrtcserveraddress parameter to point to your WebRTC server or gateway websocket listener, then you might reconfigure this ice parameter if you also have stun/turn service (you might use the coturn).
You might also configure any other stun service. A list of public stun servers can be found here or here.
There are no public free turn servers (as turn is routing the media also and not cheap to host).
You can also set multiple STUN and TURN servers if you wish.

icetimeout

(number)
Timeout for ICE address gathering (STUN/TURN/others) in milliseconds.
Default is 2000 (2 seconds).
You might increase in special circumstances if you are using some slow STUN/TURN server or decrease if peer address is public (like if your SIP or WebRTC server is on public IP always routing the media, so calls will work also without STUN).

stunturnonlocal

(number)
STUN/TURN usage on local LAN. This setting helps to avoid unnecessary STUN/TURN connections, negotiations and call setup delay.
Possible values:
0: no (will set the TURN and STUN servers to null, regardless of user settings and configurations)
1: STUN only (keep using STUN, but no TURN)
2: use both STUN and TURN (use STUN and TURN also with local servers)
Default: 1

use_rport

(number)
Check rport in SIP signaling (requested and received from the SIP server by the VIA header)
0=don't ask (rtport request will not be added to the VIA header)
1=use only for symmetric NAT (only when it is sure that the public address will be correct)
2=always (always request and use the returned value except if already on public ip)
3=request even on public IP (meaningless in most cases)
9=request with the signaling, but don't use the returned value (good If you want to keep the local IP and for peer to peer calls)
Change to 0 or 2 only if you have NAT issues (depending on your server type and settings)
(You might adjust also the use_fast_stun parameter if you change the use_rport)

Default is 1

use_fast_stun

(number)

Fast stun request on startup.
-1=force private address (if the client has both private and public IP, than the private IP will be sent in the signaling)
0=no
1=use only for stable symmetric NAT
2=use only if both tests match even if not symmetric (recommended)
3=use for symmetric NAT even if only one match
4=always
5=use even on public IP
Change if you have NAT issues (depending on your server type and settings)
(You might adjust also the use_rport parameter if use_fast_stun is changed)
Default is 2

p2psignaling

(number)

Specify to enable or disable peer to peer signaling routing feature.

Possible values:

- 0: no (direct user to user routing fully disabled forcing the call to travel via upper SIP server –will also add X-P2P: no header)
- 1: yes (full p2p direct signaling enabled when possible)
- 2: proxy (might go over proxy socket processing. Actually this is a special option between no and yes where the signaling might be forwarded by proxy but might not sent directly or via upper server –will also add X-P2P: proxy header)
- 3: process (might go over proxy with full endpoint processing. Actually this is a special option between no and yes where the signaling might get fully processed by proxy but might not travel via upper server –will also add X-P2P: process header)

Default is 1.

You might set to 0 (disable) in case if the call signaling has to reach your server (for example if you are using ring groups to fork the calls, or you wish to record the calls on your SIP server or if for billing purposes it is important that your server see also the enduser to enduser calls).
Otherwise with the default 1 value the webphone might contact directly the other party or the WebRTC gateway (in case of the WebRTC engine) might route the call directly between endusers, thus saving your server resources and shortening the network path.
Note: this might add a X-P2P: no/process/proxy/yes header to the SIP signaling notifying the gateway/server if routing have to be forced/ignored.

use_fast_ice

(number)

Fast ICE negotiations (for p2p rtp routing):
-1: suggest server side media routing (X-P2PM: 1)
0: no (set to 0 only if your server needs to always route the media)
1: auto
2: yes
3: always (not recommended)
Default is 1

Note:
If set to 1 or 2 then stun should not be disabled
Old parameter name was p2prtp.
Specify to enable (1,2) or disable (0,-1) peer to peer media routing feature (direct RTP routing). You might set to -1 (disable) in case if the call media (RTP) must be routed through your SIP server (for example if you wish to voice record the calls on your server). Otherwise the webphone might route the media directly between webphone instances thus saving your server resources and shortening the network path. Please note that the media might be routed directly between the endpoint even if this parameter is set to 0, according the SDP and ICE negotiation (this can be influenced by your SIP server configuration by the usual NAT and RTP related settings globally or by extension).
Default is 1 (p2p for media enabled routing encrypted RTP directly between the endpoints when possible).

autodetectwebrtc

(number)

Try to auto-detect webrtc address if not set (if the active SIP server has built-in WebRTC capabilities)
0: no
1: yes
Default is 1.

appengine_startat

(number)
Specify how to launch the app engine (native dialer / native softphone) if the rare events when it might be required (if no any other engine can be used).
1: at start
2: auto (default)
3: at call attempt

android_nativedialerurl

(string)
Android native softphone download URL if any. (Optional setting to allow alternative softphone offer on Google Play).
Note: Android browsers has support also for WebRTC so this might be selected only with old phones or if you disable WebRTC.
Default is empty (which means the default app).

ios_nativedialerurl

(string)
iOS native softphone download URL if any. (Optional setting to allow alternative softphone offer on Apple App Store).
Old Safari browsers (up to v.11) under iOS doesn't offer any plugin for VoIP so the webphone can use its native softphone (will be auto provisioned from your webphone settings).
Default is empty (which means the default app).

app_protocol

(string)
Specify the URI handler to be registered to launch native apps if needed.
The defaults are: sip, tel, webphone
You might reconfigure it if you use some other third-party app that is triggered for other protocol. For example: app_protocol: "myapp"

accessnumber

(string)
Set this if your IP-PBX has an access number where users can call into from PSTN and it can forward their call on VoIP (IVR asking for the target number).
This can be used when no other engines are working (no suitable environment, no internet connection).
Default is empty.

callbacknumber

(string)
Set this if your server has a callback access number where users can ring into and will receive a call from your server (possibly with an IVR which might offer the possibility to specify the destination number via DTMF).
This can be used when no other engines are working (no suitable environment, no internet connection) and it is very useful in situation where call from the server is cheaper than user call to server.
Default is empty.

useragent

(string)
This will overwrite the default User-Agent setting.
Do not set this when used with mizu VoIP servers because the server detects extra capabilities by reading this header.
Default is empty.

customsipheader

(string)

Set a custom sip header (a line in the SIP signaling) that will be sent with all messages. Can be used for various integration purposes and usually has a key:val format. For example: myheader:myvalue.
Default is empty.

Note:
Custom SIP headers should begin with "X-" to be able to bypass servers, gateways and proxies (For example: X-MyHeader: 47).
You can add more than one header, separated by semicolon or "CRLF" in case if you must use semicolon in the value (For example: customsipheader: 'x-key1: val1;x-key2: val2',).
You can also use the setsipheader API call at runtime. It can be used instead of this parameter or you can use both together (both this parameter and the API)

contact_uri_parameters

(string)
Set custom SIP Contact URI parameters for REGISTER and INVITE requests.
Default is empty.
Set to 'NULL' to clear.

Example:
If the original contact line looks like Contact: <sips:1111@any.invalid;transport=wss>;expires=180
and you set the contact_uri_parameters to 'a=1;b=2',
then the contact will be modified to: Contact: <sips:1111@any.invalid;transport=wss;a=1;b=2>;expires=180

contact_parameters

(string)
Set custom SIP Contact parameters for REGISTER and INVITE requests.
Default is empty.
Set to 'NULL' to clear.

Example:
If the original contact line looks like Contact: <sips:1111@any.invalid;transport=wss>;expires=180
and you set the contact_parameters to 'x=1;y=2',
then the contact will be modified to: Contact: <sips:1111@any.invalid;transport=wss>;x=1;y=2;expires=180

autoprovisioning

(number)
Specify whether the settings have to be downloaded from a central config server.
Possible values:

0: no
1: auto (if not ip or domain was entered)
2: yes

useaudiorecord

(number)
Specify whether calls should fail or succeed also without a microphone device.
-1: disable microphone recording (call will use only playback from remote)
0: no (calls with be allowed even if client doesn't have any microphone audio device)
1: with warning (call will be allowed but a warning message might be displayed for the user; sdp might be fixed; call might be reloaded with audio only)
2: yes (calls might fail if user doesn't have a microphone device)
3: same as 1 but if incoming video call then it will retry with audio only
Default is 1.

If your device doesn't have microphone or doesn't need microphone recording at all, you should also set the useaudiodevicerecord parameter to false. Old parameter name was *checkmicrophone*.

usecommdevice

(number)
Use VoIP optimizations on windows (WAVE_MAPPED_DEFAULT_COMMUNICATION_DEVICE). This will also enable audio ducking (audio focus: auto lowering the volume for other processes while the webphone is in call).
0: No
1: Yes
2: Force
Default value is 1.
Set to 0 if you encounter audio volume issues, especially for multi-calls.

checkvolumesettings

(number)
Check if audio device is muted or volume settings are too low (and un-mute and/or increase volume if necessary).
0: no
1: at first run
2: always
Default value is 1

callreceiver

(number)
Enable background call listener.
If the WebRTC engine is used, this means push notifications.
If you are using the NS engine on Windows, your webpage can be started automatically on incoming calls if you set this parameter to true.
When background calls is turned on, the NS engine will listen continuously for incoming call and chat notifications and will launch your website with the incoming session details if your page is not already started by the user. This will enable incoming SIP calls even if the browser is closed.
Possible values:
-1: auto
0: disable all incoming calls
1: enable incoming calls (while app is running; no background listener)
2: enable also background calls (WebRTC push or NS engine or android service)
Default is -1.

Note:

- This has nothing to do with the HTML5 notification support which can be configured with the incomingcallpopup parameter.
- In case if you need only NS engine background calls but not push notifications, then you might set the enablepush parameter to 0 (possible values: -1: auto, 0: no, 1: yes).
- The old parameter was named backgroundcalls and it is still supported as-is

Here you can read more details about VoIP push notifications.

transport

(number)
Transport protocol for the SIP signaling.
-1: Auto detect
0: UDP (User Datagram Protocol. The most commonly used transport for SIP)
1: TCP
2: TLS (SIPS / encrypted signaling)
3: HTTP tunneling (both signaling and media. Supported only by mizu server or mizu tunnel with the NS or Java engines)
4: HTTP proxy connect (requires tunnel gateway or server with the NS or Java engines)
5: Auto tunnel (automatic failover from UDP to HTTP as needed if tunnel gateway or server is used with the NS or Java engines)
Default is -1.

Note:
This parameter is important only for the NS or Java engines.
With the WebRTC engine the webphone will always use websocket (WSS or WSP) transport for the signaling and this parameter will be used only if you are using a WebRTC-SIP gateway (to request the specified transport protocol between the gateway and the upper server)

TLS related notes:

- *To encrypt also the media, set the mediaencryption parameter to 2.*
- *When using TLS, you might need to change the port in the serveraddress parameter (from the default 5060) to 5061. For example: serveraddress: 'yourdomain.com:5061')*
- *WebRTC always uses full encryption. When using WebRTC this parameter will mean the SIP transport protocol to be used with your server as for WebRTC itself the transport is controlled by the browser: http/https, websocket/secure websocket (ws/wss) and always encrypted media (DTLS/SRTP).*
- *More details here*

localip

(String)

Specify local IP address to be used.

This should be used only on devices with multiple ethernet interface to force the specified IP.

Default is empty (autodetect)

Note: This setting is not applicable for WebRTC (In case of WebRTC this is handled entirely by the browser internal WebRTC stack)

signalingport

(number)

Specify local SIP signaling port to use.

Default is 0 (a stable port which is selected randomly at the first usage)

Note: This is not the port of your server where the messages should be sent. This is the local port of the signaling socket (Usually UDP if you don't explicitly set the transport to TCP)

Note: This setting is not applicable for WebRTC (In case of WebRTC this is handled entirely by the browser internal WebRTC stack)

rtpport

(number)

Specify local RTP port base.

Default is 0 (which means signalingport + 2)

Note: If not specified, then VoIP engine will choose signalingport + 2 which is then remembered at the first successful call and reused next time (stable rtp port).

If there are multiple simultaneous calls then it will choose the next even number.

Note: This setting is not applicable for WebRTC (In case of WebRTC this is handled entirely by the browser internal WebRTC stack)

keepaliveival

(number)

NAT keep-alive packet send interval in milliseconds.
Set to 0 to disable.
Default value is 28000 (28 sec)

wskeepaliveival

(number)

Keep-alive interval for the WebRTC websocket in milliseconds.
Set to 0 to disable.
Default value is 30000 (30 sec)

sendrtponmuted

(boolean)

Send rtp even if muted (zeroed packets)

Set to true only if your server is malfunctioning when no RTP is received.

Default value is false.

mediaencryption

(number)

Media encryption method

-1: auto guess

0: not encrypted

1: auto (will encrypt if initiated by other party)

2: SRTP

Default is -1 which auto set to 1 if TLS transport is used, otherwise will use 0/not encrypted

Note:

- *This parameter will not have effect with the WebRTC engine since WebRTC calls are always secured using DTLS/SRTP encryption for the media streams.*
- *With SRTP it is recommended to also set the signaling transport to TLS if you are using the NS or Java engines.*
- *More details here*

dtmfmode

(number)
DTMF send method
0=disabled
1=SIP INFO method (out-of-band in SIP signaling INFO messages)
2=auto (auto guess from peer advertised capabilities)
3=INFO + NTE (not recommended)
4=NTE (Named Telephone Events as specified in RFC 2833 and RFC 4733)
5=In-Band (DTMF audio tones in the RTP stream)
6=INFO + InBand (not recommended)

Default is 2.

Note:

- *When more than one method is used (dtmfmode 3 or 6), the receiver might receive duplicated dtmf digits*
- *Received DTMF are recognized by default in both INFO or RFC2833/ RFC4733 formats (no In-Band DTMF processing)*
- *DTMF messages can be sent from the existing skin templates or using the dtmf API. Incoming dtmf messages are displayed automatically on skins or can be caught with the onDTMF callback.*
- *DTMF messages can be also sent by adding it to the called number after a comma. For example if you make a call to 123,456 then it will call 123 and then it will send dtmf 456 once the call is connected.*
- *If DTMF doesn't work by default with Asterisk, then we recommend to set the dtmfmode=info configuration for the extensions used by the webphone*
- *WebRTC doesn't support InBand DTMF*
- *If you are using the mizu WebRTC-SIP gateway, you can set dtmf type for both inbound and outbound globally with the fs_dtmf_type_intern and fs_dtmf_type_extern server side settings and also per user with the inbounddtmf and outbounddtmf webphone parameters. The out-bound is also handled automatically after the dtmfmode setting (defaults to SIP info) and for in-bound both info and rfc2833 are accepted by default.*
- *If gateway side WebRTC DTMF conversion is required, you might also set the use_fast_ice parameter to -1.*

playdtmfsound

(number)
Specify whether the webphone should generate local DTMF tone when DTMF is sent.
0=no
1=if one digit
2=always (also when multiple digits are sent at once)
Default is 1.

earlymedia

(number)
Start to send media when session progress is received.
0: no
1: reserved
2: auto (will early open audio if wideband is enabled to check if supported)
3: just early open the audio
4: null packets only when sdp received (NS only)
5: yes when sdp received
6: always forced yes
Default is 2.

prefcodec

(string)
Set your preferred audio codec. Will accept one of the followings: pcmu, pcma, g.711 (for both PCMU and PCMA), g.729, gsm, ilbc, speex, speexwb, speexuwb, opus, opusnb, opuswb, opusuwb, opusswb
Default is empty which means the built-in optimal prioritization.
By default the engine will present the codec list optimized regarding the circumstances (the combination of the followings):

- available client codec set (not all engines supports all codecs)

- server codec list (depending on your server, peer device or carrier)
- internal/external call: for IP to IP calls will prioritize wideband codecs if possible, while for outbound calls usually G.729 will be selected if available
- network quality (bandwidth, delay, packet-loss, jitter): for example iLBC is more tolerant to network problems if supported
- device CPU: some old mobile devices might not be able to handle high-complexity codec's such as opus or G.729. G711 and GSM has low computational costs

You can also fine-tune the codec settings with the use_xxx settings where xxx is the codec name as described in JVoIP documentation.

codec

(string)
List of allowed audio codec's separated by comma.
By default the webphone will automatically choose the best codec depending on available codec's, circumstances (network/device) and peer capabilities.
Set this parameter only if you have some special requirements such as forcing a specific codec, regardless of the circumstances.
Example: Opus,G.729,PCMU (This will disable Speex, GSM, iLBC, GSM and PCMA).
Default: empty (which means auto detection and negotiation)
Recommended value: leave it empty

Under normal circumstances, the following is the built-in codec priority:

I. Wideband Speex and Opus (These are set with top priority as they have the best quality. Likely used for VoIP to VoIP calls if the peer also has support for wideband)
II. G.729 (Usually the preferred codec for VoIP trunks used for mobile/landline calls because it's excellent compression/quality ratio for narrowband)
III. iLBC, GSM (If G.729 is not supported then these are good alternatives. iLBC has better characteristics and GSM is better supported by legacy hardware)
IV. G.711: PCMU and PCMA (Requires more bandwidth, but has the best narrowband quality. Preferred from WebRTC if Opus is not supported as these are present in almost any WebRTC and SIP endpoints and servers)

With the NS and Java engines you can also use the use codecname settings to disable/enable codec. Possible values: 0=never,1=don't offer,2=yes with low priority,3=yes with high priority. For example to disable all codec except PCMU you can use the following settings:
use_pcmu: 3, use_pcma: 0, use_g729: 0, use_gsm: 0, use_speex: 0, use_speexwb: 0, use_speexuwb: 0, use_opusnb: 0, use_opuswb: 0, use_opusuwb: 0, use_opusswb: 0, use_ilbc: 0, alwaysallowlowcodec: 0

vcodec

(string)
List of allowed video codec's separated by comma.
You might use this parameter to exclude some codec from the offer list.
For example if you don't wish to use VP8, then set this to: "H264, H263"
Default: empty (which means auto detection and negotiation)
More details here.

prefvcodec

(string)
Set preferred video codec.

video

(number)
Enable/disable video.

-1: auto (default)
0: disable
1: enable
2: force always

More details about video calls can be found here.

audio_bandwidth

(number)
Max bandwidth for audio in kbits.
It will be sent also with SDP "b:AS" attribute.
Default is 0 which means auto negotiated via RTCP and congestion control.

video_bandwidth

(number)

Max bandwidth for video in kbits for WebRTC video.

It will be sent also with SDP “b:AS” attribute.

Default is 0 which means auto negotiated via RTCP and congestion control.

video size parameters

(number)

You can suggest the size of the video (in pixels) with the following parameters:

- video_width
- video_height
- video_min_width (WebRTC only)
- video_min_height (WebRTC only)
- video_max_width (WebRTC only)
- video_max_height (WebRTC only)

You might set all of these to 0 to not set any constrains. Otherwise some default values might be applied.

videofacing

(string)

Will set the facingMode WebRTC video parameter:

Possible parameters:

- "no": will not set the facingMode at all (default)
- "user": The video source is facing toward the user; this includes, for example, the front-facing camera on a smartphone.
- "environment": The video source is facing away from the user, thereby viewing their environment. This is the back camera on a smartphone.
- "left": The video source is facing toward the user but to their left, such as a camera aimed toward the user but over their left shoulder.
- "right": The video source is facing toward the user but to their right, such as a camera aimed toward the user but over their right shoulder.

screensharing

(number)

Enable/disable screen sharing:

0=No (default)

1=Auto (if supported by the platform)

2=Yes (always force)

Default is: 1

Note:

Softphone skin users: If the screensharing is set, a screen share button will appear on the softphone user interface.

Enable extension:

- *Screen sharing was a long term experimental feature in WebRTC standards and you might need a browser extension to enable it.*
- *You might also need the followings:*
- *Chrome: start the browser with --enable-usermedia-screen-capturing flag*
- *Firefox: in the about:config create a media.getusermedia.screensharing.enabled key and set its value to true and in media.getusermedia.screensharing.allowed_domains append the IP address of your server*

The latest webphone will try to resolve screensharing without the need for any extensions in latest browsers.

codecframecount

(number)

Number of payloads in one UDP packet (frames per packet). This will directly influence RTP packet time (packetization interval) and packet size as shown here.

- By default it is set to 0 which means 2 frames for G729 and 1 frame for all other codec.
  This usually means 20 msec rtp packetization interval for all codec’s and it is the most optimal setting, compatible with all SIP/media stack implementations.

- In case if you wish to minimize delay, then you might set the codecframecount to 1. This usually will result in 10 msec packetization interval and will increase the required bandwidth by 40% due to high header/data ratio.
- In case if you wish to minimize bandwidth, then you might set the codecframecount to 4.

Note: with the WebRTC engine the codec frame count is controlled by the browser WebRTC media stack and can't be changed. Usually it will be 20 msec frame time.

conferencetype

(number)

Specify how to handle conference, especially for the WebRTC engine as other engines has built-in local RTP mixer.

0: disabled

1: auto

2: native only with local rtp mixer (on WebRTC will force the NS or java engine)

3: server side conference mixing controlled by DTMF

4: 3-way SIP conference

5: conference rooms via SIP MESSAGE and REFER

6: conference rooms via SIP MESSAGE

7: conference rooms via SIP REFER

Default is 1.

Old parameter name was conferenceroom, which is deprecated now.

For server side DTMF conference, you should also set the use_fast_ice parameter to -1.

For more details see here.

plc

(boolean)

Enable/disable packet loss concealment
Default is true (enabled)

vad

(number)

Enable/disable voice activity detection.
0: auto
1: no
2: yes for player (will help the jitter)
3: yes for recorder
4: yes for both
Default is 2.

Notes:
-This parameter is supported only by the NS and Java engines (not for WebRTC. For WebRTC you might use the getrtcpeerconnection API as described here)
-The vad parameter is automatically set to 4 by default if the aec2 algorithm is used.
-If you wish to use VAD related statistics in your application, you might have to also set the "vadstat" parameter after your needs. Possible values: 0=no,1=auto (default),2=detect no mic audio,3=send statistics. See the VAD notification for more details.
-If you want to disable audio related notifications for the NS and Java engines (microphone warning on no signal detected) then set the "enablenomicvoicewarning" parameter to 0

aec

(number)

Enable/disable acoustic echo cancellation

0=no

1=yes except if headset is guessed

2=yes if supported

3=forced yes even if not supported (might result in unexpected errors)

Default is 1.

aec2

(number)

Secondary AEC algorithm.
0=no
1=auto
2=yes
3: yes with extra (this might be too much under normal circumstances)

Default is 1

Note: for maximum echo cancellation you can set the aec to 1,2 or 3 and aec2 to 2 or 3.

agc

(number)
Automatic gain control.
0=Disabled
1=For recording only
2=Both for playback and recording
3=Guess
Default value is 3

silencesupress

(number)
Enable/disable silence suppression
Usually not recommended unless your bandwidth is really bad and expensive.
-1=auto
0=no (disabled)
1=yes
Default is -1 (which means no, except mobile devices with low bandwidth)

denoise

(number)
Noise suppression.
0=Disabled
1=For recording only
2=Both for playback and recording
3=Auto guess
Default value is 3

jittersize

(number)
Although the jitter size is calculated dynamically, you can modify its behavior with this setting.
0=no jitter,1=extra small,2=small,3=normal,4=big,5=extra big,6=max
Default is 3

enablepresence

(number)
Enable/disable presence.
Possible values:
-1: auto
0: disable presence
1: auto (if presence capabilities detected)
2: always enable / force

Default is 1.
More details here.

presenceuserlist

(string)
List of users separated by comma to request presence state from (the webphone will SUBSCRIBE to their presence state and will accept NOTIFY reports).
You can also use the checkpresence() API for this.

enableblf

(number)

Enable/disable BLF (busy lamp field).

Possible values:

0: disable BLF
1: auto (from here it will auto switch to 0 or 2 regarding the circumstances –whether BLF was initiated and succeed/failed)
2: enable BLF
3: force always (if you set to 3 then it can't be switched off later and will use BLF even after failure)

Default is 1.

Note: If BLF is an important feature for you, then we recommend the usage of the NS or Java engines as this is unreliable with WebRTC (set the enginepriority_ns and enginepriority_java to 4)
More details here.

blfuserlist

(string)

List of users separated by comma to request call state from (the webphone will SUBSCRIBE to their call state and will accept NOTIFY reports).

You can also use the checkblf() API for this.

minserviceversion

(number)

Specify the minimum accepted NS engine version.

This is applicable only if the NS engine is used and the webphone will ask the user to upgrade if the installed NS plugin version is lower than this number (one click installer).

By default this is handled automatically by the webphone and you should change it only if you need to enforce a specific NS engine version for some reason.

Special values:

-2: disable asking for upgrades regardless of the NS engine version

-3: completely disable asking for all kind of ns engine install and upgrades

Default value: depends on the webphone version

Note:

- *The NS service plugin version for v.3.6.22061 engine executable which was shipped with webphone v.3.6 is 55 (so you might set the "minserviceversion" setting to 55 to force the latest version for all users)*
- *Upgrades for very old (outdated/incompatible) NS engines are handled automatically by the webphone (with a reasonable default "minserviceversion" settings in new webphone versions)*
- *Never set this to a higher value then the latest NS plugin version (will result in continuously asking to upgrade)*
- *(Developer info: this value is loaded from the NS engine MAPPVERSION define)*

More details about NS engine upgrades can be found here

nsupgrademode

(number)

Set how/when to upgrade the NS engine.

-1: never (not recommended. Use the 0 option instead which might trigger an upgrade only on user interaction or if there are no other options)

0: delayed background install only (the new version will be downloaded and applied silently in the background with no user interaction)

1: background or ask for user (default behavior to handle the upgrade. Might be delayed especially if the enduser doesn't actively using the NS engine)

2: immediate install (will ask the enduser immediately to download and install new versions)

Default: 1

Note: To completely disable all NS engine upgrades (which is not recommended), you should set the nsupgrademode to 0 or -1 and the autoupgrade to 6.

extraregisteraccounts

(string)

Use this setting for multi-account registration.

You can specify multiple SIP accounts in the following format:
IP,usr,pwd,t,proxy,realm;IP2,usr2,pwd2,t2,proxy2,realm2; IP3,usr3,pwd3,t3,proxy3,realm3;

Account parameters:

- IP: is the SIP server IP or domain name
- usr: is the SIP username
- pwd: is the SIP password
- t: is the register timeout in seconds (optional)
- proxy: SIP proxy (optional)
- realm: SIP realm (optional)
- authusr: auth (optional. Use only if separate extension id and authorization username have to be used)

Most of the parameters are optional.
The parameters have to be separated by comma (,) and the accounts have to be separated by semicolon (;).
All have to be passed in a single line which should look like this: server,usr,pwd,ival;server2,usr2,pwd2,ival2;
Example: sip1.mydomain,com,1111,xxx,300;sip2.mydomain.com,2222,xxx;

For more details read here.

autostart

(number)
Specify whether the webphone stack should be started automatically on page load.
Possible values:
0: no (webphone will not start automatically and you might need to call the start() API)
1: yes (webphone will start once loaded if previous session was not failed)
2: always (webphone will always auto start, regardless of previous success)
Default is 1.

If set to 0 or 1, then the start() method needs to be called manually in order for the webphone to start. Also the webphone will be started automatically on some other method calls such as register() or call().

You might set this parameter to 0 to prevent the auto initialization of the webphone, so you might delay the start() until actually the user wish to interact with your phone UI (such as pushing your click to call button).

Related parameters: register (auto register after start), appengine_startat (native engine launch), startsipstack (NS engine specific), autologin (softphone skin specific)

forcereregister

(number)
Specify whether you wish to disable/force reregistrations.
Please note that this is not about the normal SIP re-registrations enforced by the expires interval (That is handled automatically regardless of this settings).
This setting is considered when the sip stack unregisters unexpectedly due to error response, no response, no network or other reasons.
Possible values:
0: never auto reregister
1: try auto reregister if it was already registered successfully before
2: always force reregister regardless of the error (for example this will attempt to reregister even if user supplied wrong username/password and server rejected the registration because of the wrong credentials)
Default value is 1.

needunregister

(boolean)
Set to false to prevent unregister messages to be sent (for example to prevent unregister on web page reload).
Default is true.

unregall

(number)
Set to 1 to unregister all endpoints for the users, not only the current one.
Will send Contact: * with the unregister requests (REGISTER with Expires: 0).
Default is 0.

unregonidle

(number)
Unregister on idle (seconds).
If set to a positive value then the webphone will automatically unregister when the user doesn't user the browser anymore for the specified seconds (no click, mouse move, etc on the browser window).
The webphone will automatically re-register again on user activity.
If you set it to 1 then it will be treated specially by the NS engine only, unregistering immediately on idle. If you wish to check the browser activity instead, then you should set it to 2 or higher.
Default is 0.

preferred_storage

(number)
Specify what type of web storage should the webphone use.
Possible values:
-1=Auto(default)
0=None (no persistent storage)
1=IndexedDB
2=localStorage
3=WebSQL

The webphone might need to store various things for proper functionality. For example: remember old user settings so the user don't have to type it again, remember old decisions (for example the VoIP engine to use) so it will not spend time next time to lookup or to remember if certain notifications was already showed for the user.
It is highly recommended to keep this value at -1 (auto) as each browser has different capabilities which is handled automatically if the preferred_storage is -1.

Notes:

- *The stored data is always encrypted.*
- *This parameter can be set in the webphone_config.js only and not passed via the setparameter() API.*
- *The webphone might use other storage method, if the preferred one is not supported by the browser.*
- *Storage can also failback to cookie (only for the most important settings).*
  This can be enabled/disabled with the storagecookiefailback parameter: 0 means no,1 means yes (default 1)

resetsettings

(boolean)
Set to true to clear all previously stored or cached settings on startup.
This will reset all settings at every start. You might use the configversion parameter instead if you wish to reset only once (for example when you made some major changes).
Warning: this will forget all settings, even those that should be remembered between sessions, such as "Have we already asked X from the enduser?". To deal with different settings we recommend using the profiles instead or just set unneeded settings to NULL. More details can be found here.
Default is false.

configversion

(number)
Set to a positive value if you wish to reset the config (clear old cached config) on webphone start on configversion change.
This will reset the settings only once (not at every startup).
If already set and you wish to reset the settings again, then just increment the number.
Warning: this will forget all settings, even those that should be remembered between sessions, such as "Have we already asked X from the enduser?". To deal with different settings we recommend using the profiles instead or just set unneeded settings to NULL. More details can be found here.
Default is false.

profile

(string)
By default all settings are remembered by the webphone depending on your html document full URI (full path).
However if you specify a value for this profile parameter, then the settings will be stored in a storage named root path + profile name.

For example if you deploy a webphone to https://domain.com/path1 and a separate webphone to https://domain.com/path2 then the settings of these two webphones will be stored separately.
If you set the profile parameter to "myprofile1" for both webphones, then they will use the same settings storage (at domain.com_myprofile1).

You can also switch the settings with this parameter. For example if your webphone is hosted at https://domain.com/pathX then you might create two separate profile. Sometime you might set the profile to “profileA” and sometime to “profileB”. This way you can use the webphone with unrelated settings and you can be sure that the settings will never mismatch.

Default value is empty (which means settings bound to the webphone path or as specified by the usepathinfilenames parameter).

usepathinfilenames

(number)
Specify where to save the internal settings and cache.
0: don't use any path
1: bind settings just to domain
2: bind settings to whole path
3: just path without domain (default)
Note: this parameter can be set only in the webphone_config.js and can’t be changed by the API at runtime.

extra webrtc options

Among the mentioned ice/turn/stun related settings, you can now specify the following additional webrtc options for the peer connection object:

- bundlePolicy
- iceCandidatePoolSize
- iceTransportPolicy
- rtcpMuxPolicy.

More details can be found here.

loglevel

(number)
Trace/debug level. Values from 1 to 5.
Log level 5 means a full log including SIP signaling. Higher log levels should be avoided, because they can slow down the softphone.
Loglevel above 9 is meant only for Mizutech developers and might slow down the webphone (includes also RTP packets).
Do not set to 0 because that will disable also the important notifications presented for the users.
Recommended values:

- 1: minimal logs for production
- 5: detailed logs for tests

More details about logs can be found here.
There is also a maxloglevel parameter which you can use to prevent users to set the log level above this predefined value.

By default the demo and trial versions might ship with loglevel 5 and the licensed versions with loglevel 1 (which you can overwrite with this parameter).

logtoconsole

(boolean)
Specify whether to send logs to console.
true: will output all logs to console (default)
false: will output only level 1 (important events also displayed for the user)
The amount of logs depends on the “loglevel” parameter.
Default is: true

NS and Java extra settings

With the NS and Java engines you can also use any parameters supported by the Mizu JVoIP SDK as listed in the JVoIP documentation.
(Unrecognized parameters will be skipped if the WebRTC engine is used)

Call divert and other settings

These parameters are used for call auto-answer, forward, transfer, voicemail, number rewrite and similar tasks:

callto

(string)
The webphone can initiate call on startup if this is set. It can be used to implement click to call or similar functionality.
Can be any phone number, username, extension number or full SIP URI acceptable by your VoIP server.
Default value is empty.

autoaction

(number)

Useful for click-to-call to specify what to do if you pass the “callto” parameter

0: Nothing (do nothing, just preset the destination number; the user will have to initiate the call/chat)

1: call (default. Will auto start the call to “callto” –auto-call)

2: chat (will show the chat user interface presenting a chat session with “callto”)

3: video call (will auto start a vide call)

Default is 0.

Note:

- *You can achieve the same (and more flexible) behavior also via the API. For example just initiate a call (by using the Call function) once the webphone is registered.*
- *Some more advanced SIP settings can be found in the “Engine related settings” below (such as dtmfmode, transport or codec).*
- *Sending a chat message can be triggered by the chatto/sendchat URI query parameters*

voicemailnum

(string)
Specify the voicemail number (which the user can call to hear its own voicemails) if any.
Most PBX servers will automatically send the voicemail access number so usually this is detected automatically.
Default value is empty (auto-detect).

normalizenumber

(number)

Normalize called telephone numbers.

If the dialed number looks like a phone number (at least 5 number digits and no a-z, A-Z or @ characters and length between 5 and 20) then will drop all special characters leaving only valid digits (numbers, *, # and + at the beginning).

Possible values:

-1: auto (usually defaults to 1 yes, except if our username also contains special characters)
0: no, don’t normalize

1: yes, normalize (default)

Default is -1

techprefix

(string)

Add any prefix for the called numbers.

Default is empty.

numpxrewrite

(string)

Called number prefix rewrite.

In case if you need to rewrite numbers after your dial plan on the client side, you can use the numpxrewrite parameter (although these kind of number rewrite are usually done after server side dial plan):

You can set multiple rules separated by semicolon.

Each rule has 4 parameters, separated by comma: prefix to rewrite, rewrite to, min length, max length

For example:

‘74,004074,8,10;+,001,7,14;',

This will rewrite the 74 prefix in all numbers to 004074 if the number length is between 8 and 10.

Also it will rewrite the + prefix in all numbers to 001 if the number length is between 7 and 14.

Note: this parameter is same with numrewriterules or the old filters parameter

blacklist

(string)
Block incoming communication (call, chat and others) from these users. (username/numbers/extensions separated by comma).
Default value is empty.

callforwardonbusy

(string)
Specify a number where incoming calls should be forwarded when the user is already in a call. (Otherwise the new call alert will be displayed for the user or a message will be sent on the JS API)
Default is empty.

callforwardonnoanswer

(string)
Forward incoming calls to this number if not accepted or rejected within 15 seconds. You can modify this default 15 second timeout with the callforwardonnoanswertimeout setting.

Default is empty.

callforwardalways

(string)
Specify a number where ALL incoming calls should be forwarded.
Default is empty.

calltransferalways

(string)
Specify a number where ALL incoming calls should be transferred to.
This might be used if your server doesn't support call forward (302 answers) otherwise better to set this on server side because the call will not reach the webphone when it is offline/closed, so no chance for it to forward the call.
Default is empty.

autoignore

(number)
Set to ignore all incoming calls (ignore or reject incoming calls).
0: don't ignore (handle incoming also calls normally)
1: silently ignore (this will not reject the calls, but they will be ignored/muted)
2: reject (auto reject; the caller party will receive a reject/disconnect code)
Default value is 0.

Note: if you don't need incoming calls, then you might also disable registration (set the register parameter to 0), but only if your SIP server doesn't require register to allow also outgoing calls.

autoaccept

(boolean)
Set to true to automatically accept all incoming calls (auto answer / auto connect).
Default value is false.

Note:
You can also auto-answer calls by using the accept API function.
The calls are also auto accepted if the received INVITE contains the "x-p-auto-answer: normal" or "x-answer-mode: auto" SIP header and the enableautoaccept parameter is set to 2.
These are useful if only certain calls have to be auto accepted and you wish to control which one from your JS code or from your SIP server side.
If set to true, then you might also disable the ring tone by setting the playring parameter to 0 as described also here.

With the WebRTC engine some user interaction (such as a button click) might be required before the webphone will be capable to auto answer incoming calls as new browser versions doesn't allow auto-play without user interaction.

enableautoaccept

(number)
Specify if to enable auto-answer suggestions from peers for the incoming calls.
Possible values:
0: never
1: enable only if the autoacceptheader is set and it has a match in the incoming call INVITE message
2: enable also for server-side initiated auto answer (if headers such as Call-Info, Alert-Info, Answer-Mode or Auto-Answer are received with the incoming INVITE)
3: force auto answer: auto connect all incoming calls (works the same way like autoaccept set to true)
Default is 1.

Note: if set to 3, then you might also disable the ring tone by setting the playring parameter to 0 as described also here.

acceptcall_onsharedevice

(number)
Specify whether to auto accept incoming calls when the user clicks to enable device sharing for WebRTC (the audio device permission browser popup) on media permission accepted.
Possible values:
0: no (Do nothing. The user will have to click the "Accept" button to accept the incoming call or you must call the accept() API)
1: auto (guess when to auto accept)
2: always (Always accept webrtc call on browser share device click)
Default is 1.

beeponincoming

(number)
Will play a short sound on incoming calls.
0: No
1: Yes
Default value is 0
Note: this is not the ringtone.

beeponconnect

(number)
Will play a short sound when calls are connected
0: Disabled
1: For auto accepted incoming calls
2: For incoming calls
3: For outgoing calls
4: For all calls
Default value is 0

redialonfail

(number)
Retry the call on failure or no response.
0: no
1: yes
Default value is 1.

rejectonbusy

(boolean)
Set to true to automatically reject (disconnect) incoming call if a call is already in progress.
Default value is false.

allowcallredirect

(number)

Set to 1 to auto-redial on 301/302 call forward.

Set to 0 to disable auto call forward.

Default value is 1.

holdtype

(number)

Specify how the call hold function should work:
-2=no (will ignore hold requests)
-1=auto (defaults to 2)
0=no
1=not used
2=hold (standard hold by sending “a=sendonly”)
3=other party hold (will send “a=recvonly”)
4=both in hold (will send “a=inactive”)

Default is -1

defmute

(number)

Default mute direction:

0: both

1: mute out (speakers)

2: mute in (microphone)

3: both

4: both

5: disable mute

Default: 0 or 2

muteholdalllines

(number)

Auto Mute/Hold all call legs on conference calls.

0=no

1=yes

Default is 0.

automute

(number)

Specify if other lines will be muted on new call.

0=no (default)

1=on incoming call

2=on outgoing call

3=on incoming and outgoing calls

4= on line change (with setline or with other line button click if you are using the softphone user interface)

5=on outgoing call and on line change (like 2+4)

6=on any call and on line change (like 3+4)

Default is 0

autohold

(number)

Specify if other lines will be put on hold on new call.

-1=auto (usually no, except in NS engine transfer)
0=no
1=on incoming call
2=on outgoing call
3=on incoming and outgoing calls
4=on line change (with setline or with other line button click if you are using the softphone user interface)
5=on outgoing call and on line change (like 2+4)
6=on any call and on line change (like 3+4)

Default is -1

multilineoop

(number)
Specify hold-mute workaround for the WebRTC engine with multiple simultaneous calls.
With the WebRTC engine the webphone might use mute instead of hold with multiple lines as a workaround for some multi-line hold related WebRTC issues in some browsers.
-1: always use mute instead of hold (even if there are no multiple calls)
0: use mute instead of hold with multiple calls
1: usually use mute with multiple calls
2: use mute instead only in some cases for multi-line calls
3: never use mute instead of hold
Default is 2

audiodevicein

(string)
Audio device name for recording (microphone). Set to a valid device name or "Default" which would select the system default audio device.
Note: this is usually set at run-time from the API or from an audio device select control presented to the users (already implemented by the softphone skin). If not set, then the webphone will use the OS default recording device.

audiodeviceout

(string)
Audio device name for playback (speaker). Set to a valid device name or "Default" which would select the system default audio device.
Note: this is usually set at run-time from the API or from an audio device select control presented to the users (already implemented by the softphone skin). If not set, then the webphone will use the OS default playback device.

audiodevicering

(string)
Audio device name for ringtone. Set to a valid device name or "Default" which would select the system default audio device. You can also set it to "All" to have the ringtone played on all devices.
Note: this is usually set at run-time from the API or from an audio device select control presented to the users (already implemented by the softphone skin). If not set, then the webphone will use the OS default playback device. Separate ringer device selection is not available with the WebRTC engine.

volumein

(number)
Default microphone volume in percent from 0 to 100. 0 means muted. 100 means maximum volume.
This is the audio input volume which are going to be streamed to the other end.
Default is 50% (not changed)

Note:

- The default volume level is determined by the OS volume setting. The webphone volume changes will be only relative to the OS settings
- The result volume level might be affected by the AGC if it is enabled.
- Some browsers doesn't expose volume change functionality if you are using the WebRTC engine

volumeout

(number)
Default speaker volume in percent from 0 to 100. 0 means muted. 100 means maximum volume.
This is the audio output volume heard by the local user from the incoming audio stream sent by the other party.
Default is 50% (not changed)

Note:

- The default volume level is determined by the OS volume setting. The webphone volume changes will be only relative to the OS settings
- The result volume level might be affected by the AGC if it is enabled.
- Some browsers doesn't expose volume change functionality if you are using the WebRTC engine

volumering

(number)
Default ringback volume in percent from 0 to 100. 0 means muted. 100 means maximum volume, 0 is muted.
Default is 50% (not changed)

Note: the ring volume can't be changed for the WebRTC engine (use the OS volume settings)

androidspeaker

(number)
Control the default playback device on Android phones and tablets.
-2: ignore (don't make any changes)
-1: auto guess (prefer loudspeaker for video call, otherwise speakerphone)
0: default (most Android smartphones will use the loudspeaker by default as the playback device)
1: switch to speakerphone before the first call
2: switch to speakerphone after first call
3: switch to speakerphone at start

Default: -1

Note:
You can also use the setloudspeaker API to change it at runtime.
There is no similar setting for iOS, because changing an audio device requires "setSinkId()" which is not supported even by the latest versions of Safari.

transfertype

(number)
Specify transfer mode for native SIP.
-1: default transfer type
0: call transfer is disabled
1: transfer immediately and disconnect with the A user when the Transf button is pressed and the number entered (unattended/blind transfer)
2: transfer the call only when the second party is disconnected (attended transfer)
3: transfer the call when the VoIP phone is disconnected from the second party (attended transfer)
4: transfer the call when any party is disconnected except when the original caller was initiated the disconnect (attended transfer)
5: transfer the call when the VoIP phone is disconnected from the second party. Put the caller on hold during the call transfer (standard attended transfer)
6: transfer the call immediately with hold and watch for notifications (unattended transfer)
7: transfer with no hold and no disconnect (simple unattended transfer)
8: transfer with conference (will put the parties to conference on transfer; will mute or hold the old party by default)

Default is -1 (which is the same as 6 or 7)

Note:

- *Unattended means simple immediate transfer (just a REFER message sent)*
- *Attended transfer means that there will be a consultation call first. The softphone skin has its own attended transfer control options.*
- *It is also possible to create an attended transfer by initiating a simple call to the transfer target and on hangup an unattended call-transfer (transfertype 1, 6 or 7)*
- *If you have any incompatibility issue, then set to 7 (unattended is the simplest way to transfer a call and all sip server and device should support it correctly)*
- *More details can be found here*

transfwithreplace

(number)
Specify if replace should be used with transfer requests, so the old call (dialog) is not disconnected but just replaced.
This way the transferee and the transfer-target parties are not disconnected, just the other party is changed at runtime.
The feature is implemented by adding a Replaces= for the Contact header in the REFER request as described in RFC 5589 and RFC 5359.
The transferee (the party which receives the REFER transfer request) or the SIP server must be able to handle replaces for this to work.
Possible values:
-1: auto (if transfertype is -1 (default), 6 or 7; server/peer has replaces support; we are connected with the transfer target)
0: no
1: yes
2: force (use replaces even if peer doesn't have replaces support or there is no session with the target)
Default is -1

Note: some SIP servers doesn't announce replace capabilities ("replace" tag into the Supported or Require SIP headers). In this can you will have to set the transfwithreplace parameter to 2.

replacetype

(number)
Specify how to handle incoming call transfer with replaces requests (How to handle the Replaces= Contact tag in incoming REFER requests)
-1: auto (defaults to 1)
0: disable
1: standard (sending the Replaces header in the new call INVITE request as described in RFC 3891)
2: in place (might be useful with servers without replaces support; not supported by the WebRTC engine)
Default is -1

changesptoring

(number)
If to treat session progress (183) responses as ringing (180). This is useful because some servers never sends the ringing message, only a session progress and might not start to send in-band ringing (or some announcement). In this circumstances the webphone can generate local ringback.
The following values are defined:
0: do nothing (no ringback on session progress message)
Will not call startRingbackTone() on 183 (only for 180)
1: change status to ring
2: start local ring if needed and be ready to accept media (which is usually a ringtone or announcement and will stop the locally generated ringback once media received)
Will call startRingbackTone() on 180 and 183 but stop on early media receive.
3: start media receive and playback (and media recording if the "earlymedia" parameter is set)
4: change status to ringing and start media receive and playback (and media recording if the "earlymedia" parameter is set to true)
5: play early ringback and don't stop even if incoming early media starts
Will call startRingbackTone() on 180 and 183 and do NOT stop on early media receive.
Default value is 2.

**Note: on ringing status the web phone is able to generate local ringback tone. However with the default settings this locally generated ringtone playback is stopped immediately when media is started to be received from the server (allowing the user to hear the server ringback tone or announcements)*

ringtimeout

(number)
Maximum ring time allowed in millisecond.
Default is 90000 (90 second)

You can also set separate ring timeout for incoming and outgoing calls with the "ringtimeoutin" and "ringtimeoutout" settings.

calltimeout

(number)
Maximum speech time allowed in millisecond.
Default is 10800000 (3 hours)

mediatimeout

(number)

RTP timeout in seconds to protect again dead sessions.
Calls will be disconnected if no media packet is sent and received for this interval.
You might increase the value if you expect long call hold or one way audio periods.
Set to 0 to disable call cut off on no media.
Default value is 300 (5 minute).
At the beginning of the calls, the half of the mediatimeout value is applied (2.5 minute by default if there was no incoming audio at all).

voicerecupload

(string)
Voice record upload URL (FTP or HTTP).
With this setting you can setup VoIP call recording (voice recording).
Default value is empty (no voice call recording).
If set then calls will be recorded and uploaded to the specified ftp or http address in pcm/wave, gsm, mp3 or ogg format.

Note: mp3 support is not included by default in the NS engine to minimize the package size. Contact Mizutech to include this in your build if you need mp3 recording also from the NS engine (build option voicerecformat=3;)

The files can be uploaded to

- your FTP server (any FTP server with specified user login credentials)
- or to your HTTP server using HTTP PUT or multipart/form-data POST (in this case you need a server side script to save the uploaded data to file)

The following keywords can be used in the file name as these will be replaced automatically at runtime to their respective values:

- DATETIME: will be replaced to current date-time
- DATE: will be replaced to current date (year/month/day)
- TIME: will be replaced to current time (hour/min/sec)
- CALLID: will be replaced to sip call-id
- USER: will be replaced to local user name
- CALLER: will be replaced to caller party name (caller id)
- CALLED: will be replaced to callee party name
- SERVER: the domain or IP of the SIP server
- COUNTER: an auto-increasing number

If you set a HTTP URI, then the following headers will be also set in the HTTP PUT or POST: X-type, X-filename, X-user, X-caller, X-called, X-callid and X-server.
We recommend to use FTP first (or try this first) as this is very easy to configure and doesn't require any server side script.

If you need unique file names then we recommend using the SIP Cal-ID (CALLID keyword), so you can associate your call detail records (CDR's obtained from your VoIP server) with the recorded files. You can also access the SIP Call-ID from the webphone in many ways: use the onCallStateChange or onCdr callbacks or use the linetocallid function to get the call-id of a call.

FTP Example:
ftp://user01:pass1234@ftp.foo.com/voice_DATETIME_CALLER_CALLED

You can also suggest a particular file format by appending its extension to the file name (for example .wav or .mp3).
For example: ftp://user01:pass1234@ftp.foo.com/voice_DATETIME_CALLER_CALLED.wav

Since the username:password is part of the URI here, no special characters are allowed in the file path (don't use ftp accounts with characters like @ ; ' " / \ in the username or in the password).

HTTP Example:
http://www.foo.com/myfilehandler.php/filename=callrecord_CALLID

You can also suggest a particular file format by appending its extension to the file name (for example .wav or .mp3).
For example: http://www.foo.com/myfilehandler.php?filename=callrecord_DATETIME_USER.wav

You can also set just a path, without any URL query parameter, like this:
For example: http://www.foo.com/myfilehandler.php/rec_DATETIME_CALLID

Example HTTP POST header packet if you set the url to “http://yourdomain.com/myapi/voicerecord/anyentry?filename=callrecord_DATE_CALLID.mp3”:

```
POST /myapi/voicerecord/anyentry?filename=callrecord_2026_04_04_xxx.mp3 HTTP/1.1
Host: yourdomain.com
User-Agent: webphone
Accept: */*
X-type: fileupload
X-filename: callrecord_2026_04_04_xxx.mp3
X-user: local_username
X-caller: caller_party
X-called: called_party
X-callid: sip_callid
X-server: your_sip_server_address
Content-Length: 18623
Expect: 100-continue
Content-Type: multipart/form-data; boundary=--------------------------fde9999399e1b8eb

--------------------------fde9999399e1b8eb
Content-Disposition: form-data; name="file"; filename="callrecord_2026_04_04_xxx.mp3"
Content-Type: application/octet-stream
.....
```

You will receive “file” as the form name parameter and the name of the file as the form data “filename” parameter.
(So you will receive the file name as both the “filename” form parameter and also in the X-filename HTTP header).
The content type can be application/octet-stream, audio/x-wav, audio/mpeg, audio/x-gsm or audio/ogg.
(Regardless of the suggested file name, you can save the files on your server with any name, this is your choice).
A working example for PHP can be downloaded from here.
More details about handling HTTP file upload: C#, ASP.NET, PHP, PHP (2), NodeJS, Java.
More details about voice recording can be found here.

Note:

- *You can also use the voicerecord API to turn on/off the voice recording at runtime (if not all calls have to be recorded)*
- *The voicerecord API should be called before the call you wish to record as changing the recording state during the calls might not be supported*
- *With the NS or Java engine you can also record to local file. For this, you need to set the voicerecording parameter which has the following values defined: 0=no, 1=record to local file system, 2=remote http/ftp only, 3=both local and remote*
- *If you are using the WebRTC engine with FTP voice file storage, avoid using special characters in the ftp username/password*
- *Ogg/vorbis format can be also used with NS or Java engines (this is not supported by WebRTC)*
- *With the NS and Java engines the call recording is performed on the client side by the webphone. With the WebRTC engine the call recording is performed on the gateway or server side (but the voicerecupload parameter is still handled correctly, passing it to the server)*
- *If you are using the NS engine with built-in JVM then HTTPS upload might not work (old tiny version of the JVM might not implement the latest TLS protocols). In this case you can still use HTTP or FTP storage (not HTTPS) or notify Mizutech to build with a new full JVM (will increase the size).*

User interface related settings

Most of these apply only to the Softphone user interface which is shipped with the webphone to further customize the web softphone user interface and behavior (Softphone.html)

brandname

(string)
Brand name of the softphone to be displayed as the title and at various other places such as SIP headers such as the User-Agent.
Default is empty.

Note: purchased webphones always comes with a predefined brand name as you communicated to Mizutech. This brand must be a short unique name, with no special characters. You can overwrite the default with this setting if needed.

companyname

(string)
Your company name to be displayed in the about box and various other places.
Default is empty.

logo

(string)
Displayed on login page.
Can be text or an image name, ex: "logo.png" (image must be stored in images/folder)
Default is empty.

Note: if you set a logo, then this will be placed on the softphone skin (softphone.html), taking up some space of its user interface.
The useloginpage might be also set to 1 or 2 to show the logo.
Since the webphone is usually embedded in webpages, you might consider to place logo on other places of your website and not into the softphone skin itself as in this case it would just take precious space inside the app.
Otherwise, you also have all the html in your hand so you can edit the user interface as you wish, placing any image at any location after your like.

colortheme

(number)
You can easily change the skin of the supplied user interfaces with this setting (softphone, click to call).
Possible values:

1. Default
2. Light Blue
3. Light Green
4. Light Orange
5. Light Purple
6. Dark Red
7. Yellow
8. Blue
9. Purple
10. Turquoise
11. Light Skin
12. Green Orange

Default is 0.
More details about design changes.

language

Set the language for the user interface.
This is usually a two character language code (for example en for English or pt for Portuguese) or for specific accents/countries you can use the long format such as en-US.
If this parameter is not set, then by default the webphone will auto detect the browser language and will use it if there is a good translation for that language.
This behavior can be turned off by setting the languageautodetect parameter to 0 (will always default to original/English in this case).

Before you preset any language, make sure that the webphone has a good quality built-in translation for it.
Otherwise, you might need to add the translation yourself or improve the existing translation.
See the details here.

featureset

(number)
User interface complexity level.

0=minimal
5=reduced
10=full (default)
15=more (for tech user)

You might set to 5 for novice users or if only basic call features have to be used.
Default is 10.

showserverinput

(number)
This can be used to hide the server address setting from the user if you already preconfigured the server address in the webphone_config.js (“serveraddress” config option), so the enduser have to type only their username/password to use the softphone.
Possible values:

0: no (will hide the server input setting for the endusers)

1: auto (default)
2: yes (will shot the server input setting for the endusers)

showproxyaddress

(number)
Specify if to show proxy server input on the softphone user interface login page.
Possible values:
0: no
1: if set
2: yes
Default is 1

showusername

(number)
Specify if to show the username (caller id) input on the softphone user interface login page.
Possible values:
0: no
1: if set
2: yes
Default is 1

handleusernameuri

(number)
Specify how to handle if the user type a full SIP URI as the username input.
Possible values:
0: always ignore (will ignore also for the username; not recommended)
1: ignore for the sipusername (auth username)
2: extract username only in basic settings
3: config as serveraddress if missing in basic settings
4: config as serveraddress if missing
5: config as serveraddress always
6: config as proxyaddress always
Default is 3

handlesipusernameuri

(number)
Specify how to handle if the user type a full SIP URI as the sipusername input.
-1: auto (keep it as-is but might retry the auth with the user part only if rejected)
0: keep it as-is (the sip auth username can have user@domain format)
1: remove the domain part and keep only the SIP URI

Default is -1.

This parameter might be ignored if the handleusernameuri is set to 0.
If you (your VoIP server) are using sip auth username in user@domain format, then you might set this parameter to 0 (to avoid unneccessary auth retry with the default -1).
Otherwise (if you don't use user@domain format for sip authnetication), then you might set it to 1 (to fix the user@domain user input and keep only the user part for SIP auth).
Note: the old allowsipuriasusername parameter was deprecated by this new one.

useloginpage

(number)

Whether to use a simplified login page with username/password in the middle (instead of list style settings; old haveloginpage).
Possible values:
-1: auto (will auto set to 1 if featureset is Minimal, otherwise 0)
0: no
1: only at first login
2: always
Default is: -1

autologin

(number)
Specify when to skip the login page.
Possible values:
-1=Auto
0=No
1=Yes
Default is: -1

More details can be found here.

hidepassword

(number)
Specify if to hide the password string on the softphone user interface (login page / settings).
Whether to use a simplified login page with username/password in the middle (instead of list style settings; old haveloginpage).
Possible values:
0: don't hide
1: replace text element with ***** if the password is already set
2: use password input element if the password is already set (instead of text)
3: always use password input element (instead of text)
4: always use password input element (instead of text) and disable password reveal if the password was already set
Default is 4.
Note: When set to 2 or 3, the password string can be revealed by double-click.

pwdautocomplete

(number)
Enable/disable password autocomplete on the softphone user interface (login page / settings)
Possible values:
-1: don't set any flag (use browser default)
0: disable (off)
1: enable (on)
Default is -1

quick_access_list

(number)
Enable/disable recent important contact list on the main page.
Possible values:
0: disable (show only the dial pad)
1: enable
Default is: 1

textmessaging

(number)
Specify text messaging mode (IM/chat/SMS/API)
-1: auto guess or ask (default)

0: disable all
1: disable incoming messages and auto guess outgoing mode
2: disable message sending and auto guess incoming mode
3: API (if SMS API URI have been defined)
4: reserved
5: VoIP SMS (will send the X-Sms: yes SIP header. More details here)
6: VoIP IM/chat (SIP MESSAGE)

Note: the old haschat and chatsms parameters are deprecated now but still supported

showincomingchatas

(number)
Define how to handle incoming chat messages.
0: open/show chat window if not in call
1: just set a notification
Default is 0.

callparknumber

(string)
Can be used to add call park and call pickup (will be sent as DTMF for call park and user need to call to pickup number to later reload the call from the same or other device).
If set, then it will be displayed on the call page as an extra option.

hasringcounter

(boolean)
Enable/disable the time counter during ring-time.

hasfiletransfer

(boolean)
Set to true to enable file transfer.

filetransferurl

(string)
HTTP URI used for file transfer. By default Mizutech service is used which is provided for free with the web softphone.

displaynotification

(number)
Show notifications in phone notification bar (usually on the top corner of your phone).
0:No
1:Yes (display missed call and missed chat notifications)
Default is 1.

displayvolumecontrols

(boolean)
Always display volume controls when in call.
Default is false.

Old/deprecated parameter name was "hasvolume".

displayaudiodevice

(boolean)
Always display audio device when in call.
Default is false.

displayvideodevice

Display video preview window on video device select:
Possible values: 0 mean No (default), 1 means Yes
When set to 1, then a video preview will appear when you select a video device on the device form or via the devicepopup() API. This preview window can be closed by the user or it will be closed automatically after 10 seconds once the device form is closed.

displaypeerdetails

(string)
Specify where to display the information returned by scurl displaypeerdetails.
It can be used display details about the peers from your CRM such as full name, address or other details.
(Useful in call-centers and for similar usage)
Possible values:
0: show on call page (instead of contact picture)
1: on new page
div id: display on the specified DIV element

savetocontacts

(number)
Whether to (automatically) add new called numbers to your contact list.
0:No
1:Ask
2:Yes (will not ask for a contact name)
Default is 1.

messagepopup

(string)
Display custom popup for user once.
Default is empty.

showtoasts

(boolean)

Enable/disable hint popups.
Possible values:
true: show all toasts (default)
false: don't show toasts and popup messages

incomingcallpopup

(number)
Whether to display a popup about incoming calls (uses the HTML5 notification API for WebRTC or native popup for NS/Java).
Possible values:

0. No
1. Auto (show only if browser window is not in foreground/focused)
2. Yes
3. Always force from all engines

Default is 1.

Set to 0 to disable (in this case make sure that you handle the incoming call alert from your HTML/JS if required).
Set the callreceiver parameter to 2 to catch incoming calls even when the webphone is not running.

Note: The old name of this parameter name was hasincomingcallpopup and hasincomingcall

The incoming call HTML5 popup can be optionally customized with the following parameters:

- callnot_title (string) – the title of the notification. By default this is the brand name of the webphone.
- callnot_body (string) – the body text of the notification. By default this is: “Incoming call from: PEERNAME”
- callnot_icon (string) – URL of the notification icon. By default this is: “notification_icon.png” from “images” directory.
- callnot_image (string) – URL of the notification image. By default this is empty.

asknotifpermission

(number)
Specify when to ask for permissions if HTML5 notifications are required:
-1: auto
0: never
1: when required
2: when required and on startup/call if failed
3: on start and on call (always)

closecall_timeout

(number)
The call page will remain active after successful call hangup for this amount of time in milliseconds, after which it will close automatically.
Set to 0 to disable.
Default is 3000 (3 sec)
Note: if between 1 and 100, then it is treated as seconds (instead of milliseconds)

closecall_timeout_err

(number)
The call page will remain active after failed call hangup for this amount of time in milliseconds, after which it will close automatically.
Set to 0 to disable.
Default is 7000 (7 sec)
Note: if between 1 and 100, then it is treated as seconds (instead of milliseconds)

header

(string)
Header text displayed for users on top of softphone windows.
Default is empty.

footer

(string)
Footer text displayed for users on the bottom of softphone windows.
Default is empty.

version

(string)
Version number displayed for users.
Default is empty (will load the built-in version number)

showsynccontactsmenu

(number)
This is to allow contact synchronization between mobile and desktop.
-1=don't show
0=show Sync option in menu and Contacts page (if no contacts available)
1=show in menu only
Default is 1

defcontacts

(string)
Set one or more contacts to be displayed by default in the contact list.
Contact fields: name, number, email, address, notes, website
Fields must be separated by comma and contacts separated by semicolon:
Example: defcontacts: 'John Doe,12121;Jill Doe,231231,jill@g.com'

Note: Contacts can be added also at runtime using the addcontact API.

disableoptions

(string)
List of settings options and features to be disabled or hidden.
To disable entire features, use the upper case keywords such as CHAT,VIDEO,VOICEMAIL,CONFERENCE.
To disable settings, use the setting label or name such as Audio device, Call forward.
Example: disableoptions: 'theme,email,Call forward,callforwardonbusy,callforwardonnoanswer,callforwardalways,VIDEO'

Other examples:
"chatsms,textmessaging,disablewbforpstn,audiorecorder,audioplayer,speakerphoneplayer,useroutingapi,callback_mode,transfertype,transfwithreplace,balance_uri,rating_uri,creditrequest,ratingrequest,p2p_uri,p2p,callback_uri,callback,sms_uri,sms, disablecontactmenu";
You can also disable main pages: PAGE_MAIN, PAGE_CONTACTS, PAGE_HISTORY, RECENTS

extraoption

(string)
Custom parameters can be set in a key-value pair list, separated by semicolon Ex: displayname=John;
Default is empty.

logsendto

(number)
Specify allowed actions on the logs page.
0: no options (users will still be able to copy-paste the logs)
1: upload (default)
2: email launch (the email address set by the “supportmail” parameter or support@mizu-voip.com if not set)

links

(strings)
The webphone GUI can load additional information from your web server application or display some content from your website internally in a WebView or frame. You can integrate the included softphone user interface with your website and/or VoIP server HTTP API (if any) by using the following parameters:

- **advertisement:** Advertisement URL, displayed on bottom of the softphone windows.
- **supportmail:** Company support email address.
- **supporturl:** Company support URL.
- **newuser:** New user registration http request OR link (if API then suffix with star *)
- **forgotpasswordurl:** Will be displayed on login page if set.
- **homepage:** Company home page link.
- **accounturi:** Company user account page link.
- **recharge:** Recharge http request (pin code must be sent) or link.
- **p2p:** Phone to phone http request or link.
- **callback:** Callback http request or link (For example: *https://yourdomain.com/callback?user=USERNAME)
- **sms:** SMS http request.
- **creditrequest:** Balance http request, result displayed to user. (For example: *https://yourdomain.com/balance?user=USERNAME)
- **ratingrequest:** Rating http request, result displayed for user on call page. (For example: *http://yourdomain.com/rating?destination=CALLEDNUMBER)
- **helpurl:** Company help link.
- **licenseurl:** License agreement link.
- **extramenuurl:** Link specifying custom menu entry. Will be added to main page (dialpad) menu.
- **extramenutxt:** Title of custom menu entry. Will be added to main page (dialpad) menu.

- **serveraddressbook_url:** address book URL or API (details here)

Parameters can be treated as **API requests** (specially interpreted) or **links** (to be opened in built-in webview). For http API request the value must begin with asterisk character: "*http://domain.com/...." For example if the "newuser" is a link, then it will be opened in a browser page; if it's an API http request (begins with *), then a form will be opened in the softphone with fields to be completed.

- The followings are always treated as API request: creditrequest, ratingrequest
- The followings can be links OR API http requests: newuser, recharge, p2p, callback, sms
- The rest will be treated always as links (opened in built-in webview or separate browser tab)

You can also use keywords in these settings strings which will be replaced automatically by the web softphone. The following keywords are recognized:

- DEVICEID: unique identifier for the client device or browser
- SESSIONID: session identifier
- USERNAME: sip account username. preconfigured or entered by the user
- PASSWORD: sip account password
- CALLEDNUMBER: dialed number
- PEERNUM: other party phone number or SIP uri
- PEERDETAILS: other party display name and other available details
- DIRECTION: 1=outgoing call, 2=incoming call
- CALLBACKNR,PHONE1,PHONE2: reserved
- PINCODE: reserved. will be used in some kind of requests such as recharge
- TEXT: such as chat message
- STATUS: status messages: onAppStateChange, onRegStateChange, onCallStateChange, inChat, outChat
- MD5SIMPLE: md5 (pUser + ":" + pPassword)
- MD5NORMAL: md5 (pUser + ":" + pPassword+":"+randomSalt)
- MD5SALT: random salt

Example credit http request: https://domain.com/balance/?user= USERNAME
(Where “USERNAME” will be dynamically replaced with the currently logged in username)

Note: Ensure that you have proper authentication for payable functions such as callback, p2p or sms. Some other functions can also affect user privacy, such as creditrequest.

Parameter security

The webphone is secure by default.

From security perspective the webphone can be treated as any other regular SIP endpoint: make sure to not reveal the password of your SIP accounts and your app will be secure.
Webphone parameters are safe by default since they are used only in the user http session. This means that the enduser can discover its own settings including the password, but other users –including users for the same browser or middle-men such as the ISP- will not be able to see the sensitive parameters if you are using secure http (HTTPS) and don’t statically set a SIP account password in your HTML, which could be discovered from your webpage source.

The only sensitive parameter is the SIP account “password”! This is sent only as digest hash in signaling, but make sure to never display or log from your code. The easiest way to maintain maximum security is to allow the endusers to type their SIP password as needed, instead of hardcoding or provisioning.

You should not hardcode the password into your website (It should not be found if you check the source of your webpage in the browser. The only exception would be if you offer some free to call service which is not routed to outside paid trunks/carriers). If the password has to be preconfigured then load it via an ajax call or similar method; just make sure to use HTTPS in this case because otherwise all the communication is in clear text between the browser and your server if the page is running on unsecure HTTP. Otherwise just let the endusers to enter their password on a login/settings form and pass it to the webphone with the setsipheader() API call.

If the password needs to be passed dynamically then:

- make sure to use HTTPS if it is sent from the server (for example via an AJAX or other API request)
- if you pass it from JavaScript, then you might encrypt or obfuscate it with your preferred method

Please note that even if you pass the password around in clear text, you are still protected if your page is in secured (on HTTPS). Only the enduser might be able to found its own password from the browser in this case, but this is usually not a problem since the users should know their own password anyway (Or if somehow you don’t wish the enduser to known its own password then you can implement your own custom encryption or obfuscation in JavaScript)

There is no much reason to try to obfuscate or hide other parameters.

For example the “serveraddress” can be discovered anyway by analyzing the low level network traffic and this is perfectly normal. Most of the other parameters are completely irrelevant. Some sensitive information’s are also managed by the webphone (such as the user contact list) however these are stored only locally in the browser secure web storage or secure cookie by default (on HTTPS) and further encrypted or obfuscated by the webphone.

The following methods can be used to further secure the webphone usage:

- set the loglevel to 1 (with loglevel 5 the password might be written in the logs)
- use strong passwords for all your SIP extensions or trunks
- don’t hardcode the password if possible (let the users to enter it) or if you must hardcode it then use encryption and/or obfuscation
- restrict the account on the VoIP server (for example if the webphone is used as a support access, then allow to call only your support numbers and disable outbound calls)
- set a maximum simultaneous call limit and rate limiter for your SIP accounts
- instead of preconfigured parameters you can use the javascript VoIP api (setparameter)
- use HTTPS (secure http / TLS)
- encrypt/obfuscate the password in JavaScript code if you wish to hide it even from its owner or if you need to use a preconfigured account for your app
- for parameter encoding (encryption/obfuscation) you can use XOR + base64 with your built-in key (ask passphrase from Mizutech), prefixed with the “encrypted__3__” string (you can verify your encryption with this tool selecting XOR Base64 Encrypt)
- you can quickly encrypt any parameter using this link:
  https://mnt.mizu-voip.com/mmexreqvitserverjrsktt/xwencode?wkey=YOURKEY&wstring=YOURSTRING&wversion=3
  Replace YOURKEY with your webphone key provided by mizutech. The key in the demo version is h39idbfqw7116ghh
  Replace YOURSTRING with a parameter value (for example the password parameter). URI encode if contains special characters such as spaces.
  More details can be found here.
- secure your VoIP server (account limits, rate-limits, balance limits, fraud detection) and follow the VoIP security best practices. For example here you can find details about mizu VoIP server security.

JavaScript API

About

You can use the webphone javascript SIP library in multiple ways for many purposes:

- create your own web dialer
- add click to call functionality to your webpage
- add VoIP capability to your existing web project or website
- integrate with any CRM, callcenter client or other projects
- modify one of the existing projects to achieve your goal (see the included softphone and click to call examples) or create yours from scratch
- and many others

The public JavaScript API can be found in "webphone_api.js" file, under global javascript namespace "webphone_api".
To be able to use the webphone as a javascript VoIP library, just copy the webphone folder to your web project and add the webphone_api.js to your page.
The API functions should not throw exceptions and should not return null or undefined values (will return empty/false/0/minus/'ERROR' values on failure, depending on the context).

In case if you have to manage multiple simultaneous calls explicitly, then you might call the setline() API to specify the line before the other API calls.
For example to mute a call on line 2 first set the line to 2 (webphone_api.setline(2);) then call the mute function (webphone_api.mute();)
More details about multi line management can be found here.

Basic example

```
<head>
        <!-- Include the webphone_api.js to your webpage -->
        <script src="webphone_api.js"></script>
</head>
<body>
<script>
        //Wait until the webphone is loaded, before calling any API functions
        webphone_api.onAppStateChange (function (state)
        {
                if (state === 'loaded')
                {
                        //Set parameters (Replace upper case worlds with your settings)
                        webphone_api.setparameter('serveraddress', SERVERADDRESS);
                        webphone_api.setparameter('username', USERNAME);
                        webphone_api.setparameter('password', PASSWORD);
                        webphone_api.setparameter('other', MYCUSTOMSETTING);
                        //See the “Parameters” section below for more options
```

```
            //Start the webphone (optional but recommended)
            webphone_api.start();

            //Make a call (Usually initiated by user action, such as click on a click to call button. Number can be extension, SIP username, SIP URI or mobile/landline phone)
            webphone_api.call(NUMBER);

            //Hang-up (usually called from "disconnect" button click)
            webphone_api.hangup();

            //Send instant message (Number can be extension, SIP username. Usually called from a "send chat" button)
            webphone_api.sendchat(NUMBER, MESSAGETEXT);
        }
    });
    //You should also handle events from the webphone and change your GUI accordingly (onXXX callbacks)
</script>
</body>
```

See the webphone package for more examples. You should check especially the tech demo (techdemo_example.html / techdemo_example.js).

Note: If you don't have JavaScript/web development experience, you can still fully control and customize the webphone:

- *by its numerous configuration options which can be passed also as URL parameters*
- *from server side as described here*
- *we can also send ready to use fully customized web softphone with preconfigured settings, branding and integration with your web and VoIP server*

More details can be found here.

Functions

Use the following API calls to control the webphone:

setparameter (param, value)

You can use this function to set the webphone parameters (the various settings which are described in the Parameters chapter) dynamically (at run-time) from JavaScript.
(Alternatively you can just hardcode the parameters in the webphone_config.js or pass by URL query parameters)

Example:

```
setparameter('username ', 'john');  //this will set the SIP username to be used for upcoming registration or call
```

Note:
All parameters are saved/cached by the webphone so they will persist (except if you clear the browser storage and/or cookies).
This function also has a third optional allowempty parameter. If set to false then empty values are ignored and you should use 'NULL' instead to clear the previous/cached setting if any. Its default value is true.
You can clear previously set values using the delsettings () API or by using the setparameter with an empty or 'NULL' value. Example: setparameter('displayname', 'NULL');

getparameter (param)

Return type: string
Will return value of a parameter if exists, otherwise will return empty string.

Example:

```
var mystringvariable = getparameter('displayname ');  //this will return the display name if it was set previously
```

start()

Optionally you can "start" the phone, before making any other action.
In some circumstances the initialization procedure might take a few seconds (depending on usable engines) so you can prepare the webphone with this method to avoid any delay when the user really needs to use by pressing the call button for example.
Set the "autostart" parameter to 0 if you wish to use this function. Otherwise the webphone will start automatically on your page load.
If the serveraddress/username/password is already set and auto register is not disabled (not 0), then the webphone will also register (connect) to the SIP server upon start.
If start() is not called, then the webphone will initialize itself the first time when you call some other function such as register() or call().
The webphone parameters should be set before you call this method (preset in the js file or by using the setparameter() function). See the "Parameters" section for details.

stop()

Optionally you can "stop" the webphone engine (but this is done automatically on browser close or refresh, so usually not required).
Calling the stop() API will also unregister.

register ()

Optionally you can "register" if your SIP server has also registrar roles (most of them have this). This will "connect" to the SIP server by sending a REGISTER request and will authenticate if requested by the server (by sending a second REGISTER with the digest authorization details).

Note:

- If the serveraddress/username/password is already set and auto register is not disabled (not 0), then the webphone will register (connect) to the SIP server upon start, so no need to use this function in these circumstances.
- There is no need to call the register() multiple times as the webphone will automatically manage the re-registrations (based on the registerinterval parameter)

registerex (accounts)

You can use this function to register with multiple SIP accounts (multi-account feature).
The accounts are passed as string in the following format:
server,usr,pwd,ival,proxy,realm,authuser,displayname,transport;server2,usr2,pwd2,ival2,proxy2,realm2,authuser2,displayname2,transport2

Multiple accounts can be passed at once separated by semicolons (;).
An account must be specified as the following parameters separated by comma (,) :

0. Server address
1. Username
2. Password
3. Register interval
4. SIP proxy
5. SIP realm
6. Auth user name (if separate extension id and authorization username have to be used)
7. Displayname
8. Transport protocol (default/empty, UDP, TCP, TLS)

The server, username and password parameters are mandatory, all the others are optional. JVoIP will use the defaults for the empty parameters.
All accounts have to be passed in a single line which should look like this: server,usr,pwd,ival;server2,usr2,pwd2,ival2;etc…

Note:

When you call the registerex function, these accounts will be remembered (like you would set them with the extraregisteraccounts parameter) and will be re-used also at next startup. You should pass all the extra accounts at once (don't call this function multiple times). You can clear the old accounts by passing "null" as the accounts parameter.

Examples:

```
webphone_api.registerex ("myserver.com,john,secret"); //add a single extra account as parameters
webphone_api.registerex ("myserver.com:5061,john,secret,180, , , ,John Smith"); // add a single extra account with register interval and displayname
webphone_api.registerex ("92.168.1.50:5060,john,jsecret,180;92.168.1.50:5060,kate,ksecret;92.168.1.50:5060,mary,msecret,600"); //add 3 extra accounts
webphone_api.registerex("null"); //clear old settings
```

For more details read here.

unregister ()

Un-register from your SIP server (will send a REGISTER with Expire header set to 0, which means de-registration).
Unregister is called also automatically at browser close so usually there is no need to call this explicitly.

call (number)

Initiate call to a number, extension, sip username or SIP URI.

Perhaps this is the most important function in the whole webphone API.
It will automatically handle all the details required for call setup (network discover, ICE/STUN/TURN when needed, audio device open and call setup signaling).
With this function you can make calls via your SIP server or SIP service provider:

- *to any SIP endpoint (such as softphone or IP phone)*
- *to any WebRTC endpoint*
- *to pstn/mobile/landline (if your server allows outbound calls. If you have a SIP subscription then most probably you need a positive balance to be able to make pstn calls)*
- *to any SIP server (including IVR or callback access numbers)*

You can also send dtmf messages once the call connected by appending it to the number after a comma. For example if you make a call to 123,456 then it will call 123 and then it will send dtmf 456 once the call is connected.
For video chat use the videocall function mentioned below.

accept ()

Connect incoming call.

hangup ()

Call disconnect (disconnect the current call or the call on the active line)

For incoming not yet connected call, hangup acts like reject.
Notes about line-management (in case if you are implementing a multi-line user interface, otherwise you don't need to deal with line numbers):

- *If the line is set to -2 it will disconnect all active calls.*
- *If line is set to -1, then it will disconnect the call on the current line (default behavior).*
- *Otherwise it will disconnect the call on the specified line.*
- *More details here*

reject ()

Disconnect incoming call.

Note:

- *You can also use the hangup() function for this*
- *The disconnect code to be sent on user reject can be changed with the userrejectdisccode parameter. Defaults to 603. If your server doesn't work with 603 then we recommend 486 or 480.*

ignore ()

Silently ignore incoming call.

forward (number)

Forward incoming call to the specified number (phone number, username or extension).
For call forward to work, your SIP server and/or the caller endpoint must support the 301 and 302 response codes (Moved Temporarily/Permanently).
Forward should be used for incoming calls only when the call is not connected yet. Otherwise use transfer() to redirect connected calls.

hold (state)

Hold current call. This will issue an UPDATE or a reinvite with the hold state flag in the SDP (sendrecv, sendonly, recvonly and inactive).
Set state to true to put the call on hold or false to un-hold.

mute (state, direction)

Mute current call.
Pass true for the state to mute or false to un-mute.
The direction can have the following values:

0: mute in and out
1: mute out (speakers)
2: mute in (microphone)

mutevideo (state, direction)

Disable/enable video(stream) during a video call.
Pass true for the state to mute the video or false to un-mute.
The direction can have the following values:

0: mute in and out
1: mute remote
2: mute local

transfer (number)

Transfer current call to number which is usually a phone number or a SIP username. (Will use the REFER method after SIP standards).
If the number parameter is empty string and there are two calls in progress, then it will transfer line A to line B.
You can set the mode of the transfer with the transfertype parameter.
For call transfer to work, your SIP server and/or the caller endpoint must support the SIP REFER message (the standard SIP call transfer as specified in RFC 3515 and RFC 5589).
Transfer should be used for connected calls only. For not connected incoming calls you should use the forward() function to redirect the call.

More details can be found here.

conference (number, add)

Add/remove people to conference.
Parameters:
-number: the peer username/number or line number
-add: true if to add, false to remove

If number is empty than will mix the currently running calls (interconnect existing calls if there is more than one call in progress).
If number is a number between 1 and 9 then it will mean the line number.
Otherwise it will call the new number (usually a phone number or a SIP user name) and once connected will join with the current session.

Example:

```
call('999'); //normal call to 999
conference('1234');  //will call 1234 and add to conference (conference between local user + 999 + 1234)
conference('2',false);  //remove line 2 from conference
conference('');  //add all current calls to conference
conference('',false);  //destroy conference (but keep the calls on individual lines)
setline(3); //select the third line
hangup(); //will disconnect the third line
setline(-2); //select all lines
hangup(); //will disconnect all lines
```

Note:

- *if number is empty and there are less than 2 active calls, then the conference function can't be used (you can't put one single active call into a conference)*
- *NS and Java engines also has a local conference mixer thus they are more reliable for conferencing and doesn't depend on any server/gateway/proxy side conference support. If your main use-case is conferencing then we recommend to force one of these engines instead of WebRTC.*
- *you can also use the webphone with your server conference rooms/conference bridge. In this way, there is no need to call this function (just make a normal call to your server conference bridge/room access number)*
- *the conference method can be changed with the conferencetype parameter*
- *more details here*

dtmf (msg)

Send DTMF message by SIP INFO or RFC2833 method (depending on the "dtmfmode" parameter).
This is often used for on key press events or keypad events when to send the pressed number to the server or to the peer for IVR or voicemail inputs.
Please note that the msg parameter is a string. This means that multiple dtmf characters can be passed at once and the webphone will streamline them properly.
The dtmf messages are sent with the protocol specified with the "dtmfmode" parameter.
Use the space character to insert delays between the digits.
Valid dtmf characters in the passed string are the followings: 0,1,2,3,4,5,6,7,8,9,*,#,A,B,C,D,space.
Example:

```
dtmf("1");
dtmf(" 12 345 #");
```

Note: dtmf messages can be also sent by adding it to the called number after a comma. For example if you make a call to 123,456 then it will call 123 and then it will send dtmf 456 once the call is connected.

sendchat (number, msg)

Send a chat message via SIP MESSAGE method as specified in RFC 3428.
Number can be a phone number or SIP username/extension number (or whatever is accepted by your server).
The message can be clear ASCII or UTF-8 text or html encoded.
In order for IM to work your SIP server needs to support SIP MESSAGE. More details can be found here.

sendsms (number, msg, from)

Send a SMS message if your provider/server has support for SMS.

The number parameter can be any mobile number.
The msg is the SMS text.
The from is the local user phone number and it is optional.
SMS can be handled on your server by:
-converting normal chat message to SMS automatically if the destination is a mobile number
-or via an HTTP API (you can specify this to the webphone as the “sms” parameter)

More details can be found here.

play (start, file, looping, toremotepeer)

Play sound file.
At least wave (.wav) files (raw linear PCM) are supported in the following format: PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian (128 kbits).
Parameters:

- start: (number) 1 for start or 0 to stop the playback
- file: (string) file name or full path
- looping: (number) 1 to repeat, 0 to play once
- toremotepeer: (boolean) stream the playback to the connected peer

voicerecord (start, url)

Start/stop voice recording at runtime.
Set the start parameter to true for start or false to stop.
The url is the address where the recorded voice file will be uploaded as described by the voicerecupload setting.

Note:
You can also just set the “voicerecupload” parameter to have all calls recorded.
More details about voice recording can be found here.

videocall (number)

Initiate a video call to a number, extension, sip username or SIP URI.
More details about video calls can be found here.

screenshare (number, screenid)

Initiate a screen sharing session with the user specified by the number parameter.
Parameters:

- number: the peer number, username, extension or SIP URI
- screenid: optional parameter to pass the screen id (It might be useful for some special use-case when you acquire the screen-id via external tools such as the Chrome OS desktop capture API)

Special permissions might be needed. See also the screensharing parameter for details. More details about video calls can be found here.

stopscreenshare ()

Stop screen sharing.

setvideodisplaysize (type,width, height)

Use this function to control the video display size (both for remote and local video).
Accepted parameters:

- type: 1=for remote video container, 2=for local video container
- width: integer value of width in pixels
- height: integer value of height in pixels; this parameter can be left empty or null, and the height will be set depending on the video's aspect ratio

More details can be found in webphone_api.js and video.css file respectively (You can also set the size in the video.css file).

devicepopup ()

Open audio/video device selector dialog (built-in user interface)

getdevicelist(dev, callback)

Call this function and pass a callback, to receive a list of all available audio devices.
For the dev parameter pass 0 for recording device names list, 1 for the playback or ringer devices or 3 for video camera devices.
The callback will be called with a string parameter which will contain the audio device names in separate lines (separated by CRLF).

Note:

- *With the Java or NS engine it might be possible that you receive only the first 31 characters from the device name. This is a limitation coming from the OS audio API but it should not cause any problem, as you can pass it as-is for the other audio device related functions and it will be accepted and recognized.*
- *Some browsers requires HTTPS or file hosting to list the audio devices (might not work from localhost) if you are using the WebRTC engine.*

getdevice(dev, callback)

Call this function and pass a callback, to receive the currently set audio device.
For the "dev" parameter one of the followings are expected:

0: for recording device
1: for the playback device
2: for ringer device
3: for video camera

The callback will be called with a string parameter, which will contain the currently selected audio device.

setdevice(dev, devicename, immediate)

Select audio or video device. The devicename should be a valid audio or video device name which can be listed with the getdevicelist() call.

For the "dev" parameter pass:

0: for recording device
1: for the playback device
2: for ringer device
3: for video camera

The "immediate" parameter can have the following values:

0: default/auto
1: next call only
2: immediately for active calls

Note:

- *Before calling this function, you might call the getdevice function first to obtain the name of the available devices. Then pass one of them with this setdevice API.*
- *The "Default" devicename means system default (no any specific device will be enforced)*
- *When using the NS winapi audio engine, the device names might be truncated to the first 31 characters due to the wave audio API limitations. You can pass the same back to the webphone and it will select the audio device correctly.*
- *The ring device cannot be changed at runtime (the immediate flag will not have any effect if you call this function while the webphone is already ringing).*
- *For the WebRTC engine the ringtone playback needs user interaction. No ringtone will be played for incoming calls if the user doesn't interact with the webphone before the call arrives due to browser media permission / auto-play policy as described here.*

getvolume(dev, callback)

Call this function, passing a callback and will return the volume (percent) for the selected device.
The dev parameter can have the following values:

0 for the recording (microphone) audio device
1 for the playback (speaker) audio device
2 for the ringback (speaker) audio device

The callback will be called with the volume parameter which will be 0 (muted), 50 (default volume) or other positive number.

Note: the reason why this needs a callback (and doesn't just returns the volume as the function return value is because for some engines the volume will be requested in an asynchronous way so it might take some time to complete).

setvolume(dev, volume)

Set volume (percent for the selected device).

0% means muted, 100% means maximum volume.
Default value is 50%, which means no change.

The dev parameter can have the following values:

0 for the recording (microphone) audio device

1 for the playback (speaker) audio device
2 for the ringback (speaker) audio device

WebRTC limitations:
-Changing the recording volume level for WebRTC at run-time will take effect only for the next call. You can also use the volumein parameter to change the default volume.
-Some browsers doesn't support changing the volume with WebRTC. It is expected that the user will control the volume with the OS volume settings.
-A separate volume for ring is not supported on WebRTC

setloudspeaker (set)

Call this function with boolean parameter "true" to set the default playback device to loudspeaker on supported devices such as Android phones and tablets. The speakerphone will be restored if called with "false".

Note: You can also use the androidspeaker parameter to preconfigure.

setsipheader(header)

Set a custom sip header (a line in the SIP signaling) that will be sent with all messages.
Can be used for various integration purposes (for example for sending the http session id or any custom data).
For example: setsipheader('X-MyExtra: whatever');
You can also set extra SIP headers with the customsipheader parameter (setting this parameter is the easiest way if you have some parameter which have to be sent always, regardless of the circumstances)

Note:
- *It is recommended to prefix customer headers with X- so it will bypass SIP proxies.*
- *Multiple lines can be separated by semicolon ; Example: setsipheader('X-MyExtra1: aaa; X-MyExtra2: bbb');*
  If you must use semicolon in the string, then use the CRLF separator even if you have to set only one SIP header (in this case just append it at the end)
- *Multiple lines can be also set by calling this function multiple times with different keys.*
- *There are two kinds of headers that you can set:*
  - *Global (set for all lines including the registrar endpoint): if the line is -2 or there is no current call on the selected line (for example if you set it at startup, before any calls or with line set to -2)*
  - *Per line: if the current line is set and there is an active call on that line.*
    Note that you cannot set it for not already existing lines (lines which doesn't have calls on it yet. For the upcoming call you should set it globally)
- *You can remove all the previously passed headers (per line or global) by calling this function with an empty string. Example: setsipheader('');*
- *You can remove a previously set header by calling this function with an empty key for that header. Example: setsipheader('X-MyExtra:');*

getsipheader(header, callback, must)

Call this function passing a callback.
Example: getsipheader("Contact", mycallback); //will call mycallback with the Contact header from the incoming message
The passed callback function will be called with one parameter, which will be the string value of the requested sip header from the received SIP messages (received from your server of from the other peer). If no such header is found or some other error occurs, then the returned string begins with "ERROR" (for example: "ERROR: no such header") so you might ignore these.
The must parameter is optional and defaults to false. If set to true, then the webphone might try to return the header from any line (not just the current active line).

Note:
-The reason why this needs a callback (and doesn't just returns the last seen header values is because for some engines the signaling messages have to be requested in an asynchronous way so it might take a little time –usually only a few milliseconds- to complete the request).
-The getsipheader() will send you the headers from the incoming SIP messages (not the headers previously set by the setsipheader() function call)
-If you are using the WebRTC engine with a WebRTC-SIP gateway, then you should prefix the extra headers with X- to have to forwarded by the gateway. For example instead of Alert-Info: URN your SIP server should send X-Alert-Info: URN.

getsipmessage(dir, type, callback, must)

Will return the last SIP signaling message as specified by the current line and the dir/type parameters.
Call this function passing a callback.
Example: getsipmessage(0, 3, mycallback); //will call mycallback with the incoming INVITE message text on the current line
The passed callback function will be called with one parameter, which will be the string value of the requested sip message as raw text.
If no such message is found or some other error occurs, then the returned string begins with "ERROR" (for example: "ERROR: SIP message not found") so you might ignore these.
The following parameters are defined:
dir:

-1: any
0: in (incoming/received message)
1: out (outgoing/sent message)

type:

0: any
1: SIP request (such as INVITE, BYE)
2: SIP answer (such as 200 OK, 401 Unauthorized and other response codes)
3: INVITE (the last INVITE received or sent)
4: the last 200 OK (call connect, ok for register or other)

callback:

The callback function

must:

Optional boolean value. Defaults to false.
If set to true, then the webphone might try to return a SIP message from any line (not just the current active line).

You can use this function if you have good SIP knowledge and wish to parse the SIP messages yourself from JavaScript for some reason (for example to extract some part of it to be processed for other purposes).
Example to return the last received INVITE message about an incoming call: getsipmessage(0,3,mysipmsgrecvcallback)

Note: just as other functions, this will take in consideration the active line (set by setline() or auto set on in/out call setup). You can set the active line to "all" [with setline(-2)] to get the last message regardless of the line.

getlastrecinvite (callback)

Get last received SIP INVITE message.
This is a simplified version of the getsipmessage function if you are interested only in the last received INVITE request.
Call this function passing a callback.
The passed callback function will be called with one String parameter, which will be the last incoming call's SIP INVITE message.
This is a helper function which might be useful to get the whole INVITE message on call setup.

getlastsentinvite (callback)

Get last sent SIP INVITE message.
This is a simplified version of the getsipmessage function if you are interested only in the last sent INVITE request.
Call this function passing a callback.
The passed callback function will be called with one String parameter, which will be the last outgoing call's SIP INVITE message.
This is a helper function which might be useful to get the whole INVITE message on call setup.

getlinedetails (line)

Will return the state and various parameters of an endpoint as a string in the following format:
LINEDETAILS,line,state,callid,remoteusername,localusername,type,localaddress,serveraddress,mute,hold,remotefullname

Basically, the parameters here are very similar with the STATUS parameters described here.

getlastcalldetails ()

Returns an unstructured string with details about the previously disconnected call.
You should call this function after the call is actually finished. For example after the onCdr callback have been fired.
For example in case of the NS engine, this might return something like:

```
Line: 1 , Direction: outgoing
Local Username: CALLER , AuthUsername: CALLER
Peer Username: CALLED , Name: CALLED Called: CALLED
Server: SERVERDDRESS
Connectime: X msec
Duration: X msec
Disc By: Local , Reason: User Hung Up
AEC_SETT: 1, AEC_FULL: 2, AEC_FAST: 2, no, AEC_VOL: 1, AGC: 2, Denoise: 2, mediaench: aec: 2, denoise: 2, agc: 2, silencesupress: 0
Payload: X CODEC, pf: 1, network: fast
CPU: speed: 3500, load: 0.0, reccpuhigh: false, subshigh: 0, highdetected: no, meassured: 0 0
Audio:  Record: count: 254 ival: 20 last: 5 0 Playback: count: 257 ival: 20 last: 5 0 issues: 2 500 16 Queue: 1 0 2 4 0
```

getcallerdisplayfull (callback)

Call this function and pass a callback, to receive the incoming detailed caller id (might return two lines: caller id \n caller name)
Note: If you just need the caller id, you should just use the onCallStateChange peer name instead of this function.
The callback will be called with a string parameter which will contain the incoming detailed caller id.

See the Caller-ID FAQ point for more details.

getregfailreason ()

Returns a string with the reason about failed connect/registration.
Deprecated! Use the onRegStateChange callback “unregistered” event instead!

setline (line)

This function can be used for explicit line/channel management and it will set the current active channel.
For the line parameter you can pass one of the followings:

-line number: -2 (all), -1 (current/best), 0 (invalid), 1 (first channel), 2 (second channel) …. 100
-call id (so the active line will be set to the line number of the endpoint with this SIP Call-ID)
-peer username (so the active line will be set to the line number of the endpoint where the peer is this user)

Use this function only if you present line selection for the users. Otherwise you don’t have to take care about the lines as it is managed automatically (with each call on the first “free” line)

Note:

- You can set the line to -2 and -1 only for a short period. After some time the getline() will report the real active line or “best” line.
- The internal state machine might automatically change the active line after 300 milliseconds (this means that you must perform the operation on the selected line quickly after this setline API call)
- Using the setline function will not mute the other lines if there are multiple simultaneous calls. It will just select the line for any further operation. For example if you wish to mute line 2, then you have to call setline(2); and then mute(true);

More details about multi-line can be found in the FAQ.

getline ()

Return type: number
Will return the current active line number. This should be the line which you have set previously except after incoming and outgoing calls (the webphone will automatically switch the active line to a new free line for these if the current active line is already occupied by a call).
More details about multi-line can be found in the FAQ.

linetocallid (line, callback)

Utility function to get the SIP Call-ID for a line number. Might be useful in some circumstances.
The passed callback function will be called with one parameter, which will be the SIP Call-ID for the passed line number.
If not found then an empty string will be returned in the callback.

callidtoline (callid, callback)

Utility function to get the line number for a SIP Call-ID. Might be useful in some circumstances.
The passed callback function will be called with one parameter, which will be the line number for the passed SIP Call-ID.
If not found then a negative line number will be returned.

nextcallid (callid)

Set/get the SIP Call-ID used for the first upcoming new SIP session.
This function might be used if you need the SIP Call-ID before the call for some reason. For example you can use it before the call function to set/get the SIP Call-ID of the new call. Otherwise the webphone will automatically generate an unique SIP Call-ID for each new session which you can query using the linetocallid function or receive it with the onCallStateChanged or onCdr callbacks.
If this function is called without the callid parameter or the callid as an empty string then the webphone will generate and return a random call-id.
The return value is the new SIP Call-ID as string.

isregistered ()

Return type: boolean
Return true if the webphone is registered ("connected") to the SIP server.
Note: you can track the phone state machine also with the events callbacks or check this FAQ.

isincall ()

Return type: boolean

Return true if the webphone is in call, otherwise false.
Note: you can track the phone state machine also with the events callbacks.

ismuted ()

Return type: boolean
Return true if the call is muted, otherwise will return false.

isonhold ()

Return type: boolean
Return true if the call is on hold, otherwise will return false.

isencrypted ()

Check if communication channel is encrypted: -1=unknown, 0=no, 1=partially, 2=yes, 3=always

checkblf (userlist)

Subscribe to call state of other extensions to receive call state change information as BLF notifications by triggering the onBlfStateChange callback.
Userlist: list of sip account username separated by comma.
In order for BLF (busy lamp field) to work, make sure that your server and peer(s) has support for BLF subscribe/notify with dialog event package as described in RFC 3265 and RFC 4235.

Note:
If BLF is an important feature for you, then we recommend the usage of the NS or Java engines as this is unreliable with WebRTC (set the enginepriority_ns and enginepriority_java to 4)
More details here.

checkpresence (userlist)

Will receive presence information PRESENCE notifications and will trigger the onPresenceStateChange callback.
Userlist: list of sip account username separated by comma.
In order for presence to work, make sure that your server has support for subscribe/notify. More details here.

setpresencestatus (status)

Function call to change the user online status with one of the followings strings: Online, Away, DND, Invisible, Offline (case sensitive).

getenginename ()

Returns the currently used engine name as string: "java", "webrtc", "ns", "app", "flash", "p2p", "natividial".
Can return empty string if engine selection is in progress.
Might be used to detect the capabilities at runtime (for example whether you can use the jvoip function or not)

listcallhistory ()

This function will return a String containing the whole call history list. If there are no entries in call history, it will return null.
Call history entries will be separated by "carriage return new line": \r\n
Call history fields will be separated by "tabs": \t

The order of fields and their meaning:

- type: int - 0=Outgoing call, 1=Incoming call, 2=Missed call
- name: String - can be a name, can be the same as the number or can be empty String
- number: String - the phone number/SIP URI
- date: int - timestamp date of call
- duration: int - duration of the call in milliseconds
- discreason: String - call disconnnect reason

Example: 1 \t Name \t Number \t Date \t Duration \t Discreason

listcontacts (all)

This function will return a String containing the whole contact list. If there are no contacts, it will return null.
Set the all parameter to true to receive also virtual contacts as well, like voicenumber, etc...
Contacts will be separated by "carriage return new line": \r\n
    Contact fields will be separated by "tabs": \t
    A contact can have more than one phone numbers or SIP URIs, so these will be separated by Pipe(Vertical bar): |
    See example below:
The order of fields and their meaning:
name: String - the name of the contact
number: array of String - the number(s)/SIP URI(s) of the contact
favorite: int - 0=No, 1=Yes
email: String - email address of the contact
address: String - the address of the contact
notes: String - notes attached to this contact
website: String: web site attached to this contact
types: array of String –the type of the above number(s)

Example: Name \t Number1|Number2 \t Favorite \t Email \t Address \t Notes \t Website \t Type1| Type2 \r\n

addcontact (fullname, number, email, address, notes, website, type, favorite)

Add a contact to the contact list.
This is useful only with the softphone skin (otherwise most probably you don't need a contact list or you manage it yourself on a custom user interface after your needs).
The number and the type can be a list (strings separated by |).
For the type you can set any string, but on the softphone skin only the following are handled: phone, home, mobile, work, other, fax_home, fax_work, pager, sip.

Examples:
```
webphone_api.addcontact('Kate', '1111'); //add a simple contact with name: Kate, number: 1111
webphone_api.addcontact('John Smith', '2222|3333');  //add contact with name: John Smith and two numbers
webphone_api.addcontact('James Oliver', '+40741234567', 'james@gmail.com', 'Victoria street', 'hi', 'www.example.com', 'mobile', '1'); //add a favorite contact with all parameters set
webphone_api.addcontact('James Oliver', 444444|555, 'james@gmail.com', 'Victoria street', 'hi', 'www.example.com', 'phone|sip', '0'); //add contact with multiple numbers and all parameters set
```

Note:
Contacts can be prepopulated also by using the defcontacts parameter.
If the normalize_contact parameter is set to 1, then the webphone will "normalize" the contact details. Set to 0 if you wish to disable this.

getcontact (name, number)

Call this function passing the name and/or number of the contact.
Will return a String with the following parameters separated by "tabs": \t if found:

- name: String - the name of the contact
- number: String - number(s)/SIP URI(s) of the contact separated by Pipe(Vertical bar): |
- favorite: int - 0=No, 1=Yes
- email: String - email address of the contact
- address: String - the address of the contact
- notes: String - notes attached to this contact
- website: String: web site attached to this contact
- type: the type of the above number(s)

If contact is not found, then it will return null.

delcontact (name, number)

Delete contact from the contact list where the name or the number match.
This is useful only with the softphone skin (otherwise most probably you don't need a contact list or you manage it yourself on a custom user interface after your needs).
Use the delallcontacts() API if you wish to delete all contacts.

getworkdir ()

Returns the working directory for the NS and Java engines (not applicable for WebRTC and Flash).

The working directory is the folder which is used to save any files (configurations, logs, voice recordings).

delsettings (level)

Delete stored data (from cookie, config file and local-storage).
For the level parameters is optional and it can be set to one of the followings:

- 0: delete the settings only if it was not already cleared (default)
- 1: delete the settings
- 2: force all (delete everything: settings, contacts, call history, messages and all other data)
- 3: delete also library cache

Warning: this will delete all settings, even those that should be remembered between sessions, such as "Have we already asked X from the enduser?". To deal with different settings we recommend using the profiles instead of using this API or just set unneeded settings to NULL. More details can be found here.
This API might stop or restart the SIP engine.
You should call this on logout (not at start) if for some reason you wish to delete the stored phone settings.

getlogs ()

Returns a string containing all the accumulated logs by the webphone (the logs are limited on size, so old logs will be lost after long run).
More details about logs can be found here.

getstatus ()

Returns the webphone global status. The possible returned texts are the same like for getEvenetsnotifications.
You might use the events described below instead of polling this function.

others

We have listed all the important and commonly used API's in this documentation, however, there are some other advanced/deprecated/internal API's that can be accessed.

Advanced functions
There are a few additional not so important functions provided in the webphone_api.js. Inspect the webphone_api.js file for the details.

Helper functions
There are a few helper/internal/not so important functions that can be accessed from \js\lib\api_helper.js.

WebRTC internals
When using the WebRTC engine, you can access the WebRTC browser API via the following functions:
webphone_api.getrtcsocket() -Returns a reference to the WebRTC engine socket object. Returned value can be null or the reference to the WebSocket object.
webphone_api.getcallsession (line, destnr, callid) -Returns a reference to the internal call session object. Returned value can be null or the reference to the call session object. You might pass one or more parameters to specify the required call session: line, destination number, Call-ID. If no parameters are specified, then it will return the default/current session if any.
webphone_api.getrtcpeerconnection(line, destnr, callid) -Returns a reference to the current/last WebRTC RTCPeerconnection object or null if no RTCPeerconnection exists.

Deprecated functions
The webphone is fully backward and forward compatible and we never remove old functions. All the old deprecated functions can be still used as-is, without the need to change your code. These functions are declared in the webphone_api.js or moved to \js\lib\api_helper.js (they are still available via the webphone_api object).
The major deprecated functions are the followings:

- onLoaded, onStart, onStop (replaced with onAppStateChange)
- onRegistered, onRegisterFailed, onUnRegistered (replaced with onRegStateChange)
- callSetup, callRinging, callConnected, callDisconnected (replaced with onCallStateChange setup/ringing/connected/disconnected events)
- onDisplay, onLog, onEvents (replaced with onEvent)

The call state events strings was also changed from callSetup/callRinging/callConnected/callDisconnected to setup/ringing/connected/disconnected. You can restore the old behavior by setting the apibehaviour to 0. For the new behavior you can set the apibehaviour parameter to 1. New webphone build are set with this by default. For old customers the default value is -1 which means reporting strings both in old and new format.

Internal engines
If engine is Java or the NS Service plugin, then you can access the full low-level API as described in the JVoIP SDK documentation.
This can be done with the jvoip function call:
jvoip(name, jargs)
Parameters:

Name: name of the function
Jargs: array of arguments passed to the called function. Must be an array, if API function has parameters. If API function has no parameters, then it can be an empty array, null, or omitted altogether.
For example the low level API function: API_Call(number) can be called like this: webphone_api.jvoip('API_Call', line, number);

Events

The below callback functions can be used to receive events from the webphone such as the phone state machine status (registered/call init/call connected/disconnected) and other important events and notifications.
Note: passing null as the callback parameter for these functions will remove all previously set callbacks (clear the internal callback functions array).

onAppStateChange (callback)

You can use this function to track app state changes.
The passed callback function will be called on every webphone app state change. The callback function has a state string parameter containing the new state of the application.
The states are the followings:

- "loaded" - the webphone library is completely loaded so you can start the usage (call any other API functions only after this state is received!).
- "started" - the VoIP engine has successfully started (ready to register or make calls). Note: you can initiate calls also from the "loaded" event and it will be queued to be executed after started.
- "stopped" - the VoIP engine has stopped (all endpoints unregistered/disconnected and the sipstack was destroyed). Usually triggered at page unload.

Example:

```
webphone_api.onAppStateChange(function (state) //watch for webphone state changes
{
        if (state === 'loaded') //webphone library loaded
        {
                webphone_api.setparameter("name", "value"); //set any parameters
                webphone_api.start(); //start if not started automatically (for example if the autostart parameter is set to 0)

                //setup call state machine watcher
                webphone_api.onCallStateChange(function (event, direction, peername, peerdisplayname, line, callid)
                {
                        //handle any call events here. For example to catch the call connect events:
                        if (event === 'connected')
                        {
                            alert("Call connected with " + peername);
                        }
                        //else if (event === ' disconnected') …
                });

        }
        else if (state === 'started') //sip stack started
        {
                webphone_api.call('NUMBER'); //make a call immediately once started
        }
});
```

Note: the onAppStateChange deprecated to old onLoaded, onStart and onStop functions but these are still available and you don't need to change any old code.

onRegStateChange (callback)

You can use this function to track register state changes.
The REGISTER method in SIP is used to "connect" to the SIP server, so the server can learn the device location to be used to route incoming calls and other messages.
The passed callback function will be called on every register state change with a state string parameter.
The states are the followings:

- "registered" – registered successfully to the SIP server
- "unregistered" – unregistered from the SIP server (if user closes the webpage, then you might not have enough time to catch this event)
- "failed" – register failed. You can receive the cause of the failure in a second reason string parameter.

Example:

```
webphone_api.onRegStateChange(function (state, reason) //watch for webphone state changes
{
        console.log('register state changed to '+state);
        if (typeof (reason) !== 'undefined' && reason !== null) console.log('reason '+reason);
});
```

onCallStateChange (callback)

You can use this function to track call state changes.
The passed callback function will be called on every call state change (triggered when the webphone is in call).
Parameters:

- event: can have following string values: setup, ringing, connected, disconnected (other events such as "midcall" can be ignored)
- direction: 1 (for outgoing call), 2 (for incoming call)
- peername: is the other party username (or phone number or extension)
- peerdisplayname: is the other party display name if any
- line: channel number (this might be important only if you wish to handle multi-line management explicitly)
- callid: SIP Call-ID

A simple usage example can be found here.

Note:

- *the call might change from setup state directly into disconnected state if it was connected*
- *ringing state is optional (it is not a definitive answer for the INVITE transaction so it is not mandatory in SIP)*
- *you might receive other event strings which can be just safely ignored*

onPresenceStateChange (callback)

You can use this function to track presence changes.
The passed callback function will be called on every presence (online/offline/others) state change (of the remote users).
To initiate presence requests, you can use the checkpresence() API or the presenceuserlist parameter.
Parameters:

- peername: is the other party username (or phone number or extension)
- presence: can have following values:
  CallMe,Available,Ready,Pending,Other,CallForward,Speaking,Busy,Idle,DoNotDisturb,Unknown,Away,Offline,Exists,NotExists,Unknown
- displayname: optional peer display name if sent as part of presence
- email: optional peer email address if sent as part of presence

onBLFStateChange (callback)

You can use this function to track BLF changes.
The passed callback function will be called on every call state change (of the remote users) which might be used as BLF (busy lamp field)
To initiate presence requests, you can use the checkblf() API or the blfuserlist parameter.

Parameters:

- peername: is the other party username (or phone number or extension)
- direction: can have one of following values: undefined, initiator (outgoing call) or receiver (incoming call)
  - undefined (not for calls or no announced by the peer)
  - initiator (outgoing call)
  - receiver (incoming call)
- state: can have one of following values: trying , proceeding, early, comfirmed, terminated, unknown or failed
  - trying (call connect initiated)
  - proceeding (call connecting)
  - early (ringing or session progress)
  - confirmed (call connected)
  - terminated (call disconnected)
  - unknown (unrecognized call state received)
  - failed (BLF subscribe or notify failed)
- callid: optional SIP call-id of the call (if reported by the BLF notify)

onChat (callback)

The passed callback function will be called when chat message is received.
Parameters:

- from: username, phone number or SIP URI of the sender
- msg: the content of the text message
- line number
- group: group name; the name of the group if it's group chat, otherwise it will be an empty string or “null” (“null” might be received if other endpoint has group chat support but this message is not part of a group chat

onSMS (callback)

The passed callback function will be called when an SMS message is received.
Might be triggered only if the server has VoIP SMS support or SMS API implemented.
Parameters:

- from: username, phone number or SIP URI of the sender
- msg: the content of the text message

onDTMF (callback)

The passed callback function will be called when a DTMF digit is received.
Parameters:

- dtmf: received DTMF character as string
- line: channel number

Supported DTMF receive method with all engines is INFO SIP messages. The NS and Java engines has support also for RFC 2833 in RTP. See the dtmfmode for more details.

onCdr (callback)

The passed callback function will be called at each call disconnect. You will receive a CDR (call detail record).
Parameters:

- caller: the caller party username (or number or sip uri)
- called: called party username (or number or sip uri)
- connect time: milliseconds elapsed between call initiation and call connect/reject/cancel (includes the call setup time + the ring time)
- duration: milliseconds elapsed between call connect and hangup (0 for not connected calls, otherwise you might add 400 and divide by 1000 to obtain rounded seconds)
- direction: 1 (outgoing call), 2 (incoming call)
- peerdisplayname: is the other party display name if any
- reason: disconnect reason as string
- line: channel number
- callid: SIP Call-ID
- discparty: the party which was initiated the disconnect: 0=not set, 1=local JVoIP, 2=peer, 3=undefined/timeout

Note:
Missing values are empty or set to "NA".
You can get some more details about the call by using the getlastcalldetails() function.

Every SIP server is capable to store CDR records, however if somehow you have some special requirement, then you can also submit the CDR from this callback to your web service API (for example if the SIP server is not owned by you or if you need any extra or special processing).
More details here.

onEvent (callback)

You might use this function to receive internal events, logs and messages that might be processed or displayed to users.
This should be used only if for some reason you need more than the above described onCallStateChange and other callbacks.

Call this function once and pass a callback, to receive important events, which should be displayed for the user and/or parsed to perform other actions after your software custom logic. You will also receive all log messages depending on the value of "loglevel" parameter.
For the included softphone these are already handled, so no need to use this, except if you need some extra custom actions or functionality which depends on the notifications.
Parameters:

- type: the type of the message. This can have the following values:
  - event: low level notifications which can be parsed as string.
    Example: STATUS,1,Ringing,2222,1111,2,Katie,[callid]
    Instead of parsing these low level string notifications, you should use the callback functions instead such as onAppStateChange, onCallStateChange, onChat and the other onXXX functions.
    See the "Notifications" section below for all possible strings.
  - log: log messages
    Usually you don't need to check for the logs.
    You might watch for this only if you wish to collect the logs for some purpose or you have to watch for a log string to be triggered.
    It can be used for debugging or for log redirection if the other possibilities don't fit your needs.
  - display: important messages to be displayed for the user.
    If you wish to handle the popups yourself, then disable popups by settings showtoasts parameter to "false". From here on you will be responsible for presenting these messages to the user.
    If type is "display", then the "message" parameter will be composed of a "title" until the first comma and a "text".
    The title can be an empty string, in which case the message begins with a comma.
- message: the text of the event/log/display

```
webphone_api.onEvent( function (type, message)
{
	// For example the following status means that there is an incoming call ringing from 2222 on the first line:
	// STATUS,1,Ringing,2222,1111,2,Katie,[callid]
	// You can find more detailed explanation about events in the documentation "Notifications" section.

	// example for detecting incoming call:
	if (type === 'event')
	{
		var evtarray = message.split(','); //parameters are separated by comma (,)

		if (evtarray[0] === 'STATUS' && evtarray[2] === 'Ringing')
		{
			if (evtarray[5] === '1') // 1 means it is an outgoing call
			{
				//add any custom logic here, for example you might lookup the caller from a database
				alert('Incoming call from: '+evtarray[3] + ' ' + evtarray[6]); //of course, instead of alert you should use some better html display
			}
		}
	}

	// example for handling displayable messages
	else if (type === 'display')
	{
		var position = message.indexOf(',');
		var title = message.substring(0, position); // NOTE: title can be empty string
		var text = message.substring(position + 1);
		alert(title+'\r\n'+text); //of course, instead of alert you should use some better html display

		//NOTE: If you wish to handle the popups yourself, then disable popups by settings "showtoasts" parameter to "false".
	}
});
```

Notes:

- This is a new function in v.3.0 to replace the old onDisplay, onLog and onEvents callback.
- It is possible to use this function to catch all the event notifications to track all state machine changes instead of the onAppStateChange, onCallStateChange, onChat and the other onXXX functions. However, we recommend to use the high level callback functions instead of error prone string parsing from this onEvent function.
- You might also check the basic_example.html included in the package regarding the onEvent function usage.
- display type events:
  With the display messages you can receive important events and notifications (as strings) that should be displayed to the user.
  For example:
  - "Invalid phone number or SIP URI or username" (displayed if user is trying to call an invalid peer)
  - "Waiting for permission. Please push the Allow/Share button in your browser..." (when waiting for WebRTC browser permission)
  - “Check your microphone! No audio record detected.” (which is displayed after 6 seconds in calls if the VAD doesn’t report any activity).

  The text of the message is language dependent, meaning if the language of the webphone is changed, the message/title language is also changed.
  Engine selection related popups are always handled by the webphone (However these are presented only when really necessary and can be suppressed by forcing the webphone to a single engine)
  Notifications will be displayed as auto-hiding popups if not disabled with the showtoasts parameter

Notifications

“Notifications” means simple string messages received from the webphone which you can parse from the onEvent(callback) to receive notifications and events from the sip web phone about its state machine, calls states and important events.
Skip this section if you are not using the onEvent() function catching the “event” messages. (You can use the functions such as onRegStateChange/onCallStateChange/others to catch the important events in which you are interested in and completely skip this section about the low-level notification strings handling).

If you are using the onEvent() function then you can parse the received event strings from your java script code. Each notification is received in a separate line (separated by CRLF). Parameters are separated by comma ‘,’. For the included softphone and click to call these are already handled, so no need to change, except if you need some extra custom actions or functionality.

The following messages are defined:

STATUS,line,statustext,peername,localname,endpointtype

You will receive STATUS notifications when the SIP session state is changed (SIP session state machine changes) or periodically even if there was no any change.
A typical status strings looks like this:
STATUS,line,statustext,peername,localname,endpointtype,peerdisplayname,[callid],online,registered,incall,mute,hold,encrypted,isvideo

The **line** parameter can be -1 for general status or a positive value for the different lines.
General status means the status for the "best" endpoint.
This means that you will usually see the same status twice (or more). Once for general sip client status and once for line status.
For example you can receive the following two messages consecutively: STATUS,-1,Connected,…
You might decide to parse only general status messages (where the line is -1), messages for specific line (where line is zero or positive) or both.

Status notifications are sent on state change but also at a regular interval (so you can see the same notification again and again even if there was no state change).

The following **statustext** values are defined **for general status (line set to -1):**

- Initializing
- Ready
- Register…
- Registering…
- Register Failed
- Registered
- Accept
- Starting Call
- Call
- Call Initiated
- Calling…
- Ringing…
- Incoming…
- In Call (xxx sec)
- Hangup
- Call Finished
- Chat

Note: general status means the "best" status among all lines. For example if one line is speaking, then the general status will be "In Call".

The following **statustext** values are defined **for individual lines** (line set to a positive value representing the channel number starting with 1):

- Unknown (you should not receive this)
- Init (started)
- Ready (sip stack started)
- Outband (notify/options/etc. you should skip this)
- **Register** (from register endpoints)
- Subscribe (presence or BLF)
- Chat (IM)
    - **CallSetup** (one time event: call begin)
- Setup (call init)
- InProgress (call init)
- Routed (call init)
- Ringing (SIP 180 received or similar)
    - **CallConnect** (one time event: call was just connected)
- InCall (call is connected)
- Muted (connected call in muted status)
- Hold (connected call in hold status)
- Transfer
- Conference
- Speaking (call is connected)
- Midcall (might be received for transfer, conference, etc. you should treat it like the Speaking status)
    - **CallDisconnect** (one time event: call was just disconnected)
- Finishing (call is about to be finished. Disconnect message sent: BYE, CANCEL or 400-600 code)
- Finished (call is finished. ACK or 200 OK was received or timeout)
- Deletable (endpoint is about to be destroyed. You should skip this)
- Error (you should not receive this)

You will usually have to display the call status for the user, and when a call arrives you might have to display an accept/reject button.
For simplified call management, you can just check for the one-time events (CallSetup, CallConnect, CallDisconnect)

Peername is the other party username (if any)
Localname is the local user name (or username).

Endpointtype is 1 from client endpoints and 2 from server endpoints.
Peerdisplayname is the other party display name if any

Optional values (you might not always receive these):
CallID: SIP session id
Online: network status. 0 if no network or internet connection, 1 if connectivity detected (always the global status)
Registered: registration state. 0=unknown,1=not needed,2=yes,3=working yes,4=working unknown,5=working no,6=unregistered, 7+ =no (failed) (always the global status)
InCall: phone/line is in call. 0=no,1=setup or ringing,2=speaking
Mute: is muted status. 0=no,1=undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Hold: is on hold status: 0=no,1= undefined,2=hold,3=other party hold,4=both in hold (or if you wish to simplify: 0 means no, others means yes)
Encrypted: encryption status: -1: unknown, 0=no,1=partially/weak yes,2=yes,3=always strong (or if you wish to simplify: 0 and -1 means no, others means yes)
Isvideo: set to 1 for video calls (with video media offer). Otherwise 0.

For example the following status means that there is an incoming call ringing from 2222 on the first line:

STATUS,1,Ringing,2222,1111,**2**,Katie,[callid]

The following status means an outgoing call in progress to 2222 on the second line:

STATUS,2,Speaking,2222,1111,**1**,[callid]

To display the "global" phone status, you will have to do the followings:

1. Parse the received string (parameters separated by comma)
2. If the first parameter is "STATUS" then continue
3. Check the second parameter. It "-1" continue otherwise nothing to do
4. Display the third parameter (Set the caption of a custom html control)
5. Depending on the status, you might need to do some other action. For example display your "Hangup" button if the status is between "Setup" and "Finishing" or popup a new window on "Ringing" status if the endpointtype is "2" (for incoming calls only; not for outgoing)

If the "jsscripstats" is on (set to a value higher than 0) then you will receive extended status messages containing also media parameters at the end of each call:

STATUS,1,Connected,peername,localname,endpointtype,peerdisplayname,**rtpsent,rtprec,rtploss,rtplosspercet,serverstats_if_received**,[callid]

REGISTER,line,state,text,main,fcm,user,reason

This notification is received for register state changes from registrar endpoints.
The notification is sent only from the NS and Java engines (not from WebRTC).

line: channel number (should be always 0 for register)
state: 0: in progress, 1: success, 2: failed, 3: unregistered
text: register state as text
main: true for primary account, false for secondary registrations (if you are using multiple accounts)
fcm: not used (ignore)
user: local username (useful if you are using multiple accounts)
reason: failure reason text if any

Note: This notification can be disabled by setting the sendregisternotifications parameter to 0 (for compatibility reasons with old versions).

PRESENCE,peername,state,details,displayname,email

This notification is received for incoming chat messages.
Line: used phone line
Peername: username of the peer
State and details: presence status string; one of the followings:
CallMe,Available,Open,Pending,Other,CallForward,Speaking,Busy,Idle,DoNotDisturb,DND,Unknown,Away,Offline,Closed,Close,Unreacheable,
Unregistered,Invisible,Exists,NotExists,Unknown,Not Set
Displayname: peer full name (it can be empty)
Email: peer email address (it can be empty)

Notes for the state and details fields:
One of these fields might be empty in some circumstances and might not be a string in the above list (especially the details).
The details field will provide a more exact description (for example "Unreachable") while the state field will provide a more exact one (for example "Close"). For this reason if you have a presence control to be changed, check the details string first and if you can't recognize its content, then check the state string. For displaying the state as text, you should display the details field (and display the state field only if the details string is empty).

BLF,peername,direction,state,callid

These notifications are received as an answer for a previous checkblf() request or if the blfuserlist was set and it represents the call state of the peer.
Peername: username of the peer extension
direction: undefined, initiator or receiver

state: trying, proceeding, early, confirmed, terminated, unknown or failed
callid: option sip call-id of the call

CHAT,line,peername,text

This notification is received for incoming chat messages.
Line: used phone line
Peername: username of the sender
Text: the chat message body

CHATCOMPOSING,line,peername,composing

This notification might be received when the other peer start/stop typing (RFC 3994):
Line: used phone line
Peername: username of the sender
Composing: 0=idle, 1=typing

CHATREPORT,line,peername,status,text

This notification is received for the last outgoing chat message to report success/fail:
Line: used phone line
Peername: username of the sender
Status: 0=unknown,1=sending,2=successfully sent,3=failed to send
Text: failure reason (if Status is 3)

CDR,line,peername,caller,called,peeraddress,connecttime,duration,discparty

After each call, you will receive a CDR (call detail record) with the following parameters:
Line: used phone line
Peername: other party username, phone number or SIP URI
Caller: the caller party name (our username in case when we are initiated the call, otherwise the remote username, displayname, phone number or URI)
Called: called party name (our username in case when we are receiving the call, otherwise the remote username, phone number or URI)
Peeraddress: other endpoint address (usually the VoIP server IP or domain name)
Connecttime: milliseconds elapsed between call initiation and call connect
Duration: milliseconds elapsed between call connect and hangup (0 for not connected calls. Divide by 1000 to obtain seconds.)
Discparty: the party which was initiated the disconnect: 0=not set, 1=local, 2=peer, 3=undefined
Disconnect reason: a text about the reason of the call disconnect (SIP disconnect code, CANCEL, BYE or some other error text)

DTMF,line,peername,status,text

Incoming DTMF notification (the digit is passed in the msg parameter)

VREC,line,stage,type,path,reason

Voice upload status (for voice recording / call recording).
line: channel number (note: with stage 3 and 4 it will always report -1 or -2 means default/not specified)
stage: 0:disabled, 1:call record begin,2:upload begin, 3: upload success, 4: upload fail (note: stage 0 might not be reported)
type: upload method: 0: unknown, 1: local file, 2: ftp, 3: http, 4: server
path: upload path/file (note: if stage is 1 then type and path is not reported yet)
reason: failure reason (if stage is 4)
Note: this is sent only from the NS and Java engines. There is no client side feedback about WebRTC call recording

START,what

This message is sent immediately after startup (so from here you can also know that the SIP engine was started successfully).
The what parameter can have the following values:
“api” -api is ready to use
“sip” –sipstack was started

EVENT,TYPE,txt

Important events which should be displayed for the user.

The following TYPE are defined: EVENT, WARNING, ERROR
This means that you might receive messages like this:
WPNOTIFICATION,EVENT,EVENT,any text NEOL \r\n

POPUP,txt

Should be displayed for the users in some way.

ACTION,txt

Various custom messages. Ignore.

LOG,TYPE,txt

Detailed logs (may include SIP signaling).
The following TYPE are defined: EVENT, WARNING, ERROR

VAD,parameters

Voice activity (only for the NS or Java engine. Not for WebRTC)
This is sent in around every 2000 milliseconds (2 seconds) by default from java and NS engines (configurable with the vadstat_ival parameter in milliseconds) if you set the "vadstat" parameter to 3 or it can be requested by API_VAD via the internal engine. Also make sure that the "vad" parameter is set to at least "2". This notification can be used to detect speaking/silence or to display a visual voice activity indicator.

Format:
VAD,local_vad: ON local_avg: 0 local_max: 0 local_speaking: no remote_vad: ON remote_avg: 0 remote_max: 0 remote_speaking: no

Parameters:
local_vad: whether VAD is measured for microphone: ON or OFF
local_avg: average signal level from microphone
local_max: maximum signal level from microphone
local_speaking: local user speak detected: yes or no

remote_vad: whether VAD is measured from peer to speaker out: ON or OFF
remote_avg: average signal level from peer to speaker out
remote_max: maximum signal level from peer to speaker out
remote_speaking: peer user speak detected: yes or no

RTPSTAT,quality,sent,rec,issues,lost

Media quality reports sent by the NS and Java engines (not by WebRTC) if you set the rtpstat parameter to -1 or to a positive value.
Possible values for rtpstat:

- -1: auto (will trigger RTPSTAT events in every 6-7 seconds and more frequently at the beginning of the calls)
- 0: disabled (default)
- Positive value: seconds to generate RTPSTAT events.

RTPSTAT notification parameters:

- quality: number between 0 (worst) and 5 (best). The calculations considers many factors such as RTP issues, RTCP reports, codec problems, packet loss, packet delay, jitter and processing delay.
- sent: RTP packets sent
- rec: RTP packets received and played
- issues: number of issues (any issues are counted, such as sequence number mismatch or packet drop)
- lost: lost packets

More details here.

Other notifications

Format: messageheader, messagetext. The followings are defined:

"**CREDIT**" messages are received with the user balance status if the server is sending such messages.
"**RATING**" messages are received on call setup with the current call cost (tariff) or maximum call duration if the server is sending such messages.
"**MWI**" messages are received on new voicemail notifications if you have enabled voicemail and there are pending new messages

"**SERVERCONTACTS**" contact found at local VoIP server
"**NEWUSER**" new user request
"**ANSWER**" answer for previous request (usually http requests)

FAQ

How to get my own webphone?

1. Try from your desktop or webserver by downloading the webphone package or try the online demo.
2. If you like it, we can send your own licensed copy within one workday on your payment.
   Before to purchase, make sure that all the features which are critical to your use-case is working correctly.
   You can find the pricing and order from here. For the payment we can accept PayPal, credit card or wire transfer.
3. Once we receive the notification about your payment, we might contact you for your company details required for accounting/invoicing, the address of your server(s) and other customization details then will send your licensed copy by email/download link within maximum one work-day (usually it takes only around two hours in business hours).
   In case if you deployed the demo/trial version before, then you will just have to overwrite your old copy to continue the usage with your licensed version, which will remove the demo/trial limitations.

What about support?

We offer support and maintenance upgrades to all our customers. Guaranteed supports hours depend on the purchased license plan and are included in the price.
Email to webphone@mizu-voip.com with any issue you might have.
Please include the followings with your message:

- exact issue description
- screenshot if applicable
- detailed logs
- optionally a description about how we can reproduce the problem with valid sip test account(s)

If the included support period with your license is expired, it can be increased by 2 years for around $600 (Note: This is completely optional. There is no need for any support plan to operate your webphone). For gold partners we also offer priority, phone and 24/7 emergency support.

Note:

- We can't guarantee that all the listed features will work correctly in your environment (your SIP servers / integrated with your software / hosted on your Web server). Please test the demo/trial version before purchase to make sure that all the functionality critical to your use-case works correctly.
- Our support answer time is usually up to one work-day for issues related to core functionality (such as connect/register and voice calls). We are trying to resolve all such issues as soon as possible. Issues affecting non-core features (such as call hold or presence) might be handled only in the upcoming new releases.
- Direct support is provided for the common features (voice calls, chat, dtmf and others) and common OS/browsers (Windows/Linux/Android/MAC OS, IE/Firefox/Chrome/Safari/Opera) and not for extra features (such as presence, fax or webrtc conference) and exotic OS/Browsers (such as FreeBSD, Konqueror). The webphone should work also with other OS/browsers, but we are not testing every release against all exotic platforms.
- Direct support is not provided for the issues described in the known limitations section. If you have some special must-to-have requirement, we recommend to test with the demo version before purchasing your license.
- While video and screen-share should work via WebRTC on all supported platforms, we recommend testing it first in your environment before to purchase if this feature is important for you. Some SIP servers and devices doesn't handle the video features correctly and this is very difficult to debug (We can't offer support for extra video features. It either works or not works in your environment depending on your SIP server and SIP peers capabilities).
- Mizutech doesn't provide direct server side support. Although the webphone is known to work well with all common SIP servers (including Asterisk, FreeSWITCH and many others), your servers have to be managed by you and we are not responsible for proper configuration of your servers.
- We are highly specialized for VoIP development and not designers. Please don't contact us with user interface development or design related requests as we wish to focus only on VoIP/SIP/WebRTC specific projects. The webphone already includes a few ready to use skins for your convenience with their source code in your hand which you can modify as you wish or create your own user interface from scratch if none of them fulfills your needs. These tasks (both modifying the built-in skins and creating new ones) are described in details in this documentation.

What I will receive once I have made the payment for the webphone?

You will receive the followings:

- the web phone software itself (the webphone files –latest stable version- including the engines, javascript API, html5/css skins and examples)
- the ready-to-use/turn-key softphone skin and click to call button
- latest documentations and code examples
- invoice (on request or if you haven't received it before the payment)
- support on your request according to the license plan

Can Mizutech do custom development if required?

Yes.
You can fully customize the webphone yourself by its numerous configuration options. However if you have some specific requirement which you can't handle yourself, please contact us at webphone@mizu-voip.com. Contact us only with webphone/VoIP specific requirements (not with general web development/design related requests as these can be handled by any web developer and we are specialized for VoIP only).

Should I have programmer skills to be able to use the webphone?

No. The webphone can be deployed by anybody. If you already have a website, then you should be able to copy-paste and rewrite the example HTML codes. Some basic Java Script knowledge is required only if you plan to use the Java Script API (although there are copy-paste examples for the API usage also).

What software/service do I need to be able to use/deploy the webphone?

- A webserver (local, rented, hosted) to host the webphone files
- A SIP account by one of the followings:
  - Your existing IP-PBX or softswitch OR
  - SIP account(s) at any VoIP service provider and/or trunk/call-termination services OR
  - Buy a VoIP server software or hardware (Cisco, Mizu, Brekeke, others) OR
  - A free or open source VoIP server (Asterisk, FreePBX, OpenSIPS, others) OR
  - Rent a softswitch (SaaS)
- Optional: Some server side scripts if more customization/changes are required than possible with the webphone API and parameters
- Optionally if you need better control on WebRTC: WebRTC capable SIP server or WebRTC-SIP gateway (both options are freely available)

The webphone can be deployed both in cloud or self-hosted / on-premise.
More details can be found here and in the following FAQ points.

Web server requirements

In short:
Use any web server to host the webphone files. Just copy the webphone folder to your webhost and you are ready to go.
For testing, you can also just launch it from local file or from a localhost development server.

Some more details:
All the functionality of the web sip phone is implemented on client side (JavaScript running in users browser) so there is no any application specific requirements for the webserver. You can **use any web server** software (IIS, nginx, Apache, NodeJS, Java, JSP, others) on any OS (Linux, Windows, others).
The only purpose of the web server is to just host the webphone files and serve it to browsers, the exact same way as a static page or a simple jpg image. No any code will run on your web server.
You can also test the webphone by launching it from localhost (launch any of it's html files such as the index.html, softphone.html or html's from the samples subfolder).

The webphone can be also integrated with any server side framework if you wish (.NET, PHP, java servlet, J2EE, NodeJS and others). Integration tasks are up to you, and it can be done multiple ways such as dynamic webphone configuration per request, dynamic URL rewrite (since the webphone accepts parameters also in URL's), or add more server side app logic via your custom HTTP API which can be called from webphone (for example on call, on call disconnect or other events; the VoIP webphone has callbacks for these to ease this kind of integrations). All these are optional since you can implement any kind of app logic also on client side from JavaScript if you need so.
We recommend deploying the webphone to a secure site (**https**) otherwise the latest Chrome and Opera doesn't allow WebRTC. More details here.
If you can't enable https on your webhost for some reason, then we can host your webphone if you wish on a secure white-label domain for free.

Mime types:
Default web server configurations are usually fine.
Depending on the client browser and the selected engine, the webphone might attempt to download some platform specific binaries. These are found in the "native" folder.

Make sure that your web server allows to download exe files by allowing/adding the .exe extension in your web server allowed mime types to configuration. This might be required if you will need to use the webphone NS engine:

- extension: .exe MIME type: application/octet-stream (or application/x-msdownload)

You can easily test if works by trying to download the NS engine file by typing its exact URI in your browser such as:
http://yourwebsite.com/webphonepath/native/WebPhoneService_Install.exe
(The browser should begin to download the file, otherwise the exe mime type is still not allowed on your webserver or you entered an incorrect path or webserver doesn't serve files from the specified folder.)

Other optional/rarely required mime types that you might enable are the followings:

- extension: .bin MIME type: application/octet-stream
- extension: .mxml MIME type: application/octet-stream (or application/xv+xml)
- extension: .dll MIME type: application/x-msdownload (or application/x-msdownload)
- extension: .jar MIME type: application/java-archive
- extension: .jnilib MIME type: application/java-archive
- extension: .so MIME type: application/octet-stream
- extension: .dylib MIME type: application/octet-stream
- extension: .pkg MIME type: application/x-newton-compatible-pkg (or application/octet-stream)
- extension: .swf MIME type: application/x-shockwave-flash

These are used only by the extra engines (such as the Java engine) which is rarely required (might be used only by ancient or special browsers).
Common mime types such as .zip are not mentioned above since these are usually enabled by default on any web server.

If you wish to avoid any file download requests by the browsers, then you might prioritize/force the WebRTC engine.

You can also test the webphone without a web server by running from local file system.

How to deploy the webphone on my web server?

In short:
Extract (unzip) the webphone.zip and copy the resulting webphone folder to under your server web path and you are ready to use the webphone.

Details:
The webphone is a client side library and, it doesn't have any web server related runtime, thus you can host the webphone as a static website (exactly as you do hosting simple html or png files for example).
Just copy the webphone folder to your webserver path and access any js (to be loaded into your custom html or js files) or html files from there (the included index.html, softphone.html, click2call.html or html's from the samples folder or write your own custom. Of course, you can also use a runtime such as PHP or .NET and integrate the webphone library as you wish).

For example if you copy the webphone to your web directory root folder, then you will be able to test it by typing the following in your browser URL:
https://yourwebdomain.com/webphone/index.html

Note:

- *You can also test the webphone by just launching any html from your local file system (just copy the webphone folder to your desktop or any other location in this case and double click any html to open)*
- *Instead of yourwebdomain.com you might need to enter your web server IP:port if you don't have a domain name, but we recommend acquiring a domain name and setting up a certificate for HTTPS.*
- *Instead of https:// you might have to type http:// if your server doesn't have TLS certificate, but we recommend to upgrade to HTTPS.*

More details:

- Step by step instructions can be found here
- In case if you wish to upgrade from an old version, see the details here
- If you wish to implement some close integration with your web application or web backend, see the details here. (Otherwise the webphone is a client side library and it can be fully controlled by its settings and/or via its JavaScript API)

Why HTTPS?

We highly recommend hosting the webphone files on HTTPS (secure http) to take full advantage of the webphone capabilities.
Otherwise you might run it from local file for testing (modern browsers such as Chrome imposes more restrictions if you run it from unsecure HTTP).

Details:
The webphone can work also from unsecure HTTP hosting.
However to take full advantage of the webphone capabilities, it is strongly recommended to use HTTPS (secure HTTP web hosting using a domain with a valid TLS certificate).

The most important reason is that some browsers (for example Chrome) will not allow media access permissions for WebRTC from unsecured pages.
In this case the webphone can failback to other engines which doesn't required HTTPS (such as the NS engine), however it might be possible that the optimal engine in your environment is WebRTC (or in the worst case it might be the single option).
More details about web hosting can be found here and here.

For development, in case if you don't have a HTTPS hosting yet, you can work in the following ways:

- Use some browser which doesn't require HTTPS for WebRTC. For example old Firefox versions (until v.67.0) should run just fine from unsecured HTTP
- Run from local file (even Chrome should allow WebRTC media permission if you launch the webphone html from a local file, instead of from a web server)

Then only drawback for the above possibilities is that the webphone might have to ask for media permission at each startup as the browsers will save your preference only on HTTPS.

Note: Even if the webphone is running from unsecure HTTP, if the only possible engine is WebRTC in your environment and you are running it in a browser which requires HTTPS for media access (such as Chrome), then the webphone might be able to do automatic HTTP > HTTPS conversion using a proxy service. In this case it might simply reload itself on HTTPS in some circumstances.

Is it working with my VoIP server?

Yes. The webphone works with any SIP capable voip server/softswitch/PBX including Asterisk, FreeSWITCH, Huawei, Cisco, Mizu, 3CX, Voipswitch, Kamailio, Brekeke and many others. You don't necessarily need to have your own SIP server to use the webphone as you can use any SIP account(s) from any VoIP provider.

The web phone is using the SIP protocol standard to communicate with VoIP servers and sofswitches. Since most of the VoIP servers are based on the SIP protocol today the webphone should work without any issue. Some modules (WebRTC and Flash) might require specific support by your server or a gateway to do the translation to SIP, however these modules are optional, gateway software are available for free and also mizutech includes its own free tier service (usable by default with the webphone).

If your are using a private VoIP server (local LAN / private IP address) and you wish to use the webphone (also) from mobile devices, then you should also configure WebRTC since mobile browsers don't have plugin capabilities, thus the Java, NS and Flash engines can't work and for private IP the built-in WebRTC-SIP converter can't work. For more details see here, here and here.

Usually you can setup the webphone the exact same way as you do with a regular IP phone or softphone like X-Lite. Read more details here.

In case if you are using a multi-tenant server with different logical domains for your tenants, then you might have to set both the serveraddress parameter (to your server logical domain name) and the proxyaddress parameter (to your server resolvable domain name or IP address).

If you have any incompatibility problem, please contact webphone@mizu-voip.com with a problem description and a detailed log (loglevel set to 5). For more tests please send us your VoIP server address with 3 test accounts.

Is it working with third-party VoIP servers and VoIP providers?

Yes, the answer is similar here as for the previous FAQ point above.

The webphone is compatible with all regular SIP providers with its default settings. Usually you just have to set the serveraddress/username/password parameters to go. If the provider SIP service is not using (listening on) the standard 5060/5061 ports, then make sure to append the port number for the serveraddress parameter. For example serveraddress: 'sip.myprovider.com:12345'. The serveraddress is usually set in the webphone_config.js while the username/password is usually set at runtime from user input.

Sometime the providers also specify an outbound proxy address to be used which can be set with the proxyaddress parameter.
Sometime the providers requires a specific network transport protocol (instead of the standard default UDP), which can be set by the transport parameter.

Some very rare SIP servers might require some special settings (such as a specific realm) which you can easily set by changing the webphone parameters accordingly. These are extremely rare and in these situations and requirements should be clearly listed on the provider website.

I wish to use the webphone but I don't have a SIP server or service

If you don't have your own VoIP server, you can use any third-party solution or service:

- SIP account(s) at any VoIP service provider and/or trunk/call-termination services OR
- Buy a VoIP server software or hardware (Cisco, Mizu, Brekeke, others) OR

- A free or open source VoIP server (Asterisk, FreePBX, OpenSIPS, others) OR
- Rent a softswitch (SaaS)

There are many SIP servers over the internet where you can create free SIP accounts.
We also provide such a service here: voip service (you can create multiple sip accounts for free and make calls between them)

Server side and connectivity requirements

The webphone is a self-hosted client-side software completely running in the client browser and with no any "cloud" dependencies.
It has the following server side dependencies (all of this controllable by you so you can run the web VoIP bowser plugin on your own also on a private local LAN without the need of any third-party service):

- A webserver where the webphone files are hosted (we send all the required files so it can be hosted on any web-server including servers behind NAT)
  Note: for WebRTC to work (if you need this engine) the webphone have to be hosted on https. (This means that if you run the webphone from local LAN then at least browser CA verification must be enabled to the internet or you have to setup a local valid certificate)
- Optional: connection to custom web application (This is if you have some server side business logic such as .NET or .PHP application or if you are making API calls or using any resources from a custom web application. All these is up to you and it has nothing to do with the webphone itself)
- A SIP compatible VoIP server where the webphone will connect: any SIP server which can be otherwise reached by any sip softphone, including local LAN PBX services
- Optional: helper connectivity services such as WebRTC gateway and STUN/TURN server. All of these can be also disabled/changed and/or the webphone works also if these are not reachable.

What are the main benefits?

Using the Mizu webphone you can have a single solution for all platforms with the same user interface and API. No individual apps have to be maintained anymore for different platforms such as a Windows Installer, a Web application, Google Play app for Android and other binaries.

- Unlike traditional softphones, the webphone can be embedded in webpages while providing the same functionality as a traditional native solution
- Single unified JavaScript API and custom web user interface
- Easy and flexible customization for all kind of use-case (by the numerous parameters and optionally by using the API)
- **Compatible with all browsers** (IE, Firefox, Safari, Opera, Chrome, Edge, etc) and **all OS** (Windows, Linux, MAC, Android, etc)
- Compatible with your existing IP-PBX, VoIP server or any SIP service
- **Works also behind corporate firewalls** (auto tunnel over TCP/HTTP 80 if needed)
- Combines modern browser technologies (**WebRTC**, opus) with VoIP industry standards (**G.729**, conference, transfer, chat, voice recording, etc)
- Use well known VoIP codec's with WebRTC (for example G.729 with WebRTC)
- Easy to use and easy to deploy (copy-paste HTML code)
- Easy integration with your existing infrastructure since it is using the open SIP/RTP standards
- Easy integration with your existing website design
- Proprietary SIP/RTP stack guarantees our strong long term and continuous support
- Support for all the common VoIP features
- Unlike NPAPI based solutions, the webphone works in all browsers (NPAPI is not supported anymore in modern browsers)
- Unlike pure WebRTC solutions, the webphone works in all browsers (webrtc doesn't work in IE and old browsers only with extra plugin downloads)
- Unlike pure WebRTC solutions, the webphone is optimized for SIP with fine-tuned settings (TURN, STUN and others)

Usage examples

The webphone can be used to create standalone VoIP solutions or integrated with any website or web application.
Here are a few examples about how the webphone could be used:

- As a browser phone
- JavaScript SIP library
- Integration with other web or desktop based software to add VoIP capabilities
- A convenient dialer that can be offered for VoIP endusers since it runs directly from your website
- Callcenter VoIP client for agents/operators (easy integration with your existing software)
- Ready to use web VoIP client without the need of any further development
- SIP API for your favorite JS framework such as React, jQuery, Angular, Vue, Ember, Backbone or any others or just plain/vanilla JS
- Embedded in VoIP devices such as PBX or gateways
- Click to call functionality on any webpage
- VoIP conferencing in online games
- Voice assistance over IP
- Hotels
- Buy/sell portals
- WebRTC SIP client, WebRTC SIP library or WebRTC softphone
- Salesforce help button

- Social networking websites , facebook phone
- Integrate SIP client with jQuery, Drupal, joomla, WordPress, angularjs, phpBB, vBulletin and others as a web plugin, module or API
- As an efficient and portable communication tool between company employees
- VoIP service providers can deploy the webphone on their web pages allowing customers to initiate SIP calls without the need of any other equipment directly from their web browsers
- Customer support calls (VoIP enabled support pages where people can call your support people from your website)
- VoIP enabled blogs and forums where members can call each other
- VoIP enabled sales when customers can call agents (In-bound sales calls from web)
- Java Script phone or WebRTC SIP client
- Web dialer for Asterisk, FreePBX or FusionPBX
- Turn all phone numbers into clickable links on your website
- Integrate it with any Java applications (add the webphone.jar as a lib to your project)
- HTTP Call Me buttons
- Remote meetings
- HTML5 VoIP
- Web VoIP phone for callcenter agents integrated with your callcenter frontend
- Asterisk integration (or with any other IP-PBX)
- Convert any SIP link (sip: URI) on web to clickable (click to call) links and replace native/softphone solutions with a pure web solution

The rest is up to your imagination.

Folders and file structure

- js folder: this is for javascript
  - js/lib folder: the webphone core library files. Also there is a string resource file (stringres.js) which contains all the text displayed to the user.
  - js/softphone folder: GUI files. For every jQuery mobile "page" there is an equivalent JavaScript file, which handles the behavior of the page.
- css folder: - style sheets used in skin (GUI). The style of the skin can be changed by editing "mainlayout.css" file
  - css/themes folder: jQuery mobile specific cascading style sheets and images used by the softphone and click to call skin templates
- images folder: images used by the includes skins (GUI)
- oldieskin folder: old webphone skin, it should not be used directly (the softphone might redirect to this skin only very old browsers, ex: IE 6)
- sound folder: contains sound files (for example ringtone and keypad dtmf sounds)
- native folder: platform specific native binaries (the webphone might load whichever needed if any, depending on the engine used)
- the root folder contains the following files:
  - favicon.ico: web page favicon (irrelevant)
  - index.html: a start page for the examples
  - oldapi_support.js: backward compatibility with old skin. Useful for cases where the webphone was integrated using the "old" JavaScript VoIP API.
  - softphone.html: GUI html file for a full featured web phone (use it as-is or further customize this after your needs)
  - click2call.html: GUI html file for a click to call implementation (use it as-is or further customize this after your needs)
  - softphone_launch.html: just a placeholder for the “softphone.html” so you can use the softphone skin more comfortably (centered) when launched directly
  - webphone_api.js: the public Javascript API of the web phone
  - webphone_config.js: webphone configuration
  - package.json: might be useful if you are using package managers
  - firebase-messaging-sw.js: used only if push notifications are set
  - your_custom_sip_client.html: just an empty html to remind you that you can create your own GUI
- The “samples” folder contains other usage examples/skins/samples:
  - techdemo_example.html: a simple demo html the showcase the webphone functionalities
  - minimal_example.html: shortest implementation to make a call
  - basic_example.html: simple usage example of softphone SDK
  - api_example.html: a more detailed example for the SIP API usage
  - incoming_example.html: simple example to handle incoming call
  - click2call_example.html: a minimal click to call phone button implementation
  - linkify_example.html: will convert phone numbers on a webpage to clickable SIP URI’s for click to call
  - mobile_example.html: just a show case to remind you that the webphone also works on smartphones
  - multipage_example.html: demonstrates using the same webphone instances across multiple pages

It is possible to delete the unneeded files (for example you can delete the softphone and the oldieskin folders if you are using the webphone as an API), however you should not bother too much about these and just leave all the files on your server. This can’t have any security implications; the webphone will use only the required files for your use-case. See the webphone_shrink document for more details.

Why the webphone package size is so big?

Webphone package size including all engines and documentation: ~34MB.
Endusers browsers download size first time: ~1MB
Endusers browsers subsequent usage: ~0 – 300 KB

The downloadable webphone package (webphone.zip or yourbrand.zip) is around 26 MB because it contains all the webphone related files, including samples and documentation. It will take around 34 MB space on your web server when unpacked (if you don't remove any unnecessary files), but only a tiny fraction of this have to be downloaded by browsers.
The webphone package includes also native binaries per platform which might be used in some circumstances (NS engine plugin one-click installer), but in this case only the user device specific binaries are to be downloaded (for example on 64 bit linux, only the webphone_ns_lin64.zip might be downloaded).
In case if the webphone uses the WebRTC engine, then there is no any native binary download required. The WebRTC stack will be provided by the browsers in this case and the webphone itself will be responsible mainly for the JS SIP signaling using the WebSocket Protocol as a Transport for SIP (RFC 7118).

Your customers browser will have to download only a few small files to launch the webphone at first time (usually js/css/html between 100 KB – 2 MB in total, depending on the engine and skin to be used). In some circumstances there might be an additional 1-7 MB download (depending on the OS and if using a browser plugin/engine).
We consider this as negligible, since if the enduser network connection is not fast enough to download the required webphone library files very quickly, then most probably it will also have degraded VoIP experience.
Of course, the whole webphone library can be also loaded asynchronously, unnoticeable for the endusers.
All the webphone files can be cached by the browsers minimizing the download size required at subsequent launch.
Downloading the webphone SIP library by the browsers takes around 2 seconds at first time using a typical internet connection and below 1 second on subsequent launches.

Additional plugin download?

An additional download by the users might be required only in some specific circumstances. For example if the browser can't use WebRTC or the Java engine, then the webphone might offer the NS engine plugin download which is an one-click installer.

These downloads are NOT from some remote/third-party locations but from your own webphone package (the webphone files what you copied to your web server).

The webphone includes multiple VoIP engines and the "best" one is automatically selected based on circumstances (settings/server capabilities/browser capabilities/etc).
The most important engines are the followings:

- WebRTC: no any download is required; will act as a standard WebRTC SIP client library
- NS engine: native VoIP engine browser plugin one-click installer download is always required
- Java: no any download is required as this is used only in browsers with java applet support if JRE is already found/installed
- Flash: no any download is required and might be used in some rare occasions with for very old/outdated browsers if Flash is already found/installed

If you must avoid any additional download, then make sure that WebRTC can be used in your environment. Otherwise the NS engine can also offer a top quality VoIP experience at the cost of two additional click required by the enduser (one click to download and one click to install).

Does the webphone depends on Mizutech services?

No, the webphone can be used on its own as a fully self-hosted solution, connecting to your VoIP server directly by SIP (Java, NS and App engines), via WebRTC or via Flash so you will have a solution fully owned/controlled/hosted by you, without any dependency on our services, third-party services or cloud services.

With other words: if all our servers will be switched off tomorrow, you will be still able to continue using our web softphone. You can also use in a private network, even without internet access (hosted on your internal Web server and connecting to your internal SIP server).

However, please note that by default the webphone with its default settings might use some of the services provided by Mizutech or third party cloud services for free, to ease the usage, and to make it a turn-key solution with optimal NAT handling, without any extra settings to be required from your side. All of these are for your convenience, especially to help beginners in the VoIP fields to handle some corner cases.
Most of these are used only under special circumstances and none of these are critical for functionality; all of them can be turned off or changed and the webphone can work also if these are not set or services are not available.

The following services might be used:

- Mizutech license service: demo, trial or free versions are verified against the license service to prevent unauthorized usage. This can be turned off by purchasing a license and your final build will not have any DRM or "call to home" functionality and will continue to work even if the entire mizutech network is down.
  Note: this is not used at all (completely removed) in paid/licensed versions
- WebRTC to SIP gateway: if your server doesn't have WebRTC capabilities but you enable the WebRTC engine in the webphone (using it as a JavaScript WebRTC client) then it might use the Mizu WebRTC to SIP gateway service. Other possibilities are listed here.
  Note: this might be used only if you are using the webphone WebRTC engine but your server doesn't have support for WebRTC nor you have a WebRTC-SIP gateway.
- Flash to SIP gateway: rarely used (only when there is no better engine then Flash such as ancient browsers). Just turn it off (by setting the "enginepriority_flash" parameter to 0) or install your own RTMP server and specify its address.
  Note: usually Flash is not used at all as there are better built-in engines which are supported by more than 99.9% of the browsers.

- Push notifications: Mizutech might provide push service (free tier) in case if you need push notifications but your SIP server don't have support for it. More details can be read here.
  Note: push notifications are disabled by default (you can enable it with the "backgroundservice" parameter) and there is no need to use the mizu service (you can use your SIP server push service or via any other third party gateway if needed)
- STUN server: in some circumstances the webphone might use the Mizutech STUN service. You can change this by changing the "stunserveraddress" to your server of choice (there are a lot of free public STUN services or you can run your own: stable open source software exists for this and it requires minimal processing power and network bandwidth as STUN is basically just a simple ping-pong protocol sending only a few short UDP packets and it is not a critical service).
  Note: you can use the webphone without any STUN service if your SIP server has basic NAT handling capabilities and it is capable to route the RTP if/when needed.
- TURN server: in some circumstances the webphone might use the Mizutech TURN service which can help firewall/NAT traversal in some circumstances (rarely required). You can specify your own turn server by setting the "turnserveraddress" parameter (if TURN is required at all).
  Note: you can use the webphone without any TURN service if your SIP server has basic NAT handling capabilities and it is capable to route the RTP if/when needed.
- Alternative IP lookup: in some circumstances the webphone might try to auto-detect its public IP address or the server IP (xhosttoip) by a service such as mnt.mizu-voip.com, checkip.dyndns.org, wtfismyip.com, icanhazip.com, my-ip.heroku.com, checkip.dyndns.org or ipinfo.io. This is just an extra way to check if the SIP server is public or to detect the correct external IP which might be useful in some rare circumstances (SIP servers with poor NAT support when STUN is unavailable or bogus so at the first request the webphone might be already able to present its public address). The failure of this service does not affect the SIP stack functionality.
  Note: you can disable this by setting the altexternpublicip parameter to 0.
- Network connectivity: if the SIP stack can't connect to your server or network connectivity have been lost, the webphone might ping some well know addresses such as google.com to see if there is any network connection at all. When using WebRTC, the webphone might check the internet connectivity against some public websocket servers. (it can be reconfigured with the wstestservers parameter; multiple servers can be separated by comma).
  This "ping" is then used only to clarify the connectivity problem reports and print appropriate logs to the browser console to ease the troubleshooting (to separate "No network" from other possible issues such as "SIP server is offline").
  This might be used only if your SIP server is public (not checked if on the same LAN) and the websocket connectivity test is performed only if the loglevel is higher than 2.
  Note: You can disable it by setting the networkchecks parameter to 0.
- JSONP: if you set some external API to be used by the softphone skin (such as for user balance or call rating requests) and your server can't be contacted directly with AJAX requests due to CORS, then the API calls might be relayed by a JSONP or websocket proxy. To disable this, make sure that the domain where you are hosting the web phone plugin can access your domain where your API is hosted.
  Note: this might be used only in very specific circumstances which can be avoided (used only when you integrate the webphone with your own API, but your own API isn't configured correctly so it can't be accessed by the webphone via normal AJAX GET/POST requests)
- HTTPS proxy: with the WebRTC engine if you are using the webphone from browsers which require secure page for WebRTC (such as Chrome) and your website is not secured (not https) then the webphone might reload itself via a HTTPS proxy. To disable this, host your webphone on HTTPS if you wish to use WebRTC from Chrome. API requests can be also routed via this service (such as credit or rating requests) if you are running the webphone on HTTPS but defined your SIP server API as HTTP (otherwise browser blocks requests from secure page to insecure resources)
  Note: this might be used only if your website is not on HTTPS (no SSL certificate) and you try to use the webphone WebRTC from a browser which disallows unsecure WebRTC. It can be disabled by setting the "usemizutlsproxy" parameter to 0 (0=no, 1=auto, 2=yes).
- Tunneling/encryption/obfuscation: In some conditions the webphone might use the Mizu tunneling service to bypass VoIP blockage and firewalls. This is usually required only in countries where VoIP is blocked (such as UAE or China) or behind heavy firewalls with DPI and you can turn it off by setting the "usetunneling" parameter to 0.
  For the usual encryption just setup HTTPS for your website hosting the webphone and use the webphone built-in encryption capabilities such as SIPS/TLS/SRTP or WebRTC/DTLS/SRTP as these doesn't require any external service to be used (except support by your softswitch).
  Note: this is a special feature which needs to be turned on by mizutech support, otherwise it is not enabled by default.
- Geolocation: the webphone doesn't try to ask for Location permissions, but at first start it might try to detect its location by a lookup via a random free service such as ip-api.com, ipinfo.io or www.geoplugin.net from a low priority background thread. This is just for your convenience so on your server side you can easily store the client location as this information is just sent by X-Country SIP header. By default this is turned off on private networks.
  Note: this feature has nothing to do with the SIP core functionality and it can be disabled by setting the geolocation parameter to 0.
- Connections to nsX.webvoipphone.com.
  The nsX.webvoipphone.com domain points to localhost (127.0.0.1) and it is used to allow the webphone (running in a local browser page) to securely communicate with the locally running NS engine (some browsers allow raw connections to 127.0.0.1 but others require secure connections).
  By default the webphone uses the ns4.webvoipphone.com and maintains a Let's Encrypt certificate for it. You can rewrite the domain name if you wish with the nsdomain webphone parameter (it can be any domain name owned by you with its DNS A record set to 127.0.0.1). If you change the domain then you must also change the TLS certificate (the .pem files in the NS app folder) and set the tlscert_autoupgrade parameter to 0 in the NS engine .ini file(s).
  Note: you can rewrite the domain if you wish with the nsdomain parameter as described above.
- NS installer: The NS engine might use a JVM which is not included in the installer to keep it as small as possible. If the enduser already has a Java installed on their device/PC then the NS engine will use that Java virtual machine. Otherwise the installer will auto download a tiny JVM of around 9 MB or the system default JVM (a minimal JRE/JDK).
  Note: this is used only by the NS engine installer and if you wish, we can include the JVM into the installer instead (mbuildoption_jrebundle mbuild option)
- NS engine TLS cert: The webphone communicates with the NS engine via WSS (secure websocket) on localhost using the ns4.webvoipphone.com domain with an automatically maintained Let's Encrypt TLS certificate. The certificate is refreshed time to time from C:\Mizutech\WEB\webvoipphone.com\publicfiles\localhostcert2
  Note: this is used only by the NS engine in some browsers which doesn't allow simple HTTP or TLS localhost connections
- Auto upgrade: the native components can auto-upgrade itself from Mizutech download service. This is enforced only from known old versions to know good versions. You can disable this by setting the "autoupgrade" to 6. (You can also set the "autoupgrade" to 5 which will also disable the upgrade of the SSL certificates if any)
  Note: this might be used only if you use the webphone NS engine and can be turned off as described above.
- Send log to support: the softphone skin (softphone.html) has a "Send log to support" option on its menu which is a convenient way for endusers to report any issue. By default this uploads the log file to mizutech support, but you just need to rewrite the logform_action URL as described in the Logs FAQ to change to yours
  Note: this is present only in the softphone skin and it can be removed or rewritten to upload logs to your server

If you are using the webphone on a local LAN then these services are not required and are turned off automatically (so the webphone will not try to use these if your VoIP and/or Web server are located on local LAN / private IP).

If you need to white-list (or block for some reason) our servers, here is the address list associated with the above services:
mnt.mizu-voip.com, rtc.mizu-voip.com, usrtc.webvoipphone.com, usrtc3.webvoipphone.com, usrtcx.webvoipphone.com, www.webvoipphone.com, www.mizu-voip.com, fcm.webvoipphone.com;
107.174.212.78, 148.251.28.176 / 28 (148.251.28.177 - 148.251.28.190)

(In case if you can set domain name based firewall rules, that is better, otherwise you can just use the IP addresses -all of these are static fix IP's and it is not expected to be changed anytime soon)

Note: blocking the license service for demo or trial webphones will make them unusable after some time (But licensed/paid versions doesn't depend on any central license service at all).

How to configure the webphone

The most important settings are listed here.

The webphone can be configured by its parameters or dynamically via the setparameter API. There are many ways to set its parameters. You can statically hardcode them in the webphone_config.js file, pass as URL parameters or load from a server API by setting the scurl_setparameters to point to your API (HTTP AJAX URL). For more details about how you can pass the parameters for the webphone, see the beginning of the parameters chapter.
Make sure to configure it correctly for your SIP server (this usually means only setting the serveraddress parameter correctly).

You can make a quick test with your SIP server using one of the included html (such as the softphone.html or techdemo.html).
Once the webphone has the correct configuration, you can go ahead using one of the included solution (such as the softphone or click to call; you can also modify these as you wish) or build your own custom solution which usually involves creating your custom HTML/CSS and interacting with the webphone with JavaScript using its API.

VoIP engine

The webphone includes multiple VoIP engines (such as NS engine, WebRTC, Java, etc), each with their own advantages and disadvantages.
Having multiple engines allows broad compatibility with different browsers/servers and better resource utilizations. For example if you are operating a call-center then you might force the NS engine to avoid WebRTC utilization or if your solution is deployed for wide range of customers connecting over the internet then you might prefer WebRTC for a better user experience (no need to download/install anything by the endusers).
By default the webphone will select the most optimal engine for your environment. This process can be changed by the enginepriority parameter.
More detail can be found here and here.

How to handle WebRTC?

WebRTC is one of the built-in VoIP engine in the webphone among others such as the NS or Java engines.
In most desktop browsers (Windows/Linux), the webphone can work also without WebRTC, however in mobile browsers (where external plugin install is not available) it should prefer the WebRTC whenever possible.
By default you don't need to make any configuration changes for the webphone to work for you. If possible (supported by the OS/browser/server/gateway/peer) the WebPhone will be able to use WebRTC. Otherwise it will prefer the NS or Java engine and the enduser should not notice any difference. The webphone will automatically detect engine availability on startup and it will use the best possible engine by default when more than one engines are available.

The main advantage of the WebRTC engine is that it is more convenient for occasional endusers since it doesn't need any browser plugin to be downloaded/installed. The main "disadvantage" is that this is an extra protocol above raw SIP/RTP, which must be understood on the client and server side.
When the webphone selects the WebRTC engine, then it will work as a standard JavaScript WebRTC SIP library and it can't connect directly to a server side SIP stack (if your SIP server doesn't have built-in WebRTC support), but a conversion of the WebRTC protocol (coming from the browser) to SIP (which is understood by your VoIP server or IP-PBX) will be required.
This conversion is done on the client side (the webphone is sending SIP packets in Websocket which needs to be converted to SIP UDP/TCP/TLS and it will emit a media stream compatible with SIP servers) and on the server side (the server, proxy or gateway needs to understand WebRTC, ICE and decrypt DTLS/SRTP into raw RTP packets).

The engine selection will be handled for you automatically by default, without any further configuration required. However if you need more control, you have several **options** to deal with WebRTC:

1. Don't use WebRTC at all.

Most JavaScript SIP clients implements only the WebRTC protocol, but the webphone includes also other engines, which can be used most of the time from desktop browsers. However, please note that there are circumstances (mobile browsers) when the only available engine is WebRTC. We recommend to test it with your use-case before to disable WebRTC.
To deprioritize WebRTC, set the enginepriority_webrtc setting to 1 (or to 0 to disable it; or set the enginepriority_ns to 4; or both).
In this case on the next start the webphone should offer the NS engine download. If somehow this is not happening, then you might install it manually on your test PC (because the WebPhone might try to use the previously working engine by default). On Windows just launch the WebPhoneService_Install.exe from the webphone\native folder. More details here.

2. Check if your VoIP server already has WebRTC support. Most modern VoIP server already implements WebRTC (including mizu VoIP server, Asterisk and others) or you might need to add/enable a WebRTC module on your server for this, so chances are high that your VoIP server can already handle WebRTC natively. Ask your VoIP server developers/support if you are not sure.
   In this case you will have to set the webrtcserveraddress setting to point to your server WebRTC websocket listener. More details here.
3. Use the free Mizutech WebRTC to SIP service tier. This is enabled by default and it might be suitable for your needs if you don't have too much traffic over WebRTC. The webphone might automatically start to boost the priority for other engines when you are over the free quote.
4. Use the WebRTC to SIP gateway software (MRTC). If you are running a more serious VoIP service and wish/must use WebRTC most of the time, but your VoIP server don't have WebRTC support and you don't wish to use the free service, then you can purchase a dedicated commercial license for the WebRTC-SIP gateway. This includes also support and we can install and configure it for you if you send remote access (RDP) to your machine where you wish to host it. It can be installed also on multiple servers for high-availability and used with multiple SIP servers/domains, depending on your license.
5. Use any third party WebRTC to SIP gateway: There are a few free and open-source alternatives, which are capable to do this task for you, including Janus and Dubango. The webphone is compatible with all standards based SIP/WebRTC server or gateway.
   (However if you don't have any of these installed yet, then we recommend our own gateway mentioned above as it is proven to be more reliable than the open-source alternatives and we can also provide full support for it)
6. Use the Mizutech WebRTC to SIP paid service. We can provide dedicated WebRTC to SIP conversion services for a monthly fee if required. Contact us if you are interested in this.

The webphone can be used as a JavaScript WebRTC SIP client library or as a WebRTC softphone by increasing the enginepriority_webrtc to 3 or 4 (in this case it will use the other engines only when WebRTC is not supported by the browser).
Important: Modern browsers require secure connection for both your website (HTTPS) and websocket (WSS) to allow WebRTC (media permissions).
This means that if your page is not on HTTPS, then you must use some old browser such as Firefox v.67.0. to test the webphone WebRTC capabilities.
Alternatively, you might test the webphone by just launching from local file system which allows WebRTC in most browsers.

The following are the **conditions** for the webphone to be able to use WebRTC:

- You must run the webphone from a WebRTC capable browser. All the popular modern browsers has WebRTC support (Chrome, Firefox, Edge, Safari, Opera, etc).
- The webphone must be hosted on secure HTTPS with valid SSL certificate installed on your Web server
- One of the following must be true:
  - Your SIP server(s) has built-in WebRTC capabilities and the "webrtcserveraddress" webphone parameter configured accordingly
  - or Your SIP server(s) has a public IPv4 address, reachable from the public internet, so the webphone can use our free WebRTC-SIP service
  - or You are running a WebRTC-SIP gateway to handle the protocol conversion (our MRTC gateway or any third-party solution such as the open source Doubango or Janus)

How to configure WebRTC with my WebRTC server or gateway?

In short:
Just set the webrtcserveraddress parameter to point to your websocket listener URL.

Details:
The webphone can be also used as a JavaScript WebRTC SIP client library (with a possible failback to other SIP engines when WebRTC is not available).
By default the webphone can also work without WebRTC support (using native SIP/RTP by the NS or Java engine) or it can use our WebRTC-SIP gateways (if WebRTC is preferred/needed in your environment and you haven't configured it yet with your WebRTC server settings). These will work with the default settings without the need to change the webrtcserveraddress or any other related parameters.
If for some reason this default behavior isn't suitable for you and you wish to setup your own WebRTC service to be used, then you have two possibilities:

- Configure WebRTC capabilities for your server:
  If your softswitch/PBX has WebRTC support, follow its documentation to configure WebRTC properly. For Asterisk you can find a guide here (similar configuration can be used also for FreePBX, Elastix and other Asterisk based solutions)
- Or use a separate WebRTC-SIP gateway:

If your softswitch/PBX doesn't have any WebRTC support, then you can setup a WebRTC-SIP gateway near your softswitch/PBX such as Janus, MRTC or Dubango.

Of course, if your VoIP server has built-in WebRTC support, then you should also configure the webphone accordingly.

Once your WebRTC service is running, you need to set the following configurations for the webphone:

- webrtcserveraddress (mandatory)
- stunserveraddress (optional but highly recommended when using the webphone over the public internet)
- TURN related settings (optional but recommended when using the webphone over the public internet)
  (or you can use the ice parameter to set both the STUN and TURN parameters)

For STUN and TURN you can use any TURN server such as coturn. (if your WebRTC server doesn't have this built-in. Our MRTC gateway has built-in STUN and TURN, but most third party WebRTC servers doesn't have this built-in and you need to run a separate TURN server).

Example configuration:

```
(You can set these in the webphone_config.js file)
serveraddress: 'sip.yourdomain.com', //Your SIP server address or domain name (append the port if not 5060. For example: '11.22.33.44:5678')
webrtcserveraddress: 'wss://rtc.yourdomain.com/anypath',  //This is your websocket URL socket address (WS for unsecure, WSS for secure. Append also the port if not on the default 80/443. For example: 'ws://11.22.33.44:5555/anypath'. Note: your SIP server and WebRTC websocket listener might use the same IP or domain, but usually different ports)
stunserveraddress: '22.33.45.55:3478',  //Your STUN server address (Optional)
turnserveraddress: '22.33.45.55:3480',  //TURN server address (Optional. Note: TURN and STUN might use the same port.
turnparameters: 'transport=tcp', //TURN parameter (set this only if you wish to use TURN over TCP only)
turnusername: 'turnuser', //TURN username if required by your TURN server
turnpassword: '***', //TURN password if required by your TURN server
enginepriority_webrtc: 3, //Increase the priority for the WebRTC engine a bit
loglevel: 5 //Debug trace level
```

With these settings the webphone will use your WebRTC service and you can begin to initiate/receive calls over WebRTC. Check your WebRTC server logs and the browser console log if you run into any issue. If you used the NS or Java engine before with the softphone skin, then you might need to go to the settings and set the VoIP Engine to WebRTC or clear your browser cache (This might be required, because the webphone might remember and prefer the previously used engine).

Note:

- Modern browsers require secure WebSocket connections (WSS), otherwise it will not allow WebRTC, so for production we highly recommend to set a proper SSL certificate for your websocket listener.
- You can test your server websocket connectivity here (check if can connect, you might not receive any answer for the requests sent if your server ignores non-SIP requests)
- For tests without SSL, you can use old Firefox (v.67) browser as this allows WebRTC calls also on unsecure websocket (WS) or launch the webphone from localhost/local file.
- Your WebRTC server side software must be able to understand the websocket protocol for SIP as described in RFC 7118.

How to handle Flash?

First of all it is important to mention that the browser web phone works just fine without Flash and Flash is already a deprecated technology.
Chances are high that you don't need Flash at all even if available. In the rare circumstances when the only usable engine would be Flash only (some very outdated or special browsers), the webphone can automatically use the Mizutech Flash to SIP free service. In case if somehow you wish to drive all your traffic over Flash, then you might install a Red5 server (open source free software) to handle the translation between RTMP and SIP/RTP, then set the rtmpserveraddress to point to your flash media server and increase the value of the enginepriority_flash setting.

How to handle Java, Native and App engines?

These engines doesn't need any special server side support and they works with old legacy SIP (all SIP servers) without any extra settings or software. When the browser VoIP plugin uses one of these engines, there is a direct connection between the engine (running in the user's browser) and your VoIP server, without involving any intermediary relay (Actually RTP can also flow directly between the endusers, bypassing your SIP server. This is up to your server settings and its NAT handling features).

- NS: The NS engine requires a one-click install process to be performed by the endusers. You can prioritize the NS engine by setting the enginepriority_ns parameter to 4. (And you can prioritize other engines by setting the corresponding enginepriority_xxx parameter to 4 where xxx is the engine name. See the enginepriority parameters for more details)
- Java: If you wish to force the usage of Java (which can offer top quality VoIP), then make sure to install the JRE from here (if not already installed on your system) and use Internet Explorer (any version), Pale Moon or old Safari or Firefox below version 51 as Edge and new Chrome versions doesn't have Java applet support anymore.
- Secondary engines:

  These are failback mechanism when the webphone can't launch any VoIP stack.
  - App: Launch your app when VoIP fails. You might review/set the following related parameters: appengine_startat, android_nativedialerurl, ios_nativedialerurl, app_protocol
  - P2P and callback: initiate a callback or p2p call when VoIP fails. Set the accessnumber and/or the callbacknumber parameter if your server has such features.
  - Native Dial: launch a normal phone call when VoIP fails. Related parameter: app_protocol

What are the advantages over pure WebRTC solutions?

First of all, please note that the webphone contains also a top quality WebRTC stack, thus it can be used as a JavaScript WebRTC SIP client library. However, it includes also other engines (such as NS and Java) which works directly over plain legacy SIP/RTP using the traditional SIP protocol standard, without the need for any WebRTC capabilities.

WebRTC is becoming a trendy technology but it has a lot of disadvantages and problems:

- It is a moving target. The standards are not completed yet. Lots of changes are planned also for the upcoming years. For example Edge added a different "ORTC" implementation and then moved to Chrome engine.
- Incompatibility. WebRTC has known incompatibility issues with SIP and there are a lot of incompatibilities even between two WebRTC endpoint as browsers has different implementation and different codec support.
- Not supported by all browsers. No support in IE and old browsers. No support on old iOS and MAC (except with extra plugin downloads). No support on older Android phones.
- Lack of popular VoIP codec such as G.729 which can be solved only by expensive server side transcoding
- Weak configuration support. It is a black-box in the browser with browser specific bugs and a restrictive API. You have little control on what is going in the background.
- Missing functionalities: for example it is not possible to change to speakerphone on iOS/Safari.
- Extra server side processing required for TLS/Websocket/DTLS/SRTP and error prone WebRTC to SIP conversion.
- A WebRTC to SIP gateway required if your VoIP server don't have built-in support for WebRTC.
- Adds unneeded extra complexity. The server has to convert from the websocket protocol to clear SIP and from DTLS to RTP.

Luckily the Mizu webphone has some more robust engines that can be used without these limitations and it can prioritize these over WebRTC whenever possible, depending on available browser capabilities and user willingness. (Non-obtrusive notification might be displayed for the enduser when a better engine is available or if a user can upgrade with one-click install).

One of the advantages of the Mizu webphone is that it can offer alternatives for WebRTC, so you can be sure that all your VoIP users are served with the best available technology, regardless of their OS and browser.

However we do understand that WebRTC is comfortable for the endusers as it doesn't require any extra plugin if supported by the browser. The mizu browser phone takes full advantage of this technology and we provide full support for WebRTC by closely following the evolution of the standards. We have invested a lot of time and effort to handle the above WebRTC related inconveniences as smoothly as possible incorporating a robust and full featured WebRTC stack into the webphone, capable to handle most edge cases.

With a WebRTC only solution you would miss all the benefits that could be offered by a standard SIP/RTP client connecting directly to your VoIP server with native performance, full SIP support including all the popular VoIP codec and without the need for any protocol conversion, directly from enduser browser.

Known limitations

The webphone by default will try to handle all the differences, shortcomings and restrictions in different environments to provide an unified API and consistent experience across all browsers, engines and operating systems. However there are some operations, which might not work correctly for your use-case, depending on your SIP server, WebRTC gateway, OS, browser and the used webphone VoIP engine.
Please test with the demo version first before to purchase a license, to make sure that it fulfills your requirements and it works correctly in your environment.

The followings are the notable limitations:

- Not all the listed features are available from all engines (the webphone might automatically handle these differences internally)
- The webphone doesn't work when Private Browsing or Incognito mode is enabled (because no outbound WebSocket connections are allowed when private browsing is enabled)
- Relative paths from iframes are not working in old Internet Explorer versions such as IE8. The demo index.html are using iframes to load the different examples and the upper folder (../) from the paths are not working in this case (for example in line <script src="../webphone_api.js"></script>). As a workaround, the webphone examples are opened in a different window/tab from IE8.
- Some platforms currently have very limited VoIP support available from browsers. The most notable is old iOS releases where the default browser (old Safari versions) lacks any VoIP support. The webphone tries all the best to work around about these by using its secondary engines offering call capabilities also for users on these platforms
- Audio output devices can't be set programmatically in some browsers if you are using WebRTC because navigator.mediaDevices.enumerateDevices() and getUserMedia() returns only input devices (no output device is ever returned). In these situations it is expected that the user might change the audio device to be used from the WebRTC audio permission popup. See the Firefox bug report for more details (marked as Fixed but still bogus). Also iOS browsers don't have access at all to audio devices.
- Old Android Chrome browsers might use the speaker (speakerphone) for audio. This might affect only the WebRTC engine on old Android phones/tablets and you will have normal audio output if using the App engine on Android. Otherwise it can be changed with the androidspeaker parameter.
- Volume cannot be changed programmatically if you are using the WebRTC engine in Chrome and Firefox (use the OS volume settings)
- For chat/IM to work your server have to support SIP MESSAGE as described in RFC 3428 (Supported by most SIP servers. See this section for more details)
- In case if you wish to use BLF, then you should use the NS or Java engine as BLF for WebRTC is in beta only
- WebRTC media negotiation issues with simultaneous calls. As a workaround, delay the second/third/etc call with 2 seconds. Only the WebRTC engine is affected by this.
- JRE audio handling is not so robust on Linux and you might encounter audio open issues in some Linux/JRE/audio system combinations when using the NS or Java engine.
- No support is provided for the AEC functionality. AEC is not capable to remove echo in all circumstances (after average statistics it is capable to eliminate more than 90% of the echo with around 90% success rate; the success rate depends on many factor such as device, audio driver, engine, room, network, peers, environment, settings).
- Not all the listed codec's are supported by all engines. If you are using the webphone as a JavaScript WebRTC library, then codec support is up to the browser. More details here.
- Video is implemented only with the WebRTC engine (the webphone will auto-switch or auto-offer WebRTC whenever possible on video request). WebRTC-SIP video calls works only if there are a compatible codec on SIP side (H.264 or VP8). WebRTC-WebRTC video calls are expected to work in all circumstances between same browsers if the users have a camera device. Video re-INVITE will not work with the webphone (use with SIP devices which initiate or accept the video from start)
- There is no video support on old Safari versions (video is supported from iPhone 7 / iOS 11 / Safari 11)
- While video should work via WebRTC on all supported platforms, we recommend testing it first in your environment before to purchase if this feature is important for you. Some SIP servers and devices doesn't handle the video features correctly and this is very difficult to debug (We can't offer support for extra video features. It either works or not works in your environment depending on your SIP server and SIP peers capabilities).
- Screen sharing might require extra plugin and might not work in all browsers (it was tested and works with both Chrome and Firefox but due to API changes we can't guarantee this functionality in all circumstances)
- Mizutech doesn't provide direct support for the video functionality with third-party devices, because of the many bogus implementations and codec fragmentation. You will have the maximum chance for the success if both parties are connected by WebRTC
- Conference support with local RTP mixer is implemented only in the NS and Java engines. While the enduser is on WebRTC, then local conference mixer is not available which means that conference can be done only via server side dtmf control or conference rooms between webphones or by auto switching to NS when available. Read more here.
- The NS engine for macOS have been revoked because latest macOS versions doesn't allow audio recording from services. It might be reintroduced later after a rewrite to run it as regular app instead of a service. You can use any other supported engine on MacOS such as the WebRTC engine (handled automatically by default)
- Some features might not work correctly between WebRTC and SIP. This is not a webphone limitation, but it depends completely on server side (your softswitch or gateway responsible for WebRTC-SIP protocol conversion) and for the loss of direct mapping between WebRTC and SIP protocol capabilities
- Some features require also proper server side support to work correctly. For example presence, chat, conference, video, call hold, call transfer and call forward. See your VoIP server documentation about proper setup

OS/browser related issues

There are many browser and OS related bugs either in the browser itself or in the plugins used for VoIP (native/webrtc/java/flash). Most of the issues are handled automatically by the webphone by implementing workarounds for a list of well-known problems. Rarely there is no any way to circumvent such issues from the webphone itself and needs some adjustment on server or client side.

Some chrome versions only use the default input for audio. If you have multiple audio devices and not the default have to be used changing on chrome, Advanced config, Privacy, Content and media section will fix the problem.

Some linux audio drivers allow only one audio stream to be opened which might cause audio issues in some circumstances. Workaround: change audio driver from oss to alsa or inverse. Other workarounds: Change JVM (OpenJDK); Change browser.

Incoming calls might not have audio in some circumstances when the webphone is running in Firefox with the WebRTC engine using the mizu WebRTC to SIP gateway (fixed in v.1.8).

Java related:

These details are important only if for some reason you might wish to force the Java engine. However please note that these java related issue are not real problems since when possible the webphone uses WebRTC or NS engines, especially as Java is not available in latest Chrome and Firefox anymore.

If the java (JVM or the browser) is crashing under MAC at the start or end of the calls, please set the “cancloseaudioline” parameter to 3. You might also set the "singleaudiostream” to 5. If the webphone doesn’t load at all on MAC, then you should check this link.

One way audio problem on OSX 10 Maverick / Safari 6/7 when using the Java engine: Safari allows users to place specific websites in an "Unsafe Mode" which grants access to the audio recording. Navigate to "Safari ->Preferences -> Security (tab) and tick "Allow Plug-ins" checkbox. Then depending on safari version:
-from the Internet plug-ins (Manage Website Settings)" find the site in question and modify the dropdown to "Run In Unsafe Mode".
-or go to Plug-in Settings and for the option "When visiting other websites" select "Run in Unsafe Mode". A popup will ask again, click "Trust"
You will be asked to accept the site's certificates or a popup will ask again, click "Trust". Alternatively, simply use the latest version of the Firefox browser.

Java in modern browsers is not supported anymore (the webphone will select NS or WebRTC engine by default).

If for some reason you still wish to force Java, then in Chrome versions prior September 1, 2015 it can still be re-enabled:
Go to this URL in Chrome: chrome://flags/#enable-npapi (then mark activate)
Or via registry: reg add HKLM\software\policies\google\chrome\EnabledPlugins /v 1 /t REG_SZ /d java

Java can be also re-enabled in Firefox 52 by configuring the Firefox setting plugin.load_flash_only to false.

(By default the webphone will handle these automatically by choosing some other engine such as WebRTC unless you forced java by the engine priority settings)

The webphone is not loading

Symptoms:

- If your html can’t find the webphone library files you might see the following errors in your browser console:
  - Failed to load resource: …/js/lib/api_helper.js
  - ReferenceError: webphone_api is not defined
- If not supported by browser or your webserver doesn’t allow the required mime types, then the page hosting the webphone might load, but you will not be able to make calls (VoIP engine will not start)

Fixes:

- Missing library: Make sure that you have copied all files from the webphone folder (including the js and other sub-folders)
- Browser support: Make sure that your browser has support for any of the implemented VoIP engines: either Java or WebRTC is available in your browser or you can use the NS engine (on Windows, MAC and Linux) or the app engine (on Android and iOS)
- Web server mime settings: Make sure that the .jar and .exe mime types are allowed on your webserver so the browsers are able to download platform specific native binaries
- HTTPS: Set a SSL certificate for your website for secure http, otherwise WebRTC will not work in chrome
- Lib not found: If your webphone files are near your html (in the same folder) then you might have to set the webphonebasedir parameter to point to the javascript directory

  webphonebasedir
  This setting is deprecated after 1.9 as the webphone should automatically detect its library path automatically.
  If the html page, where you are including the webphone, is not in the same directory as the webphone, then you must set the "webphonebasedir" as the relative path to the webphone base directory in relation to your html page.
  The base directory is the "webphone" directory as you download it from Mizutech (which contains the css,js,native,... directories).
  For example if your page is located at http://yoursite.com/content/page.html and the webphone files are located at http://yoursite.com/modules/webphone then the webphonebasedir have to be set to '../modules/webphone/'
  The webphonebasedir parameter must be set in the webphone_config.js file directly (not at runtime by webphone_api.webphonebasedir).
  Default is empty (assumes that your html is in the webphone folder).

- NS engine download not found: you might have to set the nativepluginurl parameter to point to the ns installer file.

  nativepluginurl
  (string)
  This setting is deprecated after 1.9 as the webphone should automatically detect its library path automatically.
  The absolute location of the Native Service/Plugin installer. In most of the cases this is automatically guessed by the webphone, but if for some reason (for example: you era using URL rewrite) the guessed location is incorrect, then it can be specified with this parameter.
  The Service/Plugin installer is located in webphone package "native" directory.
  Example:
  “https://mydomain.com/webphopne/native/WebPhoneService_Install.exe”
  Default value is empty.

The webphone is not starting

Symptom: The webphone user interface is loaded but nothing is happening (webphone doesn’t start / connect).

Solution: If the webphone doesn't start or doesn't connect, that means that most probably you haven't set the necessary parameters (such as serveraddress/username/password) and/or you haven't called any of the API functions.

The webphone will start when one of the followings happens:

- Autostart: If the important parameters such as the serveraddress/username/password (and sometimes others such as the proxyaddress) are preconfigured (set in the webphone_config.js, via the setparameter API call or passed by URL). In this case the webphone can auto connect if you haven't set the autostart parameter to 0.
- API triggered: even if the autostart parameter is set to 0 and/or the important parameters are not set yet, you can force the webphone start using the start() API. The start API can also auto connect/register (if the above parameters are preset), otherwise you can connect later after you have set these parameters with the register() API.
  Other API's (such as the call() API) might also trigger the start action on demand.

See this FAQ point for more details regarding auto start / auto register.

Note: There are multiple ways to initialize the webphone:

- Example 1:
  - Set all the important parameters in the webphone_config.js and just launch the webphone html (any of the included skins or samples or your own including the webpone_api.js). The webphone should auto start and auto connect.
- Example 2:
  - Set only the serveraddress parameter in the webphone_config.js and just launch a html which also has a login page (such as the included softphone.html). In this case the webphone should just stop at the login page (if there is no any cached credentials yet) waiting for the enduser to enter it's username/password and the start will be triggered from the OK/Login button
- Example 3:
  - Set the autostart parameter to 0.
  - Preconfigure the serveraddress (or other required parameters such as the proxyaddress if needed) set in the webphone_config.js, via the setparameter API call or passed by URL
  - Call the start() function
  - Supply the username/password (or other required parameters such as the sipusername if needed) using the setparameter API.
  - Call the register() function
- Example 4:
  - Set all the required parameters using the setparameter API
  - Use the call() function (this will auto trigger the start action and also the register action if the register parameter is not set to false)
- Example 5:
  - Any other combinations
  - The important thing is to just set the webphone parameters with any method and then call any action API such as start/register/call/etc.

Can't connect to SIP server

If the webphone cannot connect or cannot register to your SIP server, you should verify the followings:

- You have set your SIP server address:port correctly (from the user interface or "serveraddress" parameter in the webphone_config.js file)
- In case if you have set by the webrtcserversaddress parameter, make sure that it actually works. You can test it with this tool (set the "Location" to the same string as you have set for the webphone webrtcserveraddress parameter and check if it can connect).
- Make sure that you are using a SIP username/password valid on your SIP server
- Make sure that the autostart parameter is 1 or 2 and the register parameter is 1 or 2. Otherwise use the start() and register() API explicitly. Details here.
- If you are using it as a WebRTC SIP client library with the Mizu WebRTC SIP gateway service, make sure that your firewall or fail2ban doesn't block the gateways. You should white-list rtc.mizu-voip.com and usrtc.webvoipphone.com
- Make a test from a regular SIP client such as mizu softphone or x-lite from the same device (if these also doesn't work, then there is some fundamental problem on your server not related to our webphone or your device firewall or network connection is too restrictive)
- Use an account-extensions with digest authentication (don't use IP authentication if you are using the webphone WebRTC engine via gateway; in Asterisk the extension should be configured as "users" and not as "peer")
- Check if some firewall, NAT or router blocks the webphone process or the SIP signaling
- Check the browser console output for possible errors (search for "ERROR", but note that some error messages are normal to occure)
- If there are no any answer for the REGISTER requests, turn on your server logs and look for the followings:
  - The REGISTER reaches your server?
  - Is there any response triggered (or some error)?
  - If answer is triggered, is it sent to the correct address? (it should be sent to the exact same address from where the server received the REGISTER request)
  - The answer packet reach the client PC? You can use Wireshark to see the packets at network level.
- Send us a detailed client side log if still doesn't work with loglevel set to 5 (from the browser console or from softphone skin help menu)

Webphone doesn't work with the NS engine

The NS engine is a native executable which is running in background on your OS as a system service, offering SIP/media capabilities for the webphone. The webphone communicates with the NS engine via a websocket connection (or with HTTP poll if websocket is not available in your browser).

The followings might be responsible for the NS engine failure (in case when the webphone has to use the NS engine but can't connect):

1. The NS engine is not running: make sure that the webphone service is started successfully and you can see it in your running process list from Task Manager.
   The "Webphone" or "YourBrandName" service state can be seen by opening the services management console (run services.msc from the command line)
2. You are running a very old outdated NS engine.
   For Windows you can install the latest version from http://yourdomain.com/path_to_webphone/native/WebPhoneService_Install.exe
   (or from the webphone package native folder sent to you by Mizutech)
3. The webphone can't resolve the ns4.webvoipphone.com domain name to 127.0.0.1: Verify by running ping ns4.webvoipphone.com from your command line and it should respond from 127.0.0.1. This might fail for the following reasons:
   a. No any internet connection
   b. DNS requests are blocked
   c. You have a proxy which cannot resolve this domain or blocks it
   Possible workaround: add the following line the hosts file of the client PC: 127.0.0.1 ns4.webvoipphone.com
4. Check your HTTP proxy if any
5. Your browser doesn't accept the SSL certificate of the domain for ns4.webvoipphone.com some reason. As a workaround, you might add it to your browser white list
6. The webphone was unable to listen on the required ports. Make sure that no other applications are using ports: TCP 10443, TCP 18520, UDP 18521, UDP 18522 on your client PC. You can verify this from the Resource Monitor -> Network tab or with the nestat –a command
7. The communication between the browser and the NS service is blocked. This might happen if you have a very restrictive firewall which blocks the communication also on localhost. As a workaround, add the NS engine and the java.exe to the firewall exception list
   To test the connection, open the following URL: https://ns4.webvoipphone.com:10443/extcmd_test (it should respond with some text)
8. The NS engine is unable to start on Windows for some reason. Have a look at the logs (MizuCall_Servicelog.dat and other logs files in the NS engine app directory which is located by default at C:\Program Files (x86)\YourBrandNameWebphone_Service\. Also check the engine log at C:\Program Files (x86)\ourBrandNameWebphone_Service\content\native\)
9. Corrupted JVM: In some circumstances the NS engine might use the Java JVM which might become corrupted for various reasons. Reinstall Java to fix this.
10. You are trying to use the webphone with a SIP server address which is not on your allowed (licensed) server list. In these situations the webphone might also trigger an NS engine upgrade request with the hope that you are running an old version and the current SIP server address were added only later. The NS engine upgrade might not solve the problem if the current SIP server you are trying is not allowed also in your recent webphone/NS engine build.
11. Send us a detailed client side log if still doesn't work with loglevel set to 5 (from the browser console and also the NS and Java logs as described here)

Please note that all of these applies to the client PC only (where the webphone is running in a browser) and has nothing to do with your Web or SIP server.

Failed outgoing calls

Make a test call first from a simple SIP client such as mizu softphone or x-lite

By default only the PCMU,PCMA, G.729 and the speex ultra-wideband codec's are offered on call setup which might not be enabled on your server or peer UA. You can enable all other codec's (PCMA, GSM, speex narrowband, iLBC and G.729) with the use_xxx parameters set to 2 or 3 (where xxx is the name of the codec: use_pcma=2, use_gsm=2, use_speex=2,use_g729=2,use_ilbc=2). Some servers has problems with codec negotiation (requiring re-invite which is not support by some devices). In these situations you might disable all codec's and enable only one codec which is supported by your server (try to use G.729 if possible. Otherwise PCMU or PCMA is should be supported by all servers)

If you receive the "ERROR, Already in call with xxx" error on call attempts and you wish to enable multiple calls to/from the same number, set the disablesamecall parameter to 0.

The webphone by default doesn't allow direct cross-domain calls. For example if you are registered as a@A.com (to A domain) then you will not be able to make direct calls to b@B.com (to B domain). Contact mizutech support to remove this limitation.

If still doesn't work send us a detailed client side log with loglevel set to 5 (from the browser console or from softphone skin help menu)

How to enable only one single codec

Here is an example configuration enabling only PCMU:

```
prefcodec: 'PCMU',
codec: 'PCMU',
alwaysallowlowcodec: 0
```

For NS and Java you can also use the use_xxx settings where xxx is the codec name and the accepted values are 0 for never,1 for don't offer,2 for yes with low priority,3 for yes with high priority.
For example to disable all codec's except PCMU, you might set:
use_pcmu: 3 use_pcma: 0, use_opuswb: 0, use_opusnb: 0, use_g729: 0, use_gsm: 0, use_speex: 0, use_speexwb: 0, use_speexuwb: 0, use_ilbc: 0

Note: you should never set the use_codec settings to 0. Set to 1 if you don't wish to use a codec (so the webphone will still have a chanche to use that coded if the preferred is not available or the other peer sends some other codec then negotiated)

Calls are disconnecting

Have a look at the followings if calls are disconnecting:

- Call disconnection immediately upon setup can have many reasons such as codec incompatibility issues, NAT issues, DTLS/SRTP setup problems or audio problems. If you are not sure, send a detailed log to webphone@mizu-voip.com
- If the calls are disconnecting after a few second, then try to set the "invrecordroute" parameter to "true" and the "setfinalcodec" to 0.
- If the calls are disconnecting at around 100 second, then most probably you are using the demo version which has a 100 second call limit.
- If the calls are disconnecting when you open other webpages (other tabs or in new window), make sure that your CPU usage is not too high (which might be caused by other webpages or by misusing the phone API)
- You are calling the hangup API from some code path (add a log for each of your hangup function calls and inspect your browser console)
- The disconnect might be initiated by the server or from the peer endpoint (see the logs from where the BYE is sent)

Test environment limitations

You can test the webphone also without a web server, by launching from local file system (for example from your desktop) or development environment using a localhost server.
For example just extract the webphone.zip on your desktop and double click on the index.html to begin the usage.

Limitations when run from development tools:
Sometime you might wish to run the webphone from your preferred development environment (for example for debug capabilities). The limitations in this case depends on the sandboxing technology used by your tools. Some tools might not allow websocket connections (this will block both the NS and the WebRTC engines). Try to change the browser engine if you run into these limitations or use an external browser process if possible.
Also make sure to open/run only one webphone instance at a time, especially if you are testing with the NS engine (close old browser windows before opening a new webphone instance).

Some versions of Chrome also might fail if you try to run WebRTC from html launched from local file system.
The workaround for this is to launch with --allow-file-access-from-files parameter
(Something like this on windows: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --allow-file-access-from-files C:\path\softphone.html)

For a better experience, we recommend to deploy the webphone files on a web server and test from there. The best experience is with a secure web server (via https), but you can also test with a local web server via http://localhost.
Unsecure websocket connections are usually not allowed on Linux (use a https server for testing).

Using the webphone on local LAN

In short:
Yes, the webphone works on private networks by default, without the need of any configuration changes.

Note: It is completely normal to use the webphone on LAN's (browser client with private IP). This FAQ refers to the case when the SIP server (set by the "serveraddress" parameter where the webphone will register to) and/or Web server (from where you load your webpage embedding the webphone) is located on private IP.

Details:

- The webphone can be used also on local LAN's (when your VoIP server and/or Web server are on your private network).
- The NS and Java engines will connect directly to your server as a normal SIP softphone does.
- For WebRTC to work you will need a WebRTC to SIP gateway on your LAN or your PBX have to support WebRTC, otherwise this engine will not be picked up (this is handled automatically). You should also host your webpage on https (SSL certificate installed on your Web server) for WebRTC to work in modern browsers (or test it by launching from local file system).
- If WebRTC is not supported in your environment, then the webphone might offer to download a browser plugin, which is usually an one-click installer.
- The webphone could use the Mizutech STUN, TURN, JSONP and HTTPS gateway services by default, however these are not required on local LAN's (the webphone will detect this automatically and will not try to use these services while on local LAN).

Old notes (these are not relevant anymore in latest webphone version since NS engine is implemented also for MAC and Linux):
Recommendation for quick tests on local LAN :

- *If you wish to work with the webphone on local LAN and your VoIP server doesn't have WebRTC support or your webserver doesn't have SSL installed for the domain you are using (HTTPS), we recommend to test on Windows operating system (with the NS engine). On other platforms you might use a Java enabled browsers for easy tests or setup WebRTC in your environment.*

Recommendation for production on local LAN :

- *If the client PC's are running Windows OS then there is nothing to do. The webphone will run optimally using its NS engine*
- *For other operating systems (MAC, Linux) we recommend to use the webphone' WebRTC engine: check your VoIP server WebRTC capabilities and set the webrtcserveraddress webphone parameter accordingly or install a local WebRTC-SIP gateway. You can find more details here.*

Using the webphone without internet connection

The webphone can be used also without internet connection with some limitations. An easy workaround for all this would be to enable at least CA verifications (SSL certificate verifications) to the internet, however if this is not possible then the following applies:

- WebRTC in Chrome needs HTTPS (secure http), which will work only with a local policy, otherwise the browser will not be able to verify the SSL certificate against public CA. If you can't setup a local CA or browser rule for this, just disable WebRTC (or use old Firefox v.67 instead of Chrome if you need WebRTC without certificate).
- Java applets need to be signed and on startup the JVM will have to pass the code verification signature. Workaround: Just disable the Java engine or add the applet location to the exception site list (this is only if you wish to use the Java engine which is otherwise rarely used)
- The NS engine can be used from unsecured http in local LAN's with no issues (on https you might need to add the ns4.webvoipphone.com domain to the browser security exception list. This domain name just points to localhost/127.0.0.1, used by the webphone to communicate with the NS engine)

These circumstances are also automatically handled by the webphone, always selecting the best suitable engine if it has at least one available and unless you change the engine priority related settings.

Using the webphone in controlled environment

If you are using the webphone in a controlled environment (where you have control over the clients, such as call-centers) then you might force the NS or Java engines by disabling or lowering the priority for the WebRTC engine (enginepriority_webrtc = 1). This is because NS and Java are more native for SIP/RTP and might have better quality, more features and lower processing costs on your server. The big advantage of WebRTC is that it can work without any extra plugin download/install, however in a controlled environment you can train your users (such as the callcenter agents) to allow and install the NS engine when requested and this one-time extra action will reward with long term quality improvement.

WebPhone behind HTTP proxy

Usually the webphone can auto-detect the HTTP proxy capabilities and act accordingly. However in some rare circumstances the connections might fail if you are behind a misconfigured or outdated HTTP proxy.

You can find our whether you are behind a proxy or not by checking your browser connectivity settings. The followings are the common issues:

In case if you have installed the NS engine on Windows, make sure that your browser can resolve the ns4.webvoipphone.com domain to 127.0.0.1. You can also test this by opening this link in your browser: https://ns4.webvoipphone.com:10443/extcmd_test. If can't connect, then make sure to configure your browser to bypass the proxy for this domain and/or set this domain in your hosts file to point to 127.0.0.1. This is relevant only for some browsers which doesn't allow direct connections to localhost, such as old Edge and Internet Explorer.

In case if you wish to use the webphone WebRTC engine (or it is automatically selected by the webphone), then make sure that your HTTP proxy has support for WebSocket, otherwise WebRTC will not work.

Load the webphone in iframe

If you wish to load the webphone in an iframe, make sure to allow microphone and camera access to that iFrame, because otherwise it will not work in latest Chrome or in other modern browsers.
Example:

```
<iframe id="webphoneframe" src="softphone.html" width="300" height="500" frameborder="0" allow="microphone *; camera *; autoplay *" allowfullscreen="true" ></iframe>
```

Note: It is important to set allow="microphone; camera", otherwise the WebRTC engine can't work! (you can skip this only if you prefer other engines such as NS and you wish to completely disable WebRTC for some reason).
To avoid cross domain issues, you should host the webphone on the same domain where you host your website/webpage. Otherwise make sure to configure your server and iframe permissions/security attributes accordingly.

You might place your webphone user interface in a DIV tag instead.
You should use relative paths instead of full URI if you load the webphone from the same domain.

You might also have to set the iframe sandbox attribute after your needs.
For example you might set the following sandbox attribute to allow all permissions: "allow-forms allow-modals allow-popups allow-popups-to-escape-sandbox allow-same-origin allow-scripts".
Example:

```
<iframe id="webphoneframe" src="softphone.html" width="300" height="500" frameborder="0" allow="microphone *; camera *; autoplay *"
allowfullscreen="true" sandbox="allow-forms allow-modals allow-popups allow-popups-to-escape-sandbox allow-same-origin allow-scripts" ></iframe>
```

Instead of setting your iframe parameters statically in your html, you can also set it from JS like this:
(this script will also pass the page URI query parameters to the webphone frame)

```
window.onload = function ()
{
	var page = window.location.href;
	if (typeof (page) !== 'undefined' && page !== null && page.indexOf('?') > 0)
	{
		page = page.substring(page.indexOf('?'));
	}else
	{
		page = '';
	}

	var frm = document.getElementById('IFRAME_ELEMENT_ID');
	if (typeof (frm) !== 'undefined' && frm !== null)
	{
		frm.src = 'https://yourdomain.com/path_to_webphone/softphone.html' + page;
		frm.setAttribute("allow", "microphone *; camera *; autoplay *");
		frm.setAttribute("allowfullscreen", "true");
		setTimeout( function () { frm.style.display = 'block'; }, 50); }else { try{console.log('Could not find webphone frame');} catch(e) { ; }
	}
};
```

In case if you wish to access the webphone_api object in an iframe from outside the iframe, then you should either host both (your main page and the webphone/iframe content) on the same domain or set the Access-Control-Allow-Origin header on your web server as described here, otherwise you will get *ERROR DOMException: Permission denied to access property* errors.

Including the webphone to all your pages

In case if you wish to include the webphone globally to your websites to be present on all pages (such as a “call to support” widget flying on the bottom-right side of your page), make sure to don’t let the webphone to auto-initialize itself automatically with each page load/reload because this might slow-down the responsivity of your website.
For this just set the “autostart” parameter to “0”.
In this call you can delay the VoIP engine initialization to the point when the enduser actually wish to interact with your VoIP US (such as clicking on your click to call button).

Multiple phones on the same page

You just have to include the “webphone_api.js” to your page and create multiple VoIP UI elements.
For example you might have a contact list (or people list) displayed on your page, with a “Dial” button near each other. You don’t even need to initialize the webphone on your page load (set the “autostart” parameter to “0”). Just use the webhone_api.call(number) function when a user click on the dial button and the webphone will initialize itself on the first call.

The softphone user interface (softphone.html) can’t be included multiple times in a page. If you really need multiple phone UI or multiple completely separated webphone instance on your page, then use separate iFrame for them.

Example:

```
<div class="content">
<iframe id="webphone1" src="./webphone1/softphone.html?wp_anyparameter=a" width="300" height="500" frameborder="0" allow="microphone; camera; autoplay"
allowfullscreen="true" sandbox="allow-forms allow-modals allow-popups allow-popups-to-escape-sandbox allow-same-origin allow-scripts" ></iframe>
<iframe id="webphone2" src="./webphone2/softphone.html?wp_anyparameter=b" width="300" height="500" frameborder="0" allow="microphone; camera; autoplay"
allowfullscreen="true" sandbox="allow-forms allow-modals allow-popups allow-popups-to-escape-sandbox allow-same-origin allow-scripts" ></iframe>
```

You should also adjust the profile and/or the usepathinfilenames parameters to avoid configuration mismatch between the webphone instances if you are using different settings. For example you might pass an unique string as the profile parameter in the iframe URL (wp_profile=anystring) and set the usepathinfilenames parameter to 2 in the webphone_config.js file.

Load the webphone on demand

Below are a few (both recommended and NOT recommended) methods to load the webphone into your webpage:

a. Load "webphone_api.js" using a script tag in the <head> section of your web page. This is actually not “on demand”, the webphone will be loaded when the page is loaded.

b. Load "webphone_api.js" on demand, by creating a <script> DOM element. Below is an example function which loads the script into the <head> section of the page:

```
function insertScript(pathToScript)
{
        var addScript = document.createElement( "script" );
        addScript.type = "text/javascript";
        addScript.src = pathToScript;
        document.head.appendChild( addScript );
}
```

c. The webphone can also be loaded into an iframe on demand. To have access to the webphone API in the iframe from the parent page, you have to follow the below two steps:

i. include the /js/lib/iframe_helper.js file into your parent html page <head> section

ii. set the iframe's "id" attribute to "webphoneframe", for example: <iframe id="webphoneframe" src="softphone.html" width="300" height="500" frameborder="0" allow="microphone; camera" ></iframe>

iii. access the webphone API from the parent page only after the page has finished loading, for example:

```
window.onload = function ()
{
        webphone_api.onAppStateChange (function (state)
        {
                if (state === 'loaded')
                {
                        webphone_api.setparameter("serveraddress", "SERVERADDRESS");
                        webphone_api.setparameter("username", "USERNAME");
                        webphone_api.setparameter("password", "PASSWORD");

                        webphone_api.start();
                }
        });
};
```

Not recommended:

1. The web phone can be loaded on demand using document.write(), but it is a bad practice to call document.write() after the page has finished loading.
2. The web phone can also be loaded using any AMD (Asynchronous Module Loader). This is not recommended, because webphone also uses AMD (Require JS) to load its modules, so it won't improve performance, but it can lead to conflict between AMD loaders.

How to keep the webphone call between page loads?

The webphone is a client side software and pages/tabs in your browser are separate entities, so a new page doesn’t know anything about an old one (except via server side sessions, but it is impossible to transfer live JavaScript object via your server in this way).

There is no way to keep a WebRTC session alive between page loads.
Instead of this, you should choose one of the followings:

- run the webphone in a separate page (on its own dedicated page, so the enduser can just switch to this window/tab if needs to interact with the webphone)
- run the webphone in a separate frame

- load your content dynamically (Ajax)

If this functionality is a must for your project, check the following FAQ for the possibilities.

However, if you are using the NS engine, then calls can survive page refresh by setting the following parameters:

- callreceiver: 2
- backgroundcalls: 1
- destroyonpageclose: 0
- needunregister: false

Single webphone instance on multiple pages

There might be situations when you might wish to use the same webphone instance on multiple pages (cross-tab: opened in different browser tab or windows). For example to start a call on a page, open a second page and be able to hangup the call on this second page.

First of all, it is important to note that the webphone is client side software (it is impossible to implement a voip client which would run on the server side). This means that from the browser perspective, each of the pages are treated completely separately (Only your web server knows that those belongs together called a "session"). Each page load or reload will completely re-initialize also the webphone (if the webphone is included in the page). With other worlds: multiple pages opened from your domain doesn't know about each other at all and one page can't access the other one (except if you send some ajax message via your web server, but this kind of message passing is useless in this case since you can't transfer the whole webphone javascript object).
This means that you should avoid the above use-case and just launch the webphone on a separate page in this case, so the enduser can switch to the page dedicated for the webphone if need to interact with a call (make a call, hangup current call or other operations such as mute, conference, dtmf).

Here are a few ways to implement such functionality:

- Simple data sharing: If you just want to share some details across your pages, then you can do it via cookies or from pure javascript using the window.name reference. (This can be used only for simple data sharing, but not to share live JavaScript objects such as the webphone)
- NS engine: It is possible with the NS engine to have the webphone survive page (re)loads or opening new pages on your website. This can be a solution of you know in advance that all users will use the NS engine only (increase the value of the enginepriority_ns parameter in this case)
- Using a global webphone object: There is way to share a global webphone instance across the opened pages: using the **window.opener** property which is a reference to the window that opened the current window. This means that you will pass the webphone instance across your pages so can access your global webphone object from secondary opened pages via the opener reference (Find an example for this below)
- Avoid this use-case and just launch the webphone on a separate page in this case, so the enduser can switch to the page dedicated for the webphone if need to interact with a call (make a call, hangup current call or other operations such as mute, conference, dtmf). You might also load the webphone in a iframe.

In case if you wish to keep the calls on page close, then set the destroyonpageclose parameter to 0 and the autostart parameter to 0.

Here is a simplified example to access the webphone object via window.opener:

```
//It is important to set the "autostart" parameter to "0" in the webphone_config.js to avoid auto initialization of the webphone on all pages where included (we will start the webphone explicitly when needed)

//store the wopener variable to be used here and also on subsequent pages (useful if we open a third page from the second and so)
var wopener = window;  //set to this document
if(window.opener && window.opener.webphone_api)
{
        wopener = window.opener; //set to parent page document
}
if(wopener.wopener && wopener.wopener.webphone_api)
{
        wopener = wopener.wopener;  //the parent page might also loaded from its own parent, so load it from there
}

//create a reference to the webphone so we can easily access it on this page via this variable
var wapi = webphone_api;

//Initialize your webphone if not initialized yet
if(wopener && wopener.webphone_api)
{
        //load the wapi instance from the parent page in this case
        wapi = wopener.webphone_api;

        //check if already initialized
```

```
		if(wapi.isstarted != 1)
		{
			wapi.isstarted = 1;
			//we are staring the engine here, however you can delay the start if you wish to the point when the user actually wish to use the phone such as making a call
			wapi.start();
		}
		//else already initialized by parent
}
else if(wapi && wapi.isstarted != 1)
{
		//we are the first page
		wapi.isstarted = 1;
		wopener = window; //set the wopener to point to this page
		wapi.start();
}

//use the phone api on this page
function onCalllButtonClick()
{
		if(wapi) wapi.call();
		else alert('error: no webphone found (webphone_api.js not included?)');
}
```

You can find a better/fully working example in the webphone package: multipage_example.html.

How to prevent unwanted unload event

Certain operations (such as file download controls) might trigger window.unload events which might trigger webphone unregistrations.
You might have to prevent these event being triggered by your controls by using this technique (It can be applied to any element such as ,<a>,<button>)

How to prevent multiple endpoints to one account

Users might login to the SIP server from multiple locations with the same user account and most SIP servers will forward the call to all locations by default (multiple AOR/register binding with call fork to multiple registered endpoints).
In some circumstances you might prefer only one register for an account (for example if the user leaves the webphone running at home, then logins from the office, all calls will be forked both to the office and to home).
To prevent registration from multiple locations (thus forking the calls to multiple endpoints), set the unregonidle parameter to the number of seconds after that the webphone will automatically unregister on no user activity.
You might also set the unregall parameter to 1 to always unregister all accounts and the unregonstart parameter to 1 to unregister at startup from the NS engine (before the first register).
You might also lower the registerinterval (don't set it to higher then the unregonidle parameter).
Example settings:

```
unregonidle: 600, //unregister after 10 minutes of user idle
unregall: 1, //optional if required for your use-case
unregonstart: 1, //optional if required for your use-case
```

You might also configure your server to allow only one endpoint (registrar AOR) per account and/or disable call forking.

New parameters was set but the old settings was loaded

Sometimes you might have to change the settings for each session (for example changing the user credentials).
In these situations it might happen that the webphone is still using the old setting (which you have set for the previous session and not for the current one).
Usually this might happen if the webphone is already started and registered with the old parameters before it loads the new parameters (For example before you call the setparameter() API with the new values).

To prevent this, you should set the "autostart" parameter to "0" in the webphone_config.js
You can also set the "register" parameter to "0".
The use the start() and/or register() functions only after the webphone were supplied with the new parameters.

Note:
The webphone is also capable to load it's parameters from the URL. Just us the format required (wp_username, wp_password and others).

It is not needed to call the register() after start() because the start() will automatically initiate the register if the server/username/password is already preset when it starts and if you leave the register parameter at 1.

Ports

The webphone will need the following ports to listen on (local ports):

- The SIP signaling port (TCP or UDP port configurable with the signalingport setting)
- SIP media ports for RTP/RTCP (UDP ports configurable with the rtpport settings)
- WebRTC: When using it as a WebRTC client library, the above ports are determined by your browser and can't be influenced by the webphone
- NS engine: When used with the NS engine it requires the following TCP and UDP internal ports on localhost only: 18520, 18521, 18522, 10443 (make sure to not use some restrictive firewall which blocks these ports also for localhost/127.0.0.1)

The webphone might connect to the following ports (listeners on your SIP server side):

- SIP signaling port (configurable with the serveraddress setting. Usually UDP 5060)
- SIP media ports for RTP/RTCP (UDP ports as negotiated with your server via SIP signaling SDP)
- WebRTC only (connecting to your server WebRTC module or to WebRTC-SIP gateway):
    - Websocket for WebRTC (configurable with the webrtcserveraddress setting. Usually TCP 80 or 8080)
    - STUN server address (configurable with the stunserveraddress setting. Usually UDP 80, 3478 or 8090)
    - TURN server address (configurable with turnserveraddress setting. Usually TCP 80 UDP 3478 or 5349)
- Optional services (described here with the IP/domain listed at the end of the chapter)

NAT settings

In the SIP protocol the client endpoints have to send their (correct) address in the SIP signaling, however in many situations the client is not able to detect it's correct public IP (or even the correct private local IP). This is a common problem in the SIP protocol which occurs with clients behind NAT devices (behind routers). The clients have to set its IP address in the following SIP headers: contact, via, SDP connect (used for RTP media). A well written VoIP server should be able to easily handle this situation, but a lot of widely used VoIP server fails in correct NAT detection. RTP routing or offload should be also determined based in this factor (servers should be always route the media between 2 nat-ed endpoint and when at least one endpoint is on public IP than the server should offload the media routing). This is just a short description. The actual implementation might be more complicated.

With the WebRTC engine make sure that the STUN and TURN settings are set correctly (by default it will use mizu services which will work fine if your server is on the public internet).

For NS and Java engines you may have to change the webphone configuration according to your SIP server if you have any problems with devices behind NAT (router, firewall).
If your server has NAT support then set the use_fast_stun and use_rport parameters to 0 and you should not have any problem with the signaling and media for webphone behind NAT. If your server doesn't have NAT support then you should set these settings to 2. In this case the webphone will always try to discover its external network address.

Example configurations:
If your server can work only with public IP sent in the signaling:
-use_rport: 2 or 3
-use_fast_stun: 1 or 2 or 3 (recommended is 2)

If your server can work fine with private IP's in signaling (but not when a wrong public IP is sent in signaling):
-use_rport: 9
-use_fast_stun: 0
-optionally you can also set the "udpconnect" parameter to 1

Asterisk is well known about its bad default NAT handling. Instead of detecting the client capabilities automatically it relies on pre-configurations. You should set the "nat" option to "yes" for all peers.
More details can be found here.

For FreeSWITCH we recommend to set the NDLB-force-rport and aggressive-nat-detection values to true in the sip_profiles configuration. More details here.

Server failover/fallback

Use the following settings if you have 2 SIP servers:

- serveraddressfirst: the IP or domain name of the first server to try

- serveraddress: the IP or domain name of the next server
- autotransportdetect: true
- enablefallback: true

In this way the webphone will always send a register to the first server first and on no answer will use the second server (the “first” server is the “serveraddressfirst” at the beginning, but it can change to “serveraddress” on subsequent failures to speed up the initialization time)
New versions of the webphone also implements a “backupserver” parameter which can be used instead of the above settings.

Alternatively you can also use multiple DNS A records or multiple SRV DNS records to implement failover or load balancing, or use a server side load balancer.

WebRTC specific:
Browsers doesn’t lookup for SRV DNS records, thus in case if you wish to use multiple servers based on DNS lookups, you should set multiple A records for your domain.
Otherwise the webphone can better handle multiple WebRTC servers by itself, without the need for multiple DNS records.
In case if you have multiple WebRTC servers or gateways, then you can list all of them for the “webrtcserveraddress” parameter. The webphone will automatically choose and available server and it can failover to other if can’t connect via websocket.

I have WebRTC related issues

The WebRTC functionality highly depends on your OS/browser and server side WebRTC –SIP module. Check the followings if the webphone is using the WebRTC engine and you have difficulties:

- Make sure that your browser has support for WebRTC and it works. Visit the following test pages: test1, test2
- Make sure to run the webphone from secure http (https) otherwise WebRTC will not work in Chrome and Opera
- If you have set the “webrtcserveraddress” parameter to point to your server or gateway, make sure that your server/gateway has WebRTC enabled and test it also from some other client such as sipml5: config; test
- You might contact mizutech support with a detailed log about the problem
- If you are unable to fix webrtc in your setup then you might disable the webrtc engine by setting the enginepriority_webrtc parameter to 0 or 1. See the other possibilities here.

Media access or Media stream permission denied

In case the Webphone is using WebRTC engine, then the user needs to grant access for the webphone to recording audio and/or video devices. This is required by the browsers and not by the webphone itself. This permission mechanism is implemented in all browsers and its behavior cannot be altered or changed.

Your servers should be configured with a domain name and a valid TLS certificate. The websocket connection should be secure (WSS) and the webphone should be hosted on HTTPS, otherwise browser might not allow media permissions (microphone recording). Some browsers treats localhost/127.0.0.1 or file also as trusted.

This Permission Request will be presented to the user only when he or she initiates or receives a call. The user might receive similar popups or the calls just fails if you are using the WebRTC engine but haven’t enabled the browser to use your microphone/camera device or denied it previously (Technically the WebRTC getUserMedia() function call will fail in this case).
Normally before WebRTC calls your browser should popup a box asking to allow microphone access. You should click on the Ok/Yes/Allow/Share/Always Share button there. In some situations there is no browser popup, only silent fail with “ERROR, Warning, cannot access microphone” in the log.

The Permission Requests are bit different in every browser, but usually they are presented as a popup box somewhere on the top of the page on Desktop browsers or on the bottom of the page on mobile device browsers.
These popups ask a question similar to the ones below and have an "Allow" and "Block/Don't allow/Deny" button:

- "Will you allow www.domain.com to use your microphone and/or camera?"
- "www.domain.com wants to use your microphone and/or camera"

Important note on HTTP/HTTPS:

- If the Webphone is hosted on a secure web site (HTTPS) then the Permission Request will be displayed only once and the user's choice will be remembered.
- If the Webphone is not hosted on a secure web site (HTTP), then this Permission Request will be displayed for every single call.
- Some browsers such as chrome don’t allow media permission at all on unsecure HTTP. In this case you might try to allow a website from your browser security/privacy settings (In Chrome: settings -> show advanced settings -> privacy section -> content settings -> microphone -> manage exceptions).

- More details here

If access is grant to recording devices, then you will see some kind of notification in every browser:
- in Firefox on desktop a little icon like window will appear on the top of the browser
- in Chrome on desktop a red dot will appear on the browser tab near the web page title
- on mobile devices usually a small notification icon will be displayed on the device's top bar

Another important case is when the user intentionally or accidentally blocks access to the recording devices.
If you clicked on No/Not/Don't Share/Deny/Always Deny button sometime before then the browser might not popup with this question again.
In this case the Permission Request will never be presented to the user again. To present this Permission Request again so you can grant access to the recording devices, you will have to enable it manually from browser settings. In this case you should see a red icon in your browser address bar and click on "Allow" from there. This is a little bit different for all browsers, below are a few examples for most common ones:

- in Firefox on desktop: click Menu icon in top-right corner -> Options -> select Privacy & Security on the left -> scroll down to Permissions section and there you can allow/block access to recording devices for every single web site
- in Chrome on desktop: click Menu icon in top-right corner -> Settings -> scroll down to bottom and click Advanced -> click Content settings and there you can allow/block access to recording devices for every single web site

- in other Chrome versions this can be found at Settings -> Advanced -> Privacy and security -> Site settings -> Camera or Microphone as described here
- in Chrome on Android: tap Menu icon in top-right corner -> Settings -> Site settings and there you can allow/block access to recording devices for every single web site
- in Android native browser you don't have any options to control this, instead Permission Request will be presented every time you initiate/receive a call

If access to recording devices is blocked and the Webphone initiates/receives a call, it will display a toast message: "Cannot access microphone".
If you are using HTTPS/WSS then by default access to recording devices is not blocked in browsers, all browsers ask the user for permission when a webpage (in this case the Webphone) intends to use it, but there is the possibility that the user disabled the use of recording devices for all websites in his browser. In this case he or she has to enable it manually as described above.

Other notes:
- If you disabled the permission earlier, then there will not be any popup just a red icon in your browser and you might see a "Waiting for permission" status in the webphone. Click on it and enable the permission. If you don't see a red icon that means that WebRTC or all special permissions are disable browser wide (if you mangled your browser settings). In this case make sure to re-enable it in your browser setting.
- In case if you are using the webphone in an iframe, make sure to enable permissions for the iframe as described here
- In some situations the browser might not ask for permission, just silently fails. This happens in Chrome if your website is not on secure https (Chrome doesn't allow WebRTC from http) or if you are using an invalid or self-signed certificate (yes, Chrome might just silently fail with self-signed certificates). Also you must use wss (secure websocket) for your WebRTC server WebSocket connection, otherwise Chrome will fail on unsecure ws.
- Some versions of Chrome also might fail if you try to run WebRTC from html launched from local file system
  The workaround for this is to launch with --allow-file-access-from-files parameter
  (Something like this on windows: "C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --allow-file-access-from-files C:\path\softphone.html)
- Also test your browser webrtc capabilities from here and here
- See this FAQ if you are using the webphone in a WebView or if you plan to integrate it into an app

VoIP calls without microphone device

The webphone is capable to handle the situation when calls are being connected without a microphone device. This is useful only if the user needs to listen for some audio such as an IVR.
The only exception is if you use its WebRTC engine with Firefox since Firefox requires the MediaStream to have a valid MediaStreamTrack, but this is returned from getUserMedia() which fails on Firefox if the user don't have a microphone with the following error:
WRTC, ERROR, InternalError: Cannot create an offer with no local tracks, no offerToReceiveAudio/Video, and no DataChannel.
This is a bug in Firefox already reported also by others as you can see here and here.
This situation is handled automatically by the webphone or you can force calls to always pass or always fail via the useaudiorecord setting.

You might set the useaudiodevicerecord parameter to false and the useaudiorecord parameter to 0 if you don't have an audio recorder device installed and don't wish the webphone to check the device. You might also change the volumein/volumeout and mute/defmute settings. If your server disconnects the calls on no RTP activity then you can set the sendrtponmuted parameter to true.
To disable microphone related warnings, set the useaudiorecord parameter to 0 or -1 and the enablenomicvoicewarning parameter to 0.

Video calls

Video calls are supported with the WebRTC engine.
You might need to set the video parameter to 1 first to enable the video features.
Use the videocall function to initiate a video call.

If you are using the webphone as a SIP library with a custom skin (not via the "softphone.html"), then the video will be displayed in a div with id set to "video_container", so your html must have this element: . You might also the setvideodisplaysize to adjust the display size.

You can use the getdevicelist / getdevice / setdevice functions to manage the video devices.

You can also restrict the possible video codec's with the vcodec parameter and/or set a preferred codec with the prefvcodec parameter.
The webphone has support for the H.263, H.264, H.265, VP8 and VP9 video codec's, but codec availability is up to the browsers WebRTC stack. H.264 and VP8 is supported by most browsers.
Other related parameters are the followings: displayvideodevice, video_bandwidth, video size parameters, videofacing.
Change the above parameters only if really required for your use-case and after you already made a test with the default parameters. The defaults are optimal for most use-cases.

The video call feature can be disabled by setting the video parameter to 0 and the disableoptions parameter to 'VIDEO'.

Note:
The WebRTC engine is required for video to work (the webphone will try to handle this automatically by switching to WebRTC on video request if it is available).
Video calls might failback to a simple voice call if you try to use it with an unsupported engine or if you don't have a camera device or if video is not supported by the peer or by the server/gateway. It should always work between WebRTC endpoints if users has a camera device.
The video call features are subject to support restrictions as described here and here. If your main use case is video calls, then you should test with the demo version before to get your license for production.

Video not supported or not accepted

If you make a video call, the followings will happen if the video is not supported or not accepted by the other end:

- Webrtc to Webrtc: if the other party doesn't have video, the call will connect just fine but there will be no video
- Webrtc to SIP: the call will connect just fine but there will be no video
- Webrtc to Webrtc: the other party doesn't grant access to the video device: if you have camera device, and audio and camera permission is requested, you can either grant permission for both, or deny for both. You cannot share just audio or video device.

Note:

- Video calls are not supported with the NS and Java engine, however in this case the webphone can automatically switch to WebRTC (on enduser video call request).
- For the video to work, your SIP server must have full support for H.264 and VP8 or it bypass the media routing and forwards the SDP attributes correctly. The only exception is for webphone to webphone calls when the video might be routed directly between the clients, bypassing your server (if your server doesn't force RTP relay and otherwise is passing the SDP correctly)
- Not all H.264 profiles are accepted by the browsers (you should try with the baseline profile if fails)
- If there is any issue with the video calls from the webphone, you might test the video call capability of your browser here and here.

The webphone can also fix incorrect video parameters or remove video on failure.
You might adjust this with the fixvideo parameter. Possible values:

- -1: auto (default; usually defaults to 1 or 2)
- 0: never
- 1: fix on fail
- 2: fix or remove on fail
- 3: remove on fail
- 4: always fix
- 5: always fix or remove
- 6: always remove
- 7: set to 0
- 8: always set to 0
- 9: redial on outgoing and fix or remove incoming

What are the "best" settings?

This is a question often asked by our customers about how to optimize the webphone library for best call quality. The answer is rather simple for this question: The best settings are the default settings. The default settings are optimized and should be preferred in almost all use cases except if you have some uncommon needs. You should change the default settings only if you have a good reason to do so. See the below FAQ point for the possible changes for production usage.
Related: best codec.

Optimize for production

You might change the following parameters when you switch to live:

- Set the loglevel parameter to 1 (off). The loglevel is usually set to 5 (on) by default which is very helpful for debugging but will consume more CPU/RAM. You can always switch back to 5 if you run into any issue later where detailed logs can be useful.
- Set the enablepresence parameter to 0 if your server doesn't have SUBSCRIBE/NOTIFY support for presence or if you are not interested in this feature (otherwise presence can generate a lot of extra signaling requests using more CPU/RAM on both the client and server side)
- Configure also STUN (stunserveraddress) and TURN (turnserveraddress) if you are using WebRTC. WebRTC calls might work also without these settings, but, depending on the circumstances and your environment, you can increase the success rate for the calls using a STUN and TURN service (better NAT traversal).

Other then these, all parameters comes with optimal defaults and you should change any settings only if you have a strong reason.

How to achieve the best call quality

Since the webphone is configured optimally by default and will auto-negotiate the best possible settings by default, there is no much to tweak on the client side. However a lot will depend on your server side configuration. Here are the key points:

- If possible, avoid RTP routing on the server side altogether. This will allow peer to peer direct media. If your SIP trunk provider requires the RTP packets to be sent from your IP, then configure your server with RTP routing only for outbound/inbound calls (and not for user to user calls)
- Use (enable) OPUS (and speex) wideband if possible. Wideband codec has much better quality then narrowband and even if your service providers(s) accepts only codec like G.729 or G.711, this can be auto-negotiated and you should not force only these codec's on your server (allowing wideband for user to user calls)
- If possible, avoid codec conversion (transcoding). End to end G.711 has better quality then half-way OPUS and then conversion to G.711
- Follow best practices for VoIP (make sure that you have enough and good quality bandwidth with minimal jitter and no packet loss, make sure that your server resources are not over utilized and the CPU is cable to handle RTP routing with minimal delay, etc)

I have call quality issues

When there is any call quality problem with the webphone it is almost 100% that it is caused by some external factor (such as poor network quality) or otherwise it can be fixed by adjusting some of the webphone parameters.

Call quality is influenced primarily by the followings:

- Network conditions (QoS)
- The engine used (NS, Java and WebRTC tends to have the best quality, Flash has the worst quality)
- Codec used to carry the media (wideband has better quality)
- AEC and denoise availability (mediaench)
- Hardware: enough CPU power and quality headset/microphone/speaker (try a headset, try on another device)
- OS/Browser (drivers problems)
- Settings (codec, jitter size, AEC, etc)
- Software problem or bug (webphone/softswitch)
- VoIP route (carrier/call termination/voip service provider call quality)

If you have call quality issues then the followings should be verified:

- Make a test call with some other simple third-party SIP softphone such as X-Lite. If call quality is bad also with this, then the problem is not related to the webphone but it will be some network/hardware/server/trunk issue.
- Check your network quality: upload speed, packet loss, delay and jitter here. (Other tools: a , b, c)
- TURN usage: make sure that the webphone is not behind a restrictive NAT or firewall to make sure that the call can proceed normally on UDP and doesn't require TCP transport over TURN which might result in lower quality then UDP especially on lower quality links
- Check whether you have good call quality using a third party softphone from the same location (try X-Lite for example). If not, than the problem is not related to the webphone (continue to look for server, termination gateway or bandwidth issues)
- Make sure that the CPU load is not near 100% when you are doing the tests (on multi-core systems none of the cores should be near 100% utilization)

- Make sure that you have enough bandwidth/QoS for the codec that you are using (G.729 requires around 24 kbits. G.711 requires around 80 kbits. Opus requires around 20-50 kbits. However the overall bandwidth is rarely the problem and you should check if you have these minimums constantly, with less packet loss then 2% and no big variations in roundtrip delays)
- Change the codec (disable/enabled codec's with the codec and/or prefcodec parameter)
- Try to set only one single codec (such as PCMU) to make sure that the problem is not caused by poor server side handling of the automatically selected codec
- If the audio quality is wrong at the peer side only then make sure that you are using a good quality microphone (try also from another device) and make sure that the RTP packets are actually received correctly to the other side (do a network capture on the other side to rule out network issues for upload –internet connections are usually better for downloads then for uploads)
- Deploy the mediaench module (for AEC and denoise). Make sure that the native dll/so/other files can be actually downloaded from your Web server (enable their mime types if you can't download these files by using their full URI in your browser)
- Try to disable the audio enhancements. Set the following parameters to 0: aec, aec2, agc, denoise
- Set the usecommdevice parameter to 0
- Set the volume in the OS settings to 50% (or to other "normal" level, not near 100%)
- If you are on Linux, try to change your audio driver (from oss to alsa or others)
- If you are using Windows, check if you are not affected by high DPC latency
- If you are using the webphone as a JavaScript WebRTC SIP library, then it might happen that the webphone will chose to use our WebRTC-SIP gateway to handle your traffic. Our gateways are located in US and Germany and if you are fare from these locations (Asia, Australia, South America) then you might have too much delay. There are many ways to solve this: webrtc solutions for web phone
- Check the webphone logs (Check audio and RTP related log entries. Also check the statistics after call disconnect by searching for "EVENT, call details")
- Perform a network trace (With Wireshark for example; Filter for SIP and RTP. Check missing or duplicated packets.)

The remote party hear itself back (echo)

You can fine-tune AEC (acoustic echo cancellation) with the aec, aec2, agc and denoise parameters.

Echo is generated by some devices/circumstances when the output audio signal (from the speaker) arrives back to the input signal (microphone). This usually happens only if you use a loudspeaker (and not with headsets).
Please note that acoustic echo cancellation is not an exact science and its success rate depends on many factors such as device, OS version, browser, audio chip, audio driver, VoIP engine, environment, network delay, headset/speaker/microphone quality, codec and others.
The webphone is capable to remove more than 90% of the echo for more than 90% of the calls. There are some circumstances (depending on the factors listed above) when echo cannot be detected correctly and it is left in the voice signal.
When using the WebRTC, the webphone cannot influence the media processing and thus can't do any additional AEC (but it can force the browser to apply AEC and AEC in this case is done by the browser internal WebRTC media stack).

You hear yourself back (echo)

In this case the echo is generated by the other end and there is little to do with it on the webphone side.
Make sure that the other party uses an AEC capable SIP client or device.

No audio or one-way audio

1. Review your server NAT related settings
2. Make sur that your audio device is connected and working correctly. If you have multiple devices, make sure that the correct one is selected
3. Set the "setfinalcodec" parameter to 0 (especially if you are using Asterisk or OpenSIPS)
4. If you are running the webphone on the same local LAN with your SIP server, try to set the "stunserveraddress" and "turnserveraddress" to "NULL". (Sometimes this can be used also over the public internet to avoid STUN and TURN related issues)
5. Check stun and turn settings (might be used for WebRTC if your server is not on the public internet, doesn't route the RTP or you need peer to peer media routing)
6. Set use_fast_stun, use_fast_ice and use_rport to 0 (especially if you are using SIP aware routers). If these don't help, set them to 2.
7. Set the enableearlyreinvite parameter to 0.
8. Check this FAQ if you are using Asterisk
9. If you are using Mizu VoIP server, set the RTP routing to "always" for the user(s)
10. Make sure that you have enabled all codec's
11. Make a test call with only one codec enabled (this will solve codec negotiation issues if any)
12. Check the next section if there is any audio open related issues in the log (Audio device cannot be opened)
13. Change the VoIP engine (switch to NS if you have used WebRTC before or inverse. This can be enforced with the enginepriority… settings)

14. If you still have one side audio, please make a test with any other softphone from the same PC. If that works, then contact our support with a detailed log (set the" loglevel" parameter to 5 for this)

Audio device cannot be opened

First of all, make sure that you have a working audio device (and driver).

If you can't hear audio, and you can see audio related errors in the logs (with the loglevel parameter set to 5), then make sure that your system has a suitable audio device capable for full duplex playback and recording with the following format:
PCM SIGNED 8000.0 Hz (8 kHz) 16 bit mono (2 bytes/frame) in little-endian

If you have multiple sound drivers then make sure that the system default is workable or set the device explicitly from the webphone (with the "Audio" button from the default user interface or using the "devicepopup()" function call from java-script)

To make sure that it is a local PC related issue, please try the webphone also from some other PC.
If you are using Windows 10, make sure that microphone access is not blocked.

You might also try to disable the wideband codec's (set the codec parameter explicitly to a narrowband codec or set the following parameters to 0 or 1: use_opuswb, use_opusuwb, use_opusswb, use_speexwb, use_speexuwb).

Another source for this problem can be if your sound device doesn't support full duplex audio (some wrong Linux drivers has this problem). In this case you might try to disable the ringtone (set the "playring" parameter to 0 and check if this will solve the problem).

Some linux audio drivers allow only one audio stream to be opened which might cause audio issues in some circumstances. Workaround: change audio driver from oss to alsa or inverse. Change the JVM (Open JRE vs Oracle JRE) if you are using the Java engine; Change browser if you are using the JS API.

If these don't help, you might set the "cancloseaudioline" parameter to 3 and/or the "singleaudiostream" to 5.

Microphone is not closed

In some circumstances the browser might still display the recording icon after call hangup, even if the webphone actually doesn't record anything anymore.
This can happen only for WebRTC call if the webphone caches the microphone handler for fast reuse, usually only after multi-line or multiple/simultaneous calls.
You might set the removemultilinestream parameter to true to fix this issue, as mentioned also at the multi-line FAQ point.

How to change the sounds?

You can change the sounds by just replacing the audio files.
Make sure to change the files in both the native and the sound folders.
For example to change the ring tone, replace the followings:

- \sound\rtc_ringtone.wav
- \sound\rtc_ringbacktone.wav
- \native\ring.wav

Also set the ringtone parameter to "ring.wav" (otherwise the NS and Java engines might use their built-in ringtone instead of the ring.wav file)

If you wish to use other files for the ringtone, you can set it with the following parameters (otherwise will default to the above paths):

- ringtone: for the NS/java engine (it can be a local path)
- wringtone: incoming call ring for the WebRTC engine (URL path)
- wringtone_out: outgoing call ringback for the WebRTC engine (URL path)

The ringtone can be disabled with the playring parameter: 0: don't play any ring tone, 1: only for incoming calls, 2: play for both in/out. Default is 2.

The sound files should be standard 8 kHz 16 bit mono PCM files (128 kbits - 15 kb/sec).
To disable ringtones, sent the playring parameter to 0 (disable all) or 1 (disable ringback). Default is 2 which means enable all.
Other ringtone related settings: volumering, ringtimeout, beeponincoming.

No ringback tone

Depending on your server configuration, you might not have ringback tone or early media on call connect.
There are a few parameters that can be used in this situation:

- set the “changesptoring” parameter to 3
- set the “natopenpackets” parameter to 10
- set the “earlymedia” parameter to 3
- change the “use_fast_stun” parameter (try with 0 or 2 or 4)
- set the “nostopring” parameter to 1
- set the “ringincall” parameter to 2

One of these should solve the problem.

“Local WebRTC server required” error

This might be displayed in rare circumstances if your try to use the webphone with a local SIP server (SIP server with private IP) and no any other engines are available in your environment. The problem can be fixed with local WebRTC capabilities (built in your server or by a WebRTC-SIP gateway.

In these circumstances (for example running on Android with local SIP server), the webphone might not find any suitable engine because of the following reasons:
-the NS engine is not available on Android and iOS
-Java applets are not supported by your browser
-the webphone also can’t use our WebRTC gateway, since your SIP server is on local LAN (it could use our gateway if your SIP server would be on a public IP)
-also you haven’t configured any local WebRTC server (with the “webrtcserveraddress” parameter)

If any of the above conditions would be false, then the webphone would run with no issues.

If this is only a test environment and later you plan to use a SIP server with public IP, then the problem will be resolved automatically and for now I recommend to use some other environment for test.

If your final production environment will be similar, then the followings can solve the problem:
-configure the webphone “webrtcserveraddress” parameter to point to your server websocket listener (if your server has WebRTC capabilities)
-alternatively you can use a separate WebRTC-SIP gateway (any third party solution is fine such as doubango or janus or our MRTC)

You can find more details about this in the “How to handle WebRTC” and “How to configure WebRTC with my WebRTC server” FAQ points.

Chat is not working

Make sure that your softswitch has support for IM and it is enabled. The webphone is using the MESSAGE protocol for this from the SIP SIMPLE protocol suite as described in RFC 3428.
Most old Asterisk installations might not have support for this by default. You might use Kamailio for this purpose or any other softswitch (most of them has support for RFC 3428).
For new versions of Asterisk based SIP servers you need to properly configure SIP MESSAGE routing in order for IM to work.

If subsequent chat messages are not sent reliably, set the “separatechatdiag” parameter to 1.
In case if you are using the softphone user interface (softphone.html) then you can disable chat by setting the textmessaging parameter to 0. (Alternatively you might use the disableoptions parameter).

Presence is not working

Presence is implemented as standard SIP PUBLISH/SUBSCRIBE/NOTIFY so your SIP server must support RFC 3265 and RFC 3856 in order for this to work.
For Asterisk based servers you can find configuration details here and here.

The webphone doesn’t receive incoming calls

To be able to receive calls, the webphone must be registered to your server by clicking on the “Connect” button on the user interface (or in case if you don’t display the webphone GUI than you can use the “register” parameter with supplied username and password, or via the register() JavaScript SIP API)

Once the webphone is registered, the server should be able to send incoming calls to it.

To find out the reason of failed incoming calls, check the followings:

1. Check if the call arrives to your SIP server (if not, then the problem has nothing to do with the webphone)
2. Check if your server is sending the INVITE to the proper IP:port (from where it received the latest valid REGISTER from the webphone)
   If your server doesn’t route the call or routes to wrong address, check the followings:
   a. Increase your server log level, so you can see the SIP signaling messages in the logs
   b. Check any error in your server log near the call setup or near the previous registration coming from the webphone
   c. NAT: if your browser webphone is behind NAT, check if your server can handle NAT’s properly (via rport and other settings). As a workaround you might try to start the webphone with use_fast_stun parameter set to 0 and if still not works then try it with 2 or 4.

d. Call fork: if you are registered from multiple locations with the same credentials then your server must be able to support call fork to ring on all devices. Otherwise make sure to use the same credentials only from one location and one protocol (don't mismatch SIP and WebRTC logins)

3. Check if the INVITE arrives to the webphone and the webphone process it normally
   a. Search for "INVITE SIP" in the webphone log (if not found, then check if the webphone is actually connected)
   b. Check any error in the webphone log near the incoming INVITE
   c. Make sure that autoignore or DND (do not disturb) are not set

If the calls are still not coming, send a detailed log from the webphone (set the loglevel parameter to 5) and also from the caller (your server or remote SIP client)

What is the best codec?

This depends on the circumstances and there is no such thing as the "best codec". All commonly used codec's present in the webphone are well tested and suitable for IP calls with optimized priority order by default, regarding to environment (client device, bandwidth, server capabilities).
This means that usually you don't need to change any codec related settings except if you have some special requirement.
Between webphone users (or other IP to IP calls) you should prefer wideband codec's (this is why you just always leave the **opus** and **speex** wideband and ultra wideband with the highest priority if you have calls between your VoIP users. These will be picked for IP to IP calls and simply omitted for IP to PSTN calls).
Otherwise **G.729** provides both good quality and low bandwidth if this codec is available for you.
G.711 (PCMU/PCMA) is always supported and they offer good call quality using some more bandwidth then G.729.
The other available codec's are **iLBC** and **GSM**. These offers a good compromise between quality and bandwidth usage if the above mentioned opus and G.729 is not supported by your server or the other peer.
The WebRTC engine will use the browser media stack capabilities. Most browsers has support for at least G.711 (PCMU/PCMA) and OPUS, so you always have a narrowband codec which should work with all trunks (G.711) and a wideband codec for the best quality (OPUS). Other codec's might also work with automatic transcoding. By default the WebRTC engine will negotiate the best possible codec.
To calculate the bandwidth needed, you can use this tool. You might also check this blog entry: Codec misunderstandings
With the webphone you don't need to change the codec settings except if you have some special requirement. With the default settings the webphone is already optimized and will always choose and negotiate the "best" available codec.

Not all the listed codec's are supported by all engines. WebRTC codec support is up to the browser.
The webphone will always automatically choose the best available codec based on your settings, local capabilities, peer capabilities and other circumstances (networking, CPU type, etc).
Audio codec support per engine:

- PCMU (G.711/μ-law): WebRTC, NS, Java, Flash
- PCMA (G.711/A-law): WebRTC, NS, Java, Flash
- G.729: NS, Java, Flash
- GSM: NS, Java
- iLBC: NS, Java
- iSAC: WebRTC
- SPEEX: NS, Java (both narrowband and wideband)
- OPUS: WebRTC, NS, Java (both narrowband and wideband)

Call quality statistics

NS or Java engine:
For RTP statistics increase the loglevel to at least 3 and then after each call longer than 7 seconds you should see the following line in the log:

EVENT, rtp stat: sent X rec X loss X X%.

If you set the loglevel parameter to at least 5 than the important rtp and media related events are also stored in the logs.

You can also access the details about the last call from the softphone skin menu "Last call statistics" item.

More detailed RTP statistics can be obtained by setting the rtpstat parameter to -1 or to a positive value to receive RTPSTAT notifications by the onEvent callback. More details here.

WebRTC engine:

In case if you are using the WebRTC engine then you can capture the statistics via the RTCPeerConnection object which is exposed by the webphone_api.getrtcpeerconnection() function.
From there you can use the RTCRtpStreamStats object to get the statistics.
If you don' wish to use the RTCRtpStreamStats directly, then there are also libraries for this such as webrtcmetrics.

RTP statistics

For RTP statistics increase the log level to at least 3 and then after each call longer than 7 seconds you should see the following line in the log:

EVENT, rtp stat: sent X rec X loss X X%.

If you set the "loglevel" parameter to at least "5" than the important rtp and media related events are also stored in the logs.

You can also access the details about the last call from the softphone skin menu "Last call statistics" item.

This will work only with the NS and Java engines and not with WebRTC (the WebRTC media is handled internally by the browser with no access for RTP statistics).

Callcenter integration

[Optimizations for VoIP callcenter]
The webphone is a favorite VoIP client for callcenters as it can be easily integrated with any frontend, backend or CRM and it can be used for both inbound and outbound campaigns. The integration usually consists of database lookup for caller/callee details on incoming/outgoing calls so the agent can see all the details about the customer.
There are multiple ways to implement such kind of database/CRM/API lookups:

- From JavaScript catch the call init from the onCallStateChange callback (on event = 'setup') and load the required data by an AJAX call to your backend
- Via the "scurl_displaypeerdetails": implement a HTTP API on your server which will return the peer details and set the scurl_displaypeerdetails webphone setting to point to this API URL
- If your backend has VoIP client integration capabilities, then just implement its specification. For example here is a tutorial about integrating the webphone with salesfoce

There are many other things what you can do for a better integration, such as processing cdr records or recording the calls however most of these can be easily controlled by the webphone parameters or implemented via the API. You can also easily implement any kind of call disposition after your needs integrated with your user interface and/or pushed to your server with a simple AJAX call.

We recommend use the NS and/or the Java VoIP engine in call-centers since these provides native call processing, connecting directly to your SIP server without the need of any extra layer such as WebRTC. More details here.

How to store the call details in my database?

The webphone is a client side mostly javascript solution running in the client browsers so it can't touch directly any server side database.

However, you can easily send the call details from the webphone to the server (web service) and from there insert the required fields to your database with a simple script.
(Please note that you will need to write a small script for this on the server side to handle the HTTP AJAX request from the webphone or just extend your existing web service if any with one more API for this. You can do this in your preferred language such as .PHP, .NET, nodejs or anything what you are using on the server side)

There is multiple ways to catch the call details and send it to your server (use any ONE of these):

- Set an onCdr callback function which will be triggered after each call and make an AJAX request from there to your server with the call details.
- You could also use the getlastcalldetails function (call this function after a call to get the CDR -call details).
- Set the scurl_onoutcalldisconnected and scurl_onincalldisconnected webphone parameters to your server side script API URL and the webphone will automatically submit the call details after each call (you might use this if you don't have any JavaScript knowledge to implement a custom ajax submit mentioned in the above points).
- All SIP servers has CDR reporting/storage features and you might use that instead of the webphone to get the call details.
  In this case there is nothing to do with the webphone, you can query the call details from your SIP server via some HTTP API or from its database using SQL.

How the conference works?

The webphone has built-in support for multiple conference solutions and it's behavior can be set changed with the conferencetype parameter:

- Conference via built-in conference mixer (NS and Java client side RTP mixer, so the conference will work, even if your server doesn't have conference support)
- Conference via standard SIP 3-way calling (NS and Java engines and basic support also for WebRTC)
- There is no built-in conference support in the WebRTC protocol, but the webphone is capable to create conference calls also while using WebRTC with the following methods:
  - auto switch to NS or Java (When any of these are available, then conferencing will work with any peer regardless of your server capabilities)
  - using server or gateway side DTMF controlled conference functionality.
  - using server or gateway side conference room (When this is supported by the server or gateway, then it is transparently integrated. Note: conferencing might be created only between Webphone instances in this case and might not succeed for SIP or PSTN peers. Hangup per line is not supported in this mode)

    - Note: Mizutech doesn't provide direct support for WebRTC conference rooms (especially if your SIP server doesn't support SIP MESSAGE and SIP REFER properly or if there is no available server-side WebRTC conference room support). See the "known limitations" for more details.
- Server side conferencing is also available in all circumstances (via DTMF/IVR/conference rooms as supported by your server)

When conferencing is available, you can trigger a conference call in the following ways:

- If you are using the softphone user interface: use the conference menu option in the main menu or the add people option while in call
- If you are using the webphone as a JS SIP library: use the conference API function

P2P

The "P2P" term is misleading sometimes and it can have the following meanings:

- Peer to peer calls: refers to the ability to call other SIP endpoint directly without the need for a SIP server. This is fully supported by the webphone. You just need to call to the other endpoint full URI (such as sip:user@ip:port). Note that the main purpose of a SIP registrar is to allow calls using just the username part (extension number) without the need for the enduser to know the target network address and it allows calls also to other networks (via any NAT). Using the full URI can be uncomfortable for endusers thus we recommend to use the webphone with a SIP server to ease the usage.
- Peer to peer media: direct media routing is important to minimize the audio delay and to free up your server side resources. This is fully supported by the webphone and used whenever possible. Note that this also depends on server side capabilities (SIP servers can act as a B2B relay, removing all the necessary information from the signaling that would be required to setup a direct media stream between peers thus forcing RTP routing via the server media relay)
- Sometimes it might refer to peer to peer encryption (or end to end E2E encryption) which means that the server (if used) is a passive party from the encryption point of view and is unable to decrypt the streams between the endpoints (just forwards the stream if needed). End to end encryption is supported by the webphone and it is applied whenever direct media routing is possible. The webphone also has support for peer to peer encrypted media with direct streaming (this is done via ICE techniques with automatic failback to routing via server if a direct path can't be found)
- Server assisted phone to phone calls. This means that both endpoints will be called by the server and once connected, the server interconnects the 2 endpoint. It can be useful when the client device doesn't have internet connection or doesn't have any platform to enable VoIP, such as an old dumb phone. Exactly for this concept we refer with the P2P engine.
- Peer-to-peer distributed application architecture: this has nothing to do with the webphone capabilities. The SIP protocol itself is a client-server protocol and doesn't define any automatic internetworking capabilities, however if you wish, you can use an external library to construct a peer-to-peer network of SIP nodes. This is usually worthless as for signaling you can use a thin proxy capable to handle millions of users and for media we already have capabilities to establish direct connection between peers (STUN, ICE). There is no standard way for distributed SIP network architecture (you can implement any architecture after your needs which is supported by all your network elements).

SMS

In order for the webphone to be capable sending SMS messages, your softswitch need SMS support.
Usually this is done by using SMS gateway services such as clickatell or a directly connected SMS capable phone device.
Note: such kind of services can't be used directly from the webphone, because you need to implement billing on your SIP server. So the messages have to be sent to your SIP server first and from there it can be forwarded to a SMS gateway.

From webphone to server the SMS can be sent via one of the following methods:

- via HTTP API: just configure your server SMS API via the "sms" webphone parameter (you can set this in the webphone_config.js file).
  For example:
  sms:'https://myserver.com/api/function=sms&authkey=5829157329773&authid=4753584&authmd5=xxx&anum=MYNUMBER&bnum=TARGETMOBILE&txt=TEXT' (upper case keywords will be replaced automatically by the webphone).
- via chat/IM: instead of calling an API, the SMS messages can be also carrier via SIP MESSAGE (RFC 3428). In this case your server have to check the destination number and if that is a mobile number then it should forward the message as SMS instead of chat. You can also set the textmessaging webphone parameter to 5 which will insert a X-Sms: Yes header (the server might check this header to know when char to SMS conversion is required)

If you are using the mizu softswitch with the webphone, then the sms functionality will be automatically preconfigured in your webphone build if you have configured sms on your server (via the smsurl global config option or by adding separate SMS gateways in "Users and devices")

If your server has sms support and you have configured in the webphone as described above, then you can send SMS from the webphone in the following ways:

- if you are using the softphone skin: send SMS as you were sending normal chat messages
- if you are using the webphone as a JavaScript library: use the sendsms API

Receiving SMS messages:

The standard way for receiving messages in a VoIP network is via the SIP MESSAGE protocol, thus your SIP server will just need to convert SMS message to a SIP chat message and deliver it as simple IM.

If this feature is not supported by your server, then the webphone can also poll an API for new incoming SMS messages, although this should not be used in large networks since this kind of polling is not very efficient.

Another way to implement incoming SMS is to use push notification, however this has nothing to do with the webphone SIP stack and it should be implemented separately.

Presence

Presence can be used to publish your webphone state for others or to learn about the state of the peer devices (online, offline, etc).

You can use the presence functionality for this if your server has support for the standard presence SUBSCRIBE/NOTIFY.

Set the enablepresence parameter to

See the checkpresence/setpresencestatus/onPresenceStateChange functions for the details

BLF

In case if you wish to publish or learn also about peers call state, then you might use the BLF (busy lamp field) features (instead or in addition to presence).

Set the enableblf parameter to 2 then pass the userlist (list of extensions) to be monitored either by the blfuserlist parameter or with the checkblf(userlist) API.

Then just use the onBLFStateChage callback to watch for the subscribed extensions call status changes.

Call Forward vs Call Transfer

Here are the main differences between SIP call forward and transfer:

Call forward:

- In the webphone you can forward calls by using the callforwardonbusy, callforwardonnoanswer, callforwardalways parameters or the forward() API
- It is like HTTP redirect
- It is done by sending a 301 or 302 (moved) disconnect code to the caller (specifying the new target)
- Can be initiated by the called party only
- Not all SIP servers has support for this and some servers explicitly disables this functionality to avoid user confusion
- Can be initiated only before call connect (as a response for the incoming INVITE)

Call transfer:

- In the webphone you can forward calls by using the calltransferalways parameter or the transfer() API
- It is done by the SIP REFER method (sending the new target by the Refer-To SIP header)
- Can be configured with the transfertype parameter
- Unattended transfer means a simple REFER request and the other party should call the new target
- Attended transfer means consultation call first (supervised transfer)
- Can be initiated by both the caller and called party
- Not all servers has support for SIP REFER (check your server specification and settings)
- Can be initiated only after call connect

Transfer API usage

The transfer mode can be set with the transfertype parameter.

The call transfer is handled with the SIP REFER method as described in RFC 3515, RFC 5589 and RFC 6665.

With **unattended transfer** the transfer will be executed immediately once you call the transfer function (blind transfer; only a SIP REFER is sent).

With **attended transfer** (transfertype 5) you can use the following call-flow, supposing that you are working at A (webphone) side and wish to transfer B (the original call) to C (the transfer target):

1. A call B (outgoing) or B call to A (incoming)
   A speaking with B
2. Call transfer(C)
   The call between A and B will be put on hold by A
   A will call to C and connect (consultation call; if call fails, then the call between A and C will be un-hold automatically)
   A speaking with C
3. The actual call transfer will be initiated when A disconnect the call (hangup())
   REFER message will be sent to B (which tells to B to call C. usually by automatically replacing the A-B call with A-C)

4. After transfer events:
   Transfer related notifications (SIP NOTIFY) can be sent between the endpoints once the call transfer is initiated, reporting the transfer state.
   - If transfer fails (B can't call C) the call between A and B will be will be un-hold automatically (if server sends proper notifications)
   - If the transfer succeeds (B called C) the call between A and B will be will be disconnected automatically (if server sends proper notifications)
   - If the server doesn't support NOTIFY, then the call can be un-hold or disconnected from the API (hold(false), hangup())
5. B speaking with C at this point if the transfer was successful

If you use call transfer, you will have multiple lines and you might handle the lines explicitly as described here.

The softphone skin (in case if you are using the softphone.html) call transfer user interface depends on the transfertype and transferpopup parameters.
If the transfertype is set to attended and the transferpopup is set to -1 or 2 then once the call was initiated to the transfer target (consultation call) an additional transfer control popup will appear with the following buttons:

- Transfer: will transfer the call when clicked and will hangup the original call (otherwise the call will be transferred only at consultation call disconnect)
- Revert: close the popup and do nothing (the call will be transferred only at consultation call disconnect)
- Hangup: will disconnect the consultation call and will un-hold the original call

Create custom attended transfer

In case if attended transfer doesn't work or if you wish to do it in a different way then the webphone does, you can also perform attended transfer from your code while the webphone transfertype is set unattended transfer (1, 6 or 7).
In this case just initiate the consultation call yourself (using the call function) and then use the transfer function (with unattended transfer) whenever you wish the actual call transfer to happen (usually after hangup with the target).
Step-by-step example:

1. Set the transfertype parameter to 7 and do the followings on transfer.
2. A is the webphone, B is the original caller/called and C is the transfer target.
3. Put the old line with B on hold: hold(true)
4. Call the new number for the consultation call: call('C')
5. If the consultation call to C fails, then reload the call with B: hold(false)
6. Watch if any peers disconnects from the onCallStateChange() callback. If peer B disconnects, then there is no need for any transfer anymore. If peer C disconnect (after the consultation call) , then do the transfer right away
7. Transfer the B to C by using the transfer('C') function once the consultation call with C ends
8. Disconnect from C using the hangup() function (if not already disconnected. You might disconnect before to make the transfer but in that case transfer with replaces will not work, so a new call will be initiated to C instead of reusing the same old call-leg)

Since you are operating with multiple lines here, you might need to use also the setline(x) function before the above function call to specify the exact line.
See the How to manage multiple lines FAQ point for the details about this topic.

Transfer related parameters

Note: not all parameters are applied for all engines.

transfertype

Specify transfer mode for native SIP.
-1=default transfer type [default; same as 6]
0=call transfer is disabled
1=transfer immediately and disconnect with the A user when the Transf button is pressed and the number entered (unattended/blind transfer)
2=transfer the call only when the second party is disconnected (attended transfer)
3=transfer the call when the VoIP phone is disconnected from the second party (attended transfer)
4=transfer the call when any party is disconnected except when the original caller was initiated the disconnect (attended transfer)
5=transfer the call when the VoIP phone is disconnected from the second party. Put the caller on hold during the call transfer (standard attended transfer)
6=transfer the call immediately with hold and watch for notifications (unattended transfer)
7=transfer with no hold and no disconnect (simplest unattended transfer)
8=transfer with conference (will put the parties to conference on transfer; will mute or hold the old party by default)

Note:

- *Unattended means simple immediate transfer (just a REFER message sent)*
- *Attended transfer means that there will be a consultation call first*
- *It is also possible to simulate an attended transfer by initiating a simple call to the transfer target and on hangup an unattended call-transfer (transfertype 1)*
- *If you have any incompatibility issue, then set to 7 (unattended is the simplest way to transfer a call and all sip server and device should support it correctly)*

transfwithreplace

Specify if replace should be used with transfer requests, so the old call (dialog) is not disconnected but just replaced.
This way the transferee and the transfer-target parties are not disconnected, just the other party is changed at runtime.
The feature is implemented by adding a Replaces= for the Contact header in the REFER request as described in RFC 5589 and RFC 5359.
The transferee (the party which receives the REFER transfer request) or the SIP server must be able to handle replaces for this to work.
Possible values:
-1: auto / default (if server/peer has replaces support and we are connected with the transfer target)
0: no
1: yes

replacetype

Specify how to handle incoming call transfer with replaces requests (How to handle the Replaces= Contact tag in incoming REFER requests)
-1: auto (defaults to 1)
0: disable

1: standard (sending the Replaces header in the new call INVITE request as described in RFC 3891)
2: in place (might be useful with servers without replaces support; not supported by the WebRTC engine)

transferpopup
Specify softphone skin attended transfer behavior.
-1: auto
0: hold current line and call the new number (might transfer at call disconnect only)
1: call the new number first (without holding the old line first; the call will be transferred at disconnect)
2: with additional popup (possibility to transfer, hangup or revert)

defmute
Default mute direction:
0: both
1: mute out (speakers)
2: mute in (microphone)
3: both
4: both
5: disable mute

Default: 0 or 2

discontransfer
[For the NS and Java engines only. For WebRTC only the 0/non-0 is considered]
Specify if line should disconnect after transfer
-1=auto [default; disconnect if transfertype is 1]
0=never
1=on C party connected status
2=on timeout
3=on connected or timeout
4=on ok for refer
5=immediately on refer sent

disconincomingrefer
[For the NS and Java engines only. Ignored for the WebRTC and Flash engines]
Specify if line should disconnect after transfer
-1=auto [default]
0=no
1= yes

allowreplace
[For the NS and Java engines only. Ignored for the WebRTC and Flash engines]
Allow incoming replace requests.
0=no
1=yes and always disconnect old ep
2=yes and don't disconnect if in transfer (default)
3=yes but never disconnect old ep

inversetransfer
[For the NS and Java engines only. Ignored for the WebRTC and Flash engines]
Specify inverse attended transfer.
0: no. The REFER will be sent to the ep for which the transfer function was called. This is the standard behavior. [default]
1: yes. The REFER will be sent to the ep for which the hangup function was called. Might be used only if the original ep (for which the transfer was called) doesn't support transfer (REFER sip message).

transferdelay
[For the NS and Java engines only. Ignored for the WebRTC and Flash engines]
Milliseconds to wait before sending REFER/INVITE while in transfer.
Default value is 400.

newdialogforrefer
[For the NS and Java engines only. Ignored for the WebRTC and Flash engines]
Specify if the REFER have to be sent in a new dialog (for compatibility reasons).
0: no [default; after SIP standards]
1: yes, with no to tag

autohold
Specify if other lines will be put on hold on new call.
0=no [default]
1=on incoming call
2=on outgoing call
3=on incoming and outgoing calls
4=on line change
5=on outgoing call and on line change (like 2+4)
6=on any call and on line change (like 3+4)

automute

Specify if other lines will be muted on new call.
0=no [default]
1=on incoming call
2=on outgoing call
3=on incoming and outgoing calls
4=on line change
5=on outgoing call and on line change (like 2+4)
6=on any call and on line change (like 3+4)

holdtype
Specify how the call hold function should work:
-2=no (will ignore hold requests)
-1=auto/default (defaults to 2)
0=no
1=not used
2=hold (standard hold by sending "a=sendonly")
3=other party hold (will send "a=recvonly")
4=both in hold (will send "a=inactive")

holdontransfer
(int)
Specify if initial line should be put on hold on transfer.
-1=auto (default; means 3 for transfertype 5 and 6, otherwise 0)
0=no
1=yes, hold before transfer
2=yes, hold before transfer and reload if needed (on transfer failure)
3=yes, hold on successful transfer init (OK for REFER received or attended call progress or success received; transferred call might not be initiated/connected yet). Will reload if needed (on transfer failure).

unholdontransfer
(int)
Unhold the line just before the actual transfer if it was holded and the transfer is made with replaces (to prevent keeping the endpoint in hold)
-1=auto (default; usually same like 1)
0=no
1=yes with replaces if there was hold before
2=yes always

How to auto start / auto register

Library initialization

The webphone library will auto initialize itself by default at page load.
You might use the autostart parameter to modify this behavior or use the start() API.

Softphone skin auto login

In case if you are using the softphone.html and the SIP account parameters (serveraddress/username/password/others if needed) have been preconfigured or already entered by the user, then the softphone skin will not stop at the login screen.
You might use the autologin parameter to modify this behavior.

Auto register

The webphone will automatically register to the SIP server upon startup if the SIP account parameters (serveraddress/username/password/others if needed) have been preconfigured or entered by the user.
You might use the register parameter to modify this behavior or use the register() API.

Other not so important related parameters: appengine_startat (native engine launch), startsipstack (NS engine specific).
See this FAQ point if the webphone doesn't start correctly.

Register vs Login vs Credentials

These terms might be also misleading especially for user with no VoIP/SIP knowledge.
Register or registration provides a way for the SIP clients to connect/login to the server so the server will learn the client address and will be able to route calls and other message to it. It is implemented by sending a REGISTER message by the SIP signaling. The server might or might not challenge the request with an authentication request (in this case the client will send a second REGISTER with a hash of its credentials). On credentials we refer to the sip username/password.
However:

- Register is optional and is not really needed if your client will make only outbound calls (not used to accept calls or chat)
- You can configure your server to not require registrations (actually most server doesn't require it by default, however in some servers the default configuration is to not allow calls if there was no previous successful registration)

- For the webphone you can set the “register” parameter to 0 to skip registration (so the webphone will not send REGISTER requests)
- Disabling registration is not a security treat since the server will do the same authentication for each call as it does for registrations (so the clients will not be able to make calls if their credentials are incorrect)
- You can also configure your server to allow blind registrations. This means that the client might send the REGISTER with any credentials (any username/password) and it will be unconditionally accepted
- You can also configure your server to allow blind calls. This means that the client might send the INVITE with any credentials (any username/password) and it will be unconditionally accepted (the call will be routed)
- If your server accepts blind registrations and calls then you can set the webphone password parameter to any value since it will not be checked or used anyway. (You can set it to “nopassword” as a special value to hide it from settings and login forms)
- There are situations when even the username doesn’t matter (if you wish to make only unconditional outbound calls or calls to ivr). However you must also set the username parameter to some value or allow the user to enter something since it is required for the SIP messages. You might set it to “Anonymous” in this case.

If your users will have to login to your webportal, then we recommend using the same username/password for WEB and SIP. You might need to do some integration work between your web service and VoIP service for this, for example synchronize the users or use a single source (database table or configuration file) for both your web server and SIP server.
Sometime you might use a separate username/password combination on your website then on your SIP server. In this case you can auto-provision the webphone with the sip credentials if the user is already logged in on your website to avoid typing a different username/password. This can be implemented multiple ways:

- by dynamically generating the webphone settings from a server script (set the username/password from the server since there you already know the signed in user details and you can grab the SIP credentials from your softswitch database)
- implement a custom API which returns the sip credentials and set it’s URI as the “scurl_setparameters” parameter (webphone will call scurl_setparameters URI and wait for the (key/value) parameters in response and once received it will start the webphone)
- handle it from JavaScript (use the setparameter() API to set the username/password)
- implement some alternative authentication method on your SIP server (for example based a custom SIP header which you might set from the web session using the setsipheader() API call)

Alternative authentication methods
This is a JavaScript SIP client, so it has support for the standard SIP digest authentication (with SIP username/password). However, you are not forced to use this. If your VoIP server does not ask for authentication, then you can just configure a random username/password (it will not be used anyway) and/or you can use any other authentication as required by your platform.
For example, you can use the dtmf function to send PIN code as DTMF digits, use the setsipheader function to pass any extra data with the signaling, add a tech prefix before the called numbers or any other method.

How to register?

SIP registration means connecting to your SIP server and authenticating. From this procedure your SIP server will learn your application address and can route incoming calls to your application (or other sessions, such as chat, presence, voicemail, etc).

The SIP stack auto-start behavior can be altered with the autostart setting or you can use the start function to launch the instance explicitly.
When started, the SIP stack by default (unless you set the register parameter to 0) will automatically register to your SIP server if you configured the SIP account details (serveraddress, username, password and any other parameters that might be required such as the proxyaddress and sipusername).

Otherwise you might disable auto-start (autostart) and/or the auto register (register), pass the above parameters dynamically from your code and use the start and/or the register function to initiate the start/connect/register procedure.

Check this FAQ point if you wish to use multiple accounts in one webphone instance.
Check this FAQ point if you are having problems with connect/register.

How to find out registration status

Depending on the settings, the webphone will automatically register upon startup or you can explicitly connect to the server by calling the register() API.
To find out whether the webphone is successfully registered or not, you can use the isregistered() API to query the status at any time.
You can also receive notifications about the registration status via the followings callbacks:

- onRegStateChange called on register state change
- onEvent called with display type parameter when register fails with the message containing one of the following text:

- Connection lost
- No network
- Server address unreachable
- No response from server
- Server lost
- Authentication failed
- Rejected by server
- Register rejected
- Register expired
- Register failed

In old versions you can also use the getregfailreason function to find the reason of a failed connect/register attempt.

To get the full SIP answer message for the REGISTER request then you can use getsipmessage function like this:

```
webphone_api.getsipmessage(0, 2, function (sipmsg) {
        if(sipmsg.indexOf("SIP/2.0 ") == 0 && sipmsg.indexOf(" REGISTER\r") > 0)  //contains CSeq: x REGISTER
        {
                //the answer for the REGISTER request is in the sipmsg variable now and you can parse it as you wish
                //you might use something like this if you are interested only in register failures:
                sipmsg = sipmsg.substring(sipmsg.indexOf(' ')+1);
                sipmsg = sipmsg.substring(0, sipmsg.indexOf(' '));
                var code = parseInt(sipmsg.trim(),10);
                if(code >= 300 && code != 401 && code != 407)
                {
                        //code 401 or 407 can be normal here (server asking for auth) and can be skipped if the onRegStateChange doesn't report 'failed'
                        console.log("Register failure code: "+ codenum.toString());
                }
        }
});
```

Disconnect reasons

You can receive the call disconnect reason with the onCDR event ("reason" parameter) or using the getlastcalldetails() function.
There are endless possibilities why a call can fail, the most common reasons are listed below.
The disconnect reasons are reported in the following format: code text. (So you have the text after a space)
Code:

Is a SIP disconnect request or answer code including BYE, CANCEL or any SIP response code above 300.
If no disconnect message were received or sent then the code is -1 or empty/not set.

Text:

The text is one of the followings:

1. local disconnect reasons (listed below)
2. disconnect reason extracted from SIP Warning or Reason headers
3. response text extracted from the first line of SIP responses (textual representation of the response code)
4. textual representation of the disconnect code

The local disconnect reasons can be one of the followings (extra details might be appended and new disconnect texts might be added in the future):

- notaccepted
- User Hung Up
- bye received
- cancel received
- authentication failed
- endpoint destroy
- no response
- call setup timeout
- ring timeout
- media timeout
- endpoint timeout
- tunneling calltime limit
- call max timer expired
- max call time expired
- max speech time expired
- not acked connection expired
- disconnect on transfer
- transferalways

- transfer timeout
- transfer fail
- transfer done
- transfer terminated
- transfer (other)
- refer received
- cannot start media
- not encrypted
- srtp fail
- srtp init fail
- disc resend
- failed media
- forward
- forwardonbusy
- forwardonnoanswer
- forwardalways
- rejectbusy
- rejectonphonebusy
- call rejected by peer
- rejected by the peer
- rejected
- autoreject
- autoignore
- ignored

The SIP disconnect codes (3xx, 4xx, 5xx, 6xx) are described in the SIP RFC.

With the getlastcalldetails() function, "Disc By" text, you can also get the party which was initiated the disconnect.
This can be one of the followings:

- 0: unknown/not set
- 1: local webphone
- 2: remote peer
- 3: undefined/timeout

You can also get the exact disconnect reason / disconnect message from the SIP signaling using the getsipmessage function.
For example to extract the disconnect reason from the last incoming message, you can use something like this:

```
webphone_api.getsipmessage(0, 0, function (sipmsg) {
	var disconnectcode = "";
	if(sipmsg.indexOf("BYE ") == 0) disconnectcode = "BYE";
	else if(sipmsg.indexOf("CANCEL ") == 0) disconnectcode = "CANCEL";
	else if(sipmsg.indexOf("SIP/2.0 ") == 0)
	{
		sipmsg = sipmsg.substring(sipmsg.indexOf(' ')+1);
		sipmsg = sipmsg.substring(0, sipmsg.indexOf(' '));
		var codenum = parseInt(sipmsg.trim(),10);
		if(codenum >= 300) disconnectcode = codenum.toString();
	}
	if(disconnectcode.length > 0) console.log("The disconnect code is: "+disconnectcode);
});
```

Caller ID display

For outgoing calls:
The Caller-ID (CLID/CLI/A number display) is controlled by the server and the application at the peer side (be it a VoIP softphone or a pstn/mobile phone).
You can use the following parameters to influence the caller id display at the remote end:

- username (SIP user name / Extension number / User ID. This is used as both SIP username and authentication username if sipusername is not set)
- sipusername (Auth user name / Auth ID / PIN / Authorization name. If this parameter is set, then the "sipusername" will be used for authentication and the "username" parameter as the SIP user name and caller-id)
- displayname (SIP display name / Caller ID)

You might also set the showusername parameter to 2 to show both the username and the caller-id input on the softphone skin login page.

Some service providers will provide you a globally routed telephone number (DID) which might be applied on the server side or you might have to use it as the username parameter.
If you set all these parameters, then it will be sent in the SIP signaling in the following way (see the uppercase worlds):

```
INVITE sip:called@sipdomain.com SIP/2.0
From: "DISPLAYNAME" <sip:USERNAME@sipdomain.com>;tag=xyz
Contact: "DISPLAYNAME"<sip:USERNAME@sipdomain.com>
Remote-Party-ID: "DISPLAYNAME" <sip:USERNAME@88.150.183.87>;party=calling;screen=yes;privacy=off
Authorization: Digest username="SIPUSERNAME",realm="sipdomain.com" …
```

The WebRTC engine will send the "Contact" URI with an invalid domain by default for outgoing REGISTER/INVITE/etc requests as required by RFC 7118.
You might set the contactdomain webphone parameter to 1 if your server expects its valid domain for this.

VoIP servers might completely ignore the display name and/or the Caller-ID coming from the client and will overwrite with the account/extension settings or according to their dial-plan. If your server is managed by you, then you can configure it to always forward the values from the clients. Otherwise check if you have some enduser control panel web interface where you can set this.
Some VoIP server will suppress the CLI if you are calling to pstn and the number is not a valid DID number or the webphone account doesn't have a valid DID number assigned (You can buy DID numbers from various providers).
The CLI is usually suppressed if you set the caller name to "Anonymous" (hide CLI).
If required by your SIP server, you can also set a Caller Identity header as a "customsipheader" parameter. (P-Preferred-Identity/P-Asserted-Identity/Identity-Info)

For incoming calls:
In case if you are using one of the prebuilt skins:
For incoming calls the webphone will use the caller username, name or display name to display the Caller ID. (SIP From, Contact and Remote-Party-ID fields).

If you are using the API then you can get the caller-id (caller number / caller username / CLI / caller extension) by using one of the followings methods:

- the onCallStateChange callback (you receive the caller name and display name at each call state change in the peername/peerdisplayname variables)
- the getcallerdisplayfull callback (detailed caller id on incoming calls)
- the onCdr callback (caller details at the end of the call)
- the getlastcalldetails function (more details can be requested after the call)
- the getsipheader function can be used to request any received SIP header (such as the "From" or the "Remote-Party-ID" headers)
- the getsipmessage function can be used to request the received SIP message in raw text (such as the whole incoming INVITE message)

My server requires separate extension and authentication username

Some SIP servers might require a different user name and auth user name.
This means that for the webphone you must set both the username and sipusername parameters or if you are using the softphone skin then you must set both the Username field on the login screen and the Caller-ID field in the settings.

From the softphone skin (softphone.html):

- On the login screen:
  - set the "Server" field to your server address (domain or IP:port)
  - set the "Username" field to the Authentication username (example: '1qu3iqt') . This will be saved as the "sipusername" parameter.
  - set the "Password" field to the Authentication Password (example: 'xkclq7n')
- Go to the Settings and set the "Caller ID" field to the Extension number (example: '04'). This will be saved as the "username" parameter.
- Then just login.
  (and you might set the showusername parameter to 2 to display both the username and the auth username input on the login screen)

Via webphone parameters (configurable in the webphone_config.js):

```
webphone_api.parameters = {
        serveraddress: '11.22.33.44',  //your server domain:port or IP:port
        sipusername: '1qu3iqt', //Authentication username (Auth ID)
        username: '04', //Extension number (SIP user name or Caller-ID)
        password: 'xkclq7n',  //Authentication Password
        loglevel: 5
        //comma is not needed after the last parameter
};
```

You can configure also by passing the above settings via the setparameter API or by URL query parameters.

Please note that you can create a login user interface that requires both the Auth user name and the Auth ID, so the users don't have to go to settings details.
Set the showusername parameter to 2 to show both the username (Caller-ID) and the sipusername (username for authentication) input on the login page.
If you expect the @ character (full SIP URI) in the sipusername, then set the handleusernameuri parameter to 0 to prevent extracting the domain from it on the softphone skin.
For production you might reconfigure this on your server (if possible) to not require a separate user name and auth user name (by accepting the auth username also as the username without the need to set the extension number). This is important only if the endusers needs to type these credentials themselves (to simplify their login process) and it is not so important if you automate it with some process.
More details about caller-id processing can be found here.

Special characters in SIP username-password

The webphone filter out special characters because the SIP standard has some related flaws and a lot of SIP servers don't allow such characters.
Some SIP implementations treat the digest authentication as ASCII text while others treat it as Unicode.
To prevent compatibility issues, the webphone will warn the users if their account credential contains special characters.

The built-in algorithm is very simple:

```
if (input != encodeURIComponent(input))
{
        //don't allow and display error
}
else
{
        // input is ok. go ahead with it
}
```

This means that we don't allow special characters and some implementations might block also the followings especially as username: , / ? : @ & = + $ #.
See the RFC 3986 about URI reserved characters.

How can I change the SIP signaling

You can add any SIP features after your needs by changing the SIP headers.
This can be done with the customsipheader parameter or with the setsipheader API.
For example to add asserted identity, you might use one of the followings as required by your server:

- P-Asserted-Identity: <sip: john@yourdomain.com>
- Remote-Party-ID: <sip: john@yourdomain.com>;party=calling; privacy=off
- P-Preferred-Identity: "John Smith" sip:john@ yourdomain.com
- Privacy: id

You can receive the incoming SIP messages using the getsipheader or getsipmessage API's (then parse them after your needs from your JavaScript code).

How can I make a call?

1. Make sure that the "serveraddress" parameter is set correctly (otherwise you will be able to make calls only to direct SIP URI).
2. Optionally: Register to the server. This can be done automatically if the "username" and "password" parameters are preset. Alternatively you can register from API (Register) or just let the user to fill in the username/password fields and click on the "Connect" button. If the webphone is registered, then it can already accept incoming calls (it will do it automatically, or you can handle incoming calls from your application, or you can entirely disable incoming calls)
3. Now you can make outgoing calls in the following ways:
    a. Automatically with the webphone startup. For this you will have to preset the username/password/autocall and callto parameter. Then webphone will immediately launch the outgoing call when starts (usually with your page load)
    b. Just let the users to enter a called number and hit the "Call" button (if you are using on of the included html's)
    c. or just call the call function (from user button click or from your business logic)

Read through the parameters to find out more call divert settings, such as auto-answer or forward.

How to handle incoming calls?

To be able to receive incoming calls, make sure that you are registered to your SIP server first. Then you can easily test for example by using any third party softphone and make calls to your application SIP username (or extension id or full URI, as required by your server).

The webphone is capable for automatic line management so if you don't wish to handle lines explicitly in some specific way, then you can ignore all notifications, except those where the line (first parameter) is -1 (the global state).
If you wish to auto accept all incoming calls, you can set the autoaccept parameter to true. Otherwise use the accept to connect the call (or the reject to disconnect).

You can listen for call state changes (including for incoming call setup) using the onCallStateChange callback.

Here is a simple example:

```
//you might place this callback inside the on onAppStateChange "loaded" event as discussed here
webphone_api.onCallStateChange(function (event, direction, peername, peerdisplayname)
{
  if (event === 'setup') // call init
  {
    if (direction == 1)
    {
      // means it is an outgoing call
    }
    else if (direction == 2)
    {
      // means it is an incoming call. you might display your call notification by something like this:
      document.getElementById('icoming_call_layout').style.display = 'block'; // display Accept, Reject buttons
      /*
       <div id="icoming_call_layout">
         <button onclick="webphone_api.accept();">Accept</button>
         <button onclick="webphone_api.reject();">Reject</button>
       </div>
      alert('Incoming call from '+peerdisplayname);
      */
    }
  }
  else if (event === 'disconnected') // call disconnect
  {
    //you might hide Accept, Reject buttons by something like this:
    document.getElementById('icoming_call_layout').style.display = 'none';
  }
});
```

More details and examples can be found here.
Simple working examples can be found also among the html's in the samples folder.
Details about incoming call alert options can be found here.

How to get the call duration?

To get the call duration, you can use the onCdr() callback API function like this:

```
// you will receive call duration on oncdr() callback
// NOTE: the "connecttime" and "duration" is expressed in milliseconds
  webphone_api.onCdr(function (caller, called, connecttime, duration, direction, peerdisplayname, reason, line)
  {
    // convert string value to integer;
    var durationInt = parseInt(duration,10);
    var durationSec = Math.floor((durationInt+500)/1000);

    alert('Call duration is: ' + durationSec + ' seconds');
  });
```

Full example here (see the html source).

In case if you need the call duration during the call (for example to display a call timer), then you can just should start a javascript timer at call connect and measure the elapsed time.

Web SIP client setup for Asterisk

You can easily use the webphone with Asterisk and with any Asterisk derivatives such as FreePBX, Elastix or Trixbox.
There is no any special requirement for the webphone. You can use the webphone like any other usual SIP or WebRTC extension (as other regular SIP device, softphone or webrtc endpoint).

The webphone is fine-tuned for Asterisk out of the box and no changes are needed to.
The following settings for webphone extensions might help in some circumstances (in the Asterisk config):

- dtmfmode=info
- nat=yes

We created two tutorials to help Asterisk beginners with the webphone integrations, useful especially if you have some special requirement, such as using the built-in WebRTC module in Asterisk:

1. Using the webphone as a regular JS SIP client: Asterisk web SIP client
2. If you wish to use the Asterisk built-in WebRTC module: WebRTC client for Asterisk
   More about webphone WebRTC can be found here.

We recommend to set the NAT parameter to yes (nat=yes or nat=force_rport,comedia for new versions) for your endpoints in the pjsip.conf if you are using your Asterisk server over the internet.
More details:
https://www.voip-info.org/wiki/view/NAT+and+VOIP
https://www.voip-info.org/wiki/view/Asterisk+sip+nat
https://www.asteriskguru.com/tutorials/sip_nat_oneway_or_no_audio_asterisk.html

If you experience no audio or one way voice issue, try to set the use_fast_stun, use_fast_ice and use_rport webphone parameters to 0.
You might also check chat and presence settings if required for your use-case.

If you have some special requirement, such as using the built-in WebRTC module in Asterisk, check these articles:

- Setup Web SIP client for Asterisk
- Asterisk WebRTC setup

Web SIP client setup for 3CX

3CX server (and some other softswitch software) by default might require a different user name and auth user name.
This means that for the webphone you must set both the username and sipusername parameters or if you are using the softphone skin then you must set both the Username field on the login screen and the Caller-ID field in the settings.

From the softphone skin (softphone.html):

- On the login screen:
  - set the “Server” field to the 3CX address (domain or IP:port)
  - set the “Username” field to the 3CX Authentication ID (example: ‘1qu3iqt’). This will be saved as the “sipusername” parameter.
  - set the “Password” field to the 3CX Authentication Password (example: ‘xkclq7n’)
- Go to the Settings and set the “Caller ID” field to the 3CX Extension number (example: ‘04’). This will be saved as the “username” parameter.
- Then just login.

Via webphone parameters (configurable in the webphone_config.js):

```
webphone_api.parameters = {
        serveraddress: '11.22.33.44:5060',  //your 3CX server domain or IP:port
        sipusername: '1qu3iqt', //Auth ID (Authentication ID)
        username: '04', //Extension (Extension number or Caller-ID)
        password: 'xkclq7n',  //Authentication Password (Auth Pass)
        display name: 'John Kennedy',  //Account name (no so important)
        loglevel: 5
};
```

You can configure also by passing the above settings via the setparameter API or by URL query parameters.

Please note that you can create a login user interface that requires both the Auth user name and the Auth ID, so the users don't have to go to settings details. Set the showusername parameter to 2 to show both the username (Caller-ID) and the sipusername (username for authentication) input on the login page. For production you might reconfigure this on your 3CX to not require a separate user name and auth user name (by accepting the auth username also as the username without the need to set the extension number). This is important only if the endusers needs to type these credentials themselves (to simplify their login process) and it is not so important if you automate it with some application.
More details about caller-id processing can be found here.

Web SIP client setup for OnSIP

Similarly with the above discussed 3CX, OnSIP servers usually requires a separate username and sipusername to be set and usually also requires an outbound proxy server setting.

For example if your account details (as provided by your OnSIP service provider) looks like this:

- Address of Record: john@america.onsip.com
- SIP Password: secret
- Auth Username: america_john
- Username: john
- Domain: america.onsip.com
- Outbound Proxy: sip.onsip.com

Then these are the parameters to be set for the webphone in the webphone_config.js (for the parameters object at the beginning of the file):

- serveraddress: 'america.onsip.com'
- proxyaddress: 'sip.onsip.com'
- username: 'john'
- sipusername: 'america_john'
- password: 'secret'

Of course, the username/sipusername/password parameters can be asked from the user at runtime instead to be set statically in the configuration or you can pass these parameters also via the SetParameter API.
More details about caller-id processing can be found here.

Web phone setup with my SIP server

There is nothing special to set the webphone with any SIP server. The webphone works just like any other SIP client (like an IP phone or a softphone such as X-Lite). We mentioned Asterisk, OnSIP and 3CX above because they are popular and they require some specific settings, however with most other SIP servers all you need to do is to set the serveraddress parameter and with some SIP service providers you might have to set a separate username/sipusername (extension/auth id) and set also the proxyaddress parameter.

The parameters can be set for the webphone multiple ways as explained at the beginning of the parameters section.
The most important settings are listed here.

The most important setting is the serveraddress. Just set it to your SIP server domain or IP address. Make sure to append the port number if your server is not using the default port (5060). Example: serveraddress: 'mysipdomain.com:6789'.
The serveraddress (and other "static" parameters such as the proxyaddress) should usually be set in the webphone_config.js file.
Other user depending parameters (such as the username, sipusername, password) might be asked by the user and/or set at runtime with the setparameter API.

Some SIP servers or services might require also some other parameters to be set correctly:

- If you have a SIP proxy (such as an outbound proxy), then set it as the proxyaddress parameter
- Some servers might require a separate user name (extension) and auth user name (auth id). In this case you will need to set both the username and the sipusername parameters respectively as discussed for 3CX, OnSIP and also described here.
- Some SIP service providers provides globally routable telephone number (DID) which might be applied on the server side or you might need to use it as the username parameter (thus you might set your phone number as the webphone username parameter and the auth id as the sipusername parameter)
- If your server doesn't accept UDP transport, then you might need to set the transport parameter to TCP (or you can also set TLS/SRTP)
- If your server auth realm is different from its address, then you can set the realm parameter accordingly

- If your server doesn't accept registrations, then you might set the register parameter to 0
- If your server has WebRTC support, then configure the webrtcserveraddress accordingly. Read here for more details regarding WebRTC.
- You might also adjust some other settings such as dtmf mode, codec or the voicemail number however these are usually negotiated automatically
- For more tweaks, go through the parameter list and change anything after your needs. However we recommend to change any parameter only if you are sure, otherwise the default settings are the best settings and most of them are guessed or negotiated automatically at runtime

New settings not applied

If you have changed any parameter in the webphone_config.js, make sure that you see the latest version if you open the js file directly in the browser like: www.yourdomain.com/webphonefolderpath/webphone_config.js.js
If you don't see the recent settings that means that the old version was cached by your browser, by your webserver or some intermediary proxy.
The webphone might store/cache previous settings in cookie and indexDB "localforage".

Refresh the browser cache by pressing F5 or Ctrl+F5.
- In Chrome you can clear all settings related to the from Developers Tools (from the Menu or F12 or Ctrl+Shift+I) -> Application tab -> Clear site data
- In Firefox you can clear all settings related to the webphone by pressing ALT, then select "Show All History" from the "History" menu, then right click to your domain and select "Forget About This Site".

Check your webserver caching. You might disable caching by setting the Cache-Control header to no-store.
More details here.

Make sure that you don't have some caching proxy on the path. A sure way to bypass all caching is to change the server folder (deploy in a "test2" directory and launch from there). If you are using the NS engine, then you might need to upgrade the webphone service.

Once a parameter is set, it might be cached by the browser phone and used even if you remove it later.
To prevent this (instead of just deleting or setting an empty value) set the parameter to "DEF" or "NULL" or to its default value for string parameters. For number parameters instead of removing or commenting them out, you should change it back to their default value instead.

You might set the configversion parameter to a positive number (or increment it by one if already set) if you wish the webphone to forget the old settings.

If you have changed some parameter in the webphone_config.js then you might change its jscodeversion parameter where you include it in your project, to avoid any caching and force a re-download by the browser. For example: <script src="webphone_config.js?jscodeversion=1234"></script>

This is because browser aggressively cache js files regardless of your HTML expires and cache-control headers. You can read more about this here.

You can also use different profiles as described here.

Additionally you might force the webphone to forgot its old settings at startup with the resetsettings parameter or you can use the delsettings API to clear the settings at runtime (this should be used only when closing).

Also check this FAQ if you made a recent upgrade but still seems that the old version is running.
If still doesn't work, you should check from another PC (to make sure that nothing is preinstalled/cached on your PC).
If still doesn't work, send a detailed log to Mizutech support.

How to upgrade to a new version of the webphone?

Short answer:
Just overwrite all the webphone files that you haven't modified and merge the files where you made changes.
Typically, you only need to keep the old webphone_config.js file if you haven't modified any other webphone files.

Long answer:
You need to upgrade your webphone in the following situations:
1. If you have used the demo before and paid for the license, Mizutech will send your licensed copy
2. Upon a support request, Mizutech might send you a new build with changes regarding to your request (bug fix/improvement/new feature)
3. Occasionally you might choose to upgrade to the latest version if you see a new version published by Mizutech with features/changes of your interest or with any bug fix affecting your use-case (new versions will be provided by mizutech for free during your support period)

With the upgrades we usually make changes all over in the webphone code (js/html/css/native files).
However all the parameters and the API is fully backward and forward compatible, so you don't need to change any of your existing code or settings.

Before to upgrade, first you should backup your existing webphone folder.
Then extract the zip sent by Mizutech and replace the files in your webphone folder with the new content, but make sure to:
- preserve the settings: if you set some configuration in the webphone_config.js file, make sure to not overwrite this file
- don't overwrite files where you made changes if any (merge them if possible)

There are two main ways to work with the webphone:

1. With your code in a separate project/files by including the webphone_api.js into your project and using the webphone API to build your custom solution. In this case usually you don't make any changes in the webphone files and you can just overwrite the whole webphone folder once we send a new build, maybe except the webphone_config.js file which you should keep if you have set any parameters there.
2. Modifying/extending the webphone files (the html/css/js files included in the webphone).
   If you made changes in our files (such as modifying the softphone skin) then at least replace the "\js\lib" and the "\native" folders and try to merge the rest if/when possible.
   In this case make sure to not overwrite the files where you made modifications or merge the modifications.
   The core of the webphone is the webphone\js\lib.js files, this is where most our work are done and these files might differ depending on your license. You should never change these files and you should always overwrite them with the latest version we send (full customization/integration/any usage can be done without touching these files)*
   If you managed to achieve your goals by modifying the webphone files (and not working in a separate projects/your own files by including the webphone_api.js as suggested above), then just overwrite all files where you haven't made any changes and all should be fine. You must overwrite at least the followings:
   - all .js files from the webphone\js\lib folder
   - all files from the webphone\native folder
   - all missing files (if there are some new files in the new version not present in your old version)

In case if you are upgrading from the demo version to your licensed copy:
The important (licensing related) changes will be in the \webphone\js\lib\ and \webphone\native\ folders where most files are already obfuscated or in binary format anyway so most probably you haven't made any changes there. You can keep the rest as-is if you wish (or just replace only those files where there are no modifications or merge the modified files if any).

Note:
- Although the webphone_js.api file is rarely changed, we don't recommend writing code in this file. Use your separate js files for your project and just include the webphone_api.js script to your project.
- If you/your customers might use also the NS engine (you haven't deprioritized it): you might adjust the minserviceversion if you have this set to any value, otherwise you might have to upgrade the NS service manually or the new webphone will continue to use the old version (which is not a problem most of the time, but we don't recommend to use very old outdated versions).
- You might reinstall the NS engine on your test client PC if you have used this engine before, for the changes to be applied instantly: On windows just launch the WebPhoneService_Install.exe from the webphone\native folder.
- If you changed the parameters in the webphone_config.js then you might change its jscodeversion parameter where you include it in your project to make sure that the browser doesn't load an old cached file. For example: <script src="webphone_config.js?jscodeversion=1234"></script>
- New versions of the webphone are always backward compatible and API compatibility is always ensured except occasional minor/compatible changes so you can upgrade without any changes in your code. However each new version contains changes in the VoIP engines so you should always verify and test before to put in production to make sure that the webphone still fulfills your needs and downgrade to the previous version if you encounter issues (Then you might try the upcoming release again to see if your pending issue were fixed).

I got an upgrade for my feature/issue request, but nothings seems to be changed

Make sure that you are actually using the new version. Refresh the browser cache by pressing F5 or Ctrl+F5. Make sure that you don't have some caching proxy on the path. A sure way to bypass all caching is to change the server folder (deploy in a "test2" directory and launch from there) and/or use another PC or browser where the webphone was not loaded before.

If your webphone is using the NS engine, then it might be possible that the PC is running an old version. Upgrade the NS engine as explained here.
If you are having difficulties applying new settings, read here.

If still doesn't work, you should check from another PC (to make sure that nothing is preinstalled/cached on your PC).
If still doesn't work, send a detailed log to Mizutech support.

How to upgrade the NS engine?

In short:
The NS engine (if used by the webphone) should upgrade itself automatically if needed, however, you can also perform a manual reinstall in the following way:
- On Windows: Just run the WebPhoneService_Install.exe installer on the client PC from webphonepath\native folder to quickly upgrade to latest version.
- On Linux: Extract the webphone_ns_lin64.zip exe from the from webphonepath\native folder and run the webphone_ns_lin64.run executable with root user privileges (sudo).
- On MacOS (deprecated): Launch the webphone_ns_mac.zip from the from webphonepath\native folder which is an application container and will reinstall the NS engine.

Windows details:
In some circumstances under Windows OS the webphone might install an NT service named "Webphone" (This is the NS service plugin and it is installed only on user opt-in). The NS engine will run as a background service and it has a silent single-click installer. This is a full SIP stack running as a windows native executable and used by the webphone if set so or other engines (such as WebRTC) are not available. It is important to understand that this is a client side component, running on the endusers PC's (you don't need to install anything on your web server).

The webphone (javascript running in the browser) will auto detect the locally running NS engine to be used as a SIP stack and will communicate with the NS engine over a localhost websocket connection.

Time to time new versions of the engine are released, shipped with new versions of the webphone (you can find the engine installer in the native subfolder of the webphone package). This might result in situation when you already have a new version of the webphone but the users are still using an old version of the NS engine installed previously on their PC.

There are the following options to upgrade the NS engine to latest version:

- Auto-upgrade: the core of the ns engine is capable to auto upgrade itself if new versions are found (you can disable this by setting the "autoupgrade" parameter to 6). This is very conservative which will rarely trigger an upgrade (only if the current version is very outdated or the current version has a serious issue which have been fixed in the new version).
  (In the NS service there is a built-in SSL certificate for localhost. This is also capable for auto-upgrade when new certificates are found unless you set the "autoupgrade" to 5)
- Force upgrade: Set the minserviceversion webphone parameter to the (latest) version of the engine you have. This will trigger an automatic install. You might also set the nsupgrademode to force immediate upgrade
- Reinstall manually: just launch the \webphone\native\WebPhoneService_Install.exe (directly from the webphone package sent by Mizutech or deploy it first to your webserver and let the users to download from www.yourdomain.com\webphonepath\native\WebPhoneService_Install.exe). Reinstall from client PC is also explained here.
- Reinstall from the softphone skin: Users can install the latest NS engine from the softphone skin by just going to menu -> settings -> advanced settings -> sip settings -> voip engine -> select the NS engine. That will offer the download of the new version.
- Custom methods: You can offer the NS upgrades as you wish and when you wish from your custom solution. Just redirect the user to download the installer executable.
- Disable: you can disable the usage of the NS engine by setting the enginepriority_ns to 0 (completely disable) or 1 (deprioritize). You should do so only if other engines such as WebRTC are available in your environment.

You can also query the NS engine version with the following API calls:

- get_version_ns_num (callback): returns with the NS engine major version number which is used also for minserviceversion
- get_version_ns (callback): returns with the NS engine exe version

Linux details:
The webphone_ns_lin64.run script (from the webphone_ns_lin*.zip file) will copy the NS engine files to /opt/WebPhone_NS/ and will install it as a service (daemon).
The installer script will also auto detect local JRE if any, otherwise might install it automatically if needed.
The webphone (javascript running in the browser) will auto detect the locally running NS engine to be used as a SIP stack and will communicate with the NS engine over a localhost websocket connection.

MacOS details:
Launching the webphone_ns_mac.zip app will copy the NS engine files to /Applications/WebPhone_NS/ and will install it as a service.
The installer script will also auto detect local JRE if any, otherwise might install it automatically if needed.
The webphone (javascript running in the browser) will auto detect the locally running NS engine to be used as a SIP stack and will communicate with the NS engine over a localhost websocket connection.
Note: the NS engine for MacOS is deprecated now. Use WebRTC instead (the webphone will default to WebRTC on MacOSX).

Note: you can also ask Mizutech to send you a build which will force upgrade to latest version (forcensupgr/minimumservicebuild/BUILDNUMBER/API_GetBuildNumber)

How to uninstall or (re)install the NS engine

Windows:
In some situation under Windows OS the webphone might install an NT service named "Webphone". This is the NS service plugin running on the client PC and it is installed only on user opt-in.
Here are how you can disable/uninstall/reinstall if somehow you need to:

- Disabling: If you don't wish to use the NS engine, you can just disable the service (set startup type to Manual and Stop the service) or set the enginepriority_ns to 0
- Uninstalling: The service has its own uninstaller, so you can easily uninstall it from the Add/Remove Programs control panel. It can be also removed with the –uninstall parameter. Example: C:\Program Files (x86)\WebPhoneService\WebPhoneService.exe –uninstall.
- Re(installing): The install can be done from the softphone skin by just going to menu -> settings -> advanced settings -> sip settings -> voip engine -> select the NS engine. That should offer the download of the new version.
  You can also (re)install/upgrade manually by running the "WebPhoneService_Install.exe" from the webphone\native folder. (You can also download it from your webserver: http://yourdomain.com/path_to_webphone/native/WebPhoneService_Install.exe or from the webphone package provided by mizutech). Just run the executable and it will install the NS engine automatically (this should work even if the service is already running as it will automatically update your old version)
- Upgrade: see the above FAQ point

Linux:
Use this script to uninstall the NS engine daemon from any Linux distro.

MacOS:

Use this script to uninstall the NS engine service from macOS.

Note: There is no need to uninstall the old previous version before to install the new one. Then installer will take care about the details (will stop the old executable first and will overwrite the old files as needed)

How to upgrade from the old java applet websipphone?

Note: this is relevant only for our old customers using the old java applet based webphone.

This new webphone has an easy to use API, however if you wish to keep your old code, you can do so with minimal changes as we created a compatibility layer for your convenience. Follow the next steps to upgrade/migrate to our new webphone:

1. The root folder of the new webphone is the folder, in which "webphone_api.js", “webphone_config.js” and "softphone.html" files are located.
2. Copy the contents of the new webphone root folder, in the same folder where the old webphone's .html file is (merge "images" and "js" folders, if asked upon copy process).
3. In the <head> section of the .html file, where the old webphone is, replace line:
<script type="text/JavaScript" src="js/wp_common.js"></script>"
with the following lines:
<script type="text/JavaScript" src="webphone_api.js"></script>
<script type="text/JavaScript" src="oldapi_support.js"></script>

Note: Don't remove or add any webphone related Javascript file imports.
"jquery-1.8.3.min.js " file will be imported twice, but that is how it supposed to be, in order for the upgrade to work correctly.

For old webphone customers: please note that this new webphone is a separate product and purchase or upgrade cost might be required. The old java applet webphone have been renamed to “VoIP Applet” and we will continue to fully support it. More details can be found in the wiki.

How to activate voice recording?

The webphone has the capability to record calls, store in a voice file and upload it to your FTP server or web service.
To activate this feature, just set the set the voicerecupload parameter or use the voicerecord API after your needs.

All you need for call recording is to set the URL where the files will be uploaded.
The URL can be:

- FTP: in this case you have to run an FTP service somewhere
- or HTTP: in this case you will need to create a server side script which can accept the upload

This URL can be set:

- globally with the voicerecupload webphone parameter if all calls have to be recorded.
  Example with FTP upload (added in the webphone_config.js file):
  voicerecupload: 'ftp://user01:pass1234@ftp.foo.com/voice_DATETIME_CALLER_CALLED'
- or dynamically (for example for a user or per call) with the voicerecord Java Script API.
  Example with HTTP upload (add this somewhere into your JavaScript code):
  webphone_api.voicerecord(true, 'http://www.foo.com/myfilehandler.php/?filename=callrecord_DATETIME_USER.wav')

See the voicerecupload parameter and the voicerecord API for more details.

Notes:
Since the WebRTC engine is running from browser, it doesn't have local access such as disk access.
Activating call recording with the WebRTC engine might completely change the media path if you are using a WebRTC gateway. With other words, the media path is more optimized when call recording is not activated. This is only for WebRTC. For NS or Java you will have the same path.
If your SIP server is public, then we recommend to setup a public FTP server or Web service to handle the storage of the voice files and configure your webphone accordingly.
Otherwise, the file storage location possibilities are the followings:

- *WebRTC engine with our public WebRTC-SIP proxy service: the webphone can upload the recorded files to public a FTP server or Web service*
- *WebRTC engine with the MRTC gateway: the recording will happen on the MRTC gateway as directed by the webphone*
- *WebRTC engine your own WebRTC service: you should do the call recording on your WebRTC server and store it anywhere you wish*
- *NS or Java engine: these engines are capable to store the recorded files to any FTP server or Web service (including localhost or private servers) and also to local file (by setting the voicerecording parameter accordingly: 0=no, 1=record to local file system, 2=remote http/ftp only, 3=both local and remote)*
- *If you are using a HTTPS API with the NS engine with strict TLS requirement, then it might be possible that the default NS engine might not be able to connect (TLS compatibility issue). Contact our support in this case and we will ship an upgraded NS engine with full/latest TLS capabilities (which are removed from the default to minimize the NS engine size)*

Encryption

The webphone comes with encryption support for both the signaling and the media, including end-to-end encryption.

- For SIP signaling TLS encryption is supported (SIPS) configurable with the transport parameter.
- When using TLS, you might need to change the port in the serveraddress parameter (from the default 5060) to 5061. For example: serveraddress: 'yourdomain.com:5061')
- By default the NS and Java engines includes a minimal TLS implementation. Contact us for the full version if that is not accepted by your SIP server.
- For media (RTP) the webphone has support for SRTP, configurable with the mediaencryption parameter.
- When used as a JS WebRTC library, then by default everything is encrypted: WSS (secure websocket) for the signaling and DTLS/SRTP for the media.
- For user to user calls the webphone is capable for end-to-end encryption even if your server doesn't have any encryption support (only if your server doesn't force media proxy). Configurable with the use_fast_ice setting.
- Optionally we also provide proprietary VoIP tunneling solution. This is useful to bypass any ISP VoIP filtering especially in countries where VoIP is banned.

Auto-provisioning

Auto-provisioning or auto-configuration is a simple way to configure IP-phones for SIP servers used on local LAN.
The exact same behavior can be easily achieved by using the webphone with dynamic parameters.

Actually since the webphone is web hosted, it is auto-provisioned by default (you can just change the settings in the webphone_config.js, or change the settings from Java Script using the setparameter() API or pass them via URI query parameters or you can just deploy separate instances into separate paths).

Obviously MAC address based autoprovisioning will not work for browsers (webpages doesn't have access for the device MAC address).
However you can easily serve different configurations to your users dynamically using any logic after your needs. Since this is a web based solution its configuration can be very easily controlled from a server side script by just changing the webphone parameters dynamically as you wish.

First you should set the parameters common for all instances (all users) on your webserver in the webphone_config.js file.
Then (at runtime or from a server side script) you just have to set account related settings (per user settings) using one of the method specified in the Parameters chapter (by URL, via a server API by scurl_setparameters, or from javascript by the setparameter API).

For example you might auto provision the webphone based on username, browser IP, browser fingerprint or a random ID per client.

File transfer

The prebuilt softphone skin has the file transfer feature already built in. File transfer actually consists of two simple steps:

- upload to a webserver the file selected by the user
- send a chat message to the destination containing the download URL

By default Mizutech's service is used for storage, but this can be customized using the filetransferurl parameter, for example:
filetransferurl: 'http://www.domain.com/filestorage/'
The file name is a unique automatically generated string.

If the webphone is used as SDK and you wish to implement the file transfer feature, it can be done easily. Below is a simple example HTML form, similar to the one used by the webphone. This form consists of a filedata input, a hidden input for the filename and a submit button. The action should be the URL to your file webserver (filetransferurl):

```
<form action="http://www.domain.com/filestorage/" method="post" enctype="multipart/form-data" >
  <input type="hidden" id="filename" name="filename" value="MY_UNIQUE_FILE_NAME">
  <input name="filedata" type="file" id="fileinput" /><br />
  <input type="submit" id="btn_submit" value="Send file" />
</form>
```

Also you have to implement a simple script on server side to store the received files.

Disable popups and toasts

Set the following parameters to disable all kind of popup messages and toasts/hints:
incomingcallpopup: set to 0 to disable
showtoasts: set to false to disable hints
asknotifpermission: set to 0 to disable

transferpopup: set to 0 to disable call transfer popup
displayvideodevice: set to 0 to disable video preview
displaynotification: set to 0 to disable missed call/chat notifications
disablepopup: set to 1 to blindly/unconditionally disable all alert dialogs
disabletoasts: set to 1 to blindly/unconditionally disable all toast/hints texts
enablenomicvoicewarning: set to 0 to disable audio related notifications
useaudiorecord: set to 0 or -1 if you don't need audio recording
useaudiodevicerecord: set to false if you don't use any audio recording device/microphone

If you are using a specific engine, then set its enginepriority value to 5 to avoid asking for other engines regardless of the circumstances.

Incoming call notifications

The notification capabilities about incoming calls depends on the browser and the engine used.
Normally the webphone is capable to accept incoming calls once registered to the SIP server and on incoming calls it is possible to display various notifications and a ringtone for the endusers.
Some browsers/engines can present the incoming calls only if your page is running, however some browser/engine combinations is capable to notify the user even if the page (or even the browser) is closed.
You can adjust the following incoming call alert related settings based on your needs:

- callreceiver: specify to listen for incoming calls even if the browser page is closed. With WebRTC this might use push notifications. With the NS engine it will use background listener service.
- push notifications: may be capable to notify the user about incoming calls even if the browser page is closed while using the WebRTC engine
- incomingcallpopup: set HMTL5 notification for incoming calls
- displaynotification: configuration option for missed call/chat notifications
- sound alert related settings: volumering, ringtimeout, beeponincoming and here

In case if you wish to disable beep emitted on new calls when there is already other call in progress, set the ringincall, beeponincoming and beeponconnect parameters to 0.

See this FAQ point to handle incoming calls from your code.

Always-on availability

It is possible to configure the webphone to always receive calls, even if your webpage is closed.
If you are using the WebRTC engine you can receive incoming call notifications if push notifications are enabled for your project.
For the NS engine, set the following parameters to always run and listen in the background:

callreceiver: 2
backgroundcalls: 1
destroyonpageclose: 0
needunregister: false

With the above settings, the NS engine can also survive page reloads while in call.

VoIP Push notifications

VoIP Push notification is a method to display a browser popup for the user on incoming SIP call or text message even when the webphone is not running.
The user can pick-up the call if clicks/taps on the notification.
This is specific for the WebRTC engine only as with the NS engine you already have a local service which can listen for incoming calls even if the browser is not running.

Note: Currently desktop browsers doesn't support a background listener, thus to receive the notification a browser instance must be already running (but it is not necessary that your page with the webphone to be opened). You might force the NS engine (set the enginepriority_ns to 4) if the enduser must be always available regardless whether the browser is running or not. Browser vendors also plan to implement an always running service for push notifications but it remains to be seen when this feature will be actually released.

You can enable/disable push notifications with the callreceiver parameter or with the enablepush parameter (possible values: 0: disable, 1: enable).
Also the firebase-messaging-sw.js file have to be copied to your web server root directory (If the webphone folder is not your web root, then you must copy this file to webroot).

Push notifications can be handled in the following ways:

- Via the free Mizu PUSH service. If you are using the Mizutech WebRTC-SIP gateway service, then you also have server side push notifications support.
- Directly with your SIP server if your server has support for push notifications. Contact your server vendor or check your server documentations for the details and implement it accordingly for the webphone.
- With the Mizu WebRTC-SIP or voip push gateway. Instead of using our free service, you can setup your dedicated gateway to handle the push notifications. A detailed guide with the implementation details can be found here: VoIP Push Notifications. See the Web and WebPhone chapters specific for this SIP library
- Using the webphone with the Mizu SIP Softswitch or IP-PBX. Push notifications are supported by all Mizu server side products.

In case if you wish to manage push notifications externally (such as from a native Android or iOS app running the webphone in WebView) then you might just send the X-MPUSH extra header (if using our VoIP server or gateway) or set the push parameters in the contact header with the contact_uri_parameters parameter after the RFC 8599 standard.
For example for Android the contact_uri_parameters should be set to 'pn-provider=fcm;pn-param=APPPACKAGENAME;pn-prid=TOKEN'.
More details here.

More details about Mizutech VoIP push notifications tech can be found here and here.

Using the webphone in a WebView

You can embed the webphone into any application using a WebView to be used in any native app for any OS such as Windows, Android or iOS.
Permissions (such as audio recording) are usually required if the webphone will be used in webview! These permissions needs to be handled in your native app and most platforms have permission handling mechanisms.
It is possible to just use the webphone user interface (for example the softphone skin from softphone.html) or you can implement any native GUI running the webphone in a hidden webview and interacting with it via the API.

For example in Android:

- A WebView tutorial can be found here.
- You need to request **RECORD_AUDIO** (and **CAMERA** for video calls) to be able to make calls.
- These are considered “dangerous” permissions and have to be requested runtime, and the user needs to grant access in order to be able to record audio inside a webview.
- It is best practice to request the native android “RECORD_AUDIO” permission before you load the webphone into the webview.
- You can fully interact from your app with the webphone and inverse using your platform specific mechanisms (dynamically generate HTML from your native code, call JavaScript directly from your native code or use callbacks from JS to app)
- Below are a few useful links regarding android permissions:
    - https://developer.android.com/guide/topics/permissions/overview
    - https://developer.android.com/training/permissions/usage-notes
    - https://developer.android.com/training/permissions/requesting

A full android project example can be downloaded from here.
Another simple android example code can be found here.
A basic Electron integration example can be found here.

The same also applies to iOS and other OS: you have to request the permission runtime from the user, and only if the user grants access, will the webphone be able to record audio and/or video.

You might also implement push notifications: get a token in your native app and pass it to the webphone to be sent by SIP signaling to the server. Details here.

Note:
WebRTC is not supported in WebKIT browser on iOS. In iOS native webview (SFSafariViewController) WebRTC is supported starting from version 13.
There are also several tools and frameworks to convert your web app into a native WebView based application such as React Native or PhoneGap.

Modern JS

We take special care to remain compatible with all browsers, including down to IE6, thus we are not using the latest ES5/ES6 features.
However, since the webphone is pure vanilla JavaScript, you can easily create any wrapper around it (for example using an async/await interface instead of the existing callback functions) or convert it into a JS module if in this way it can be more easily consumed in your project.

How to use with Zoho CRM

The webphone is a client-side JavaScript library (with no server-side dependencies) and it has a pure JS API, which means that it can be integrated with any CRM, including Zoho.
You can call any Zoho API from the webphone including Zoho Flow or Zoho Custom Functions. See the "Callcenter integration" FAQ point for more details.
The webphone can be even used into a zoho widget. Contact us if you need a Zoho dialer widget example.

How to use with React?

The webphone is just plain JavaScript which can be deployed as copy-paste and consumed in any framework.
See the package.json file (in the webphone.zip) if you wish to create an npm package from it (the webphone itself doesn't have any dependencies on other JS libraries).

Since the webphone comes as simple/vanilla JS API it can be easily used from any framework, including ReactJS and React Native.
The integration should be simple.
You might use it as vanilla JS, you might create a wrapper around the webphone_api.js, you might convert it into a module or create a react component.
More help can be found here and here.

If you wish to use it as-is as a module, then you might use the exports-loader like this:
import * as webphone_api from 'exports-loader?webphone_api!webphone_api;

A simple react example generated with create react app can be downloaded from here.

How to translate?

You can quickly test your desired language by setting the language parameter.
To add a new language or change/fix existing translations, use this website.
If you don't have the login credentials, ask Mizutech support.
You will be able to add new translations, change any existing strings and generate your new webphone copy with the changes.

By default the webphone might have the following languages included with full or partial translation:

- en: English (default)
- hu: Hungarian
- it: Italian
- pt: Portuguese
- ro: Romanian
- es: Spanish
- de: German
- tr: Turkish
- ja: Japanese
- fr: French
- zh: Chineze

You can set the default language with the language parameter.

Note for Mizutech support: use the MTranslate app to merge the translations.

Angular

Use an iframe to integrate the webphone into your Angular HTML page and set the iframe's "id" attribute to "webphoneframe", for example:

```
<iframe id="webphoneframe" src="http://www.yourdomain.com/webphone_package/softphone.html" width="300" height="500" frameborder="0"
allow="microphone *; camera *; autoplay" allowfullscreen="true" ></iframe>
```

To be able to access the webphone API in the iframe from the parent page you will have to include the `iframe_helper.js` file from the webphone package.
This can be achieved following the below steps:

- Create a directory inside your Angular project `src` directory. We will name the directory `webphone`.
- Copy `iframe_helper.js` from webphone package into `src\webphone` directory.
- Open `angular.json` file and set the path to `iframe_helper.js` under projects -> scripts section:
  `"src/webphone/iframe_helper.js"`
- Open `app.component.ts` and declare the **webphone_api** variable:
  ```
  declare var webphone_api:any;
  ```

Now you are able to fully take advantage of the webphone API.
More details can be found here, here and here.

Contacts management

Although the webphone has built-in support for contacts, you might implement your own separate contact list / address-book module if you need some more related functionalities.
See the listcontacts/addcontact/getcontact/delcontact API's for more details.
These are useful especially if you are using the softphone skin (softphone.html) and you wish to manipulate the contacts from your code.

How to integrate with server side address-book

This description is mostly for non-JavaScript developers of you are going to integrate with a server side address book API with the softphone skin (softphone.html).
Otherwise, you can easily manipulate the contacts from JavaScript with the listcontacts/addcontact/getcontact/delcontact API's.

Use the serveraddressbook_sync parameter to specify how the synchronization have to be done: 0=only once, 1=once, until we receive contacts, 2=every time webphone starts.
Use the serveraddressbook_url parameter to specify the URL or API from where the contacts have to be synchronized. The downloaded data can be a static file or dynamically generated and it must have the following format:

Contacts will be separated by "carriage return new line": \r\n
Contact fields will be separated by "tabs": \t
A contact must have the "name" and "number" fields. Other fields are optional.
The order of fields and their meaning:

name: the name of the contact (usually full name or nickname)
number: username, phone number, extension ID or SIP URI
favorite: 0 means No, 1 means Yes
email: email address of the contact
address: the address of the contact
notes: notes attached to this contact
website: web site attached to this contact
type: any string, but on the softphone skin only the following are handled: phone, home, mobile, work, other, fax_home, fax_work, pager, sip

The number and the type can be a list with strings separated by the | character.

Set the serveraddressbook_clear parameter to 1 if you wish to always clear/replace all contacts instead if merge (default is 0 which means merge).
Set the serveraddressbook_allowedit parameter to 0 if you wish to disable contacts edit/delete/add by the enduser (default is 1 which means edit enabled)
Set the normalize_contact parameter to 0 if you wish to avoid also characters, otherwise (1 by default) the webphone will normalize the strings.

How to add a color theme?

Webphone comes with a few prebuilt skins, which can be changed from Settings -> Theme or preconfigured with the colortheme parameter.
The look and feel of the webphone skin can further be customized by altering any of the predefined themes found in: js\softphone\themes.js.
Open the themes.js file (it is located in webphone/js/softphone folder) with your favorite text editor.
In the "themelist" variable are stored the current webphone themes, you can edit for example the theme_1 after your needs. Please note that the theme_0 (default theme) can't be modified from this file.
From the variables names should be obvious they meaning (bg - means background), the colors are defined in RGB hex.

mainlayout.css: color to replace: #1d1d1d with urlparam: bgcolor
wphone_1.0.css: color to replace: #333 with urlparam: buttoncolor
wphone_1.0.css: color to replace: #373737 with urlparam: buttonhover
wphone_1.0.css: color to replace: #22aadd with urlparam: tabselectedcolor
mainlayout.css: color to replace: #31b6e7 with urlparam: fontctheme
mainlayout.css: color to replace: #ffffff with urlparam: fontcwhite
wphone_1.0.css: color to replace: sans-serif with urlparam: fontfamily

After you modify a variables value, you need to reload your webphone otherwise the modifications will not any effect.
You will also need to set the "colortheme" parameter to match your theme index.

You can create new themes easily by searching for existent dialer skins and after you find one that it is close to your needs just pick the preferred colors using a software like Color Pic or you can search for a color matching tool to help you in building better color schemes.
It is also possible to use the softphone build web service to customize the webphone design ("Design" link the and "Advanced coloring" option using the designer).

How to use the webphone via URL parameters?

The webphone can load its settings also from the webpage URL (URI query strings) and perform various actions such as initiate a call. All the listed parameters can be used, prefixed with "wp_".
Example to trigger a call with the softphone by url parameters:
http://www.yourwebsite.com/webphonedir/softphone.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_callto=CALLEDNUMBER&wp_autoaction=1
Example to trigger a call with the click to call by url parameters:
http://www.yourwebsite.com/webphonedir/click2call.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_callto=CALLEDNUMBER&wp_autoaction=1
Example trigger chat by url parameters:
http://www.yourwebsite.com/webphonedir/softphone.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_sendchat=TEXT&wp_to=DESTINATION&wp_autoaction=2

Note:
You should use clear password only if the account is locked on your server (can't call costly outside numbers). Otherwise you should pass it encrypted or use MD5 instead.
You should encode with encodeURIComponent(). Parameters can be encoded separately or you can encode the whole query string after the ? character.

See also click to call.

How to implement Click to call?

All you need to implement click-to-call for your webpage is this webphone and a SIP account (at any VoIP service provider or your own SIP server)

Just set your account settings for the webphone and you are ready to go:
-your voip service provider server address ("serveraddress" webphone parameter)
-sip username ("username" webphone parameter),
-sip password ("password" webphone parameter)
-optionally preconfigure the number to call ("callto" webphone parameter)

You can set these statically from the webphone_config.js file or via javascript using the setparameter and call API. (Mizutech can also hardcode these in your final licensed build if you wish.)
Then you can use the "click2call.html" from the samples folder (or use the softphone.html if you need a full UI or the call API from your custom button).

Note: if you hardcode the above parameters in the webphone_config.js, then it is a good idea to encrypt them or make sure that the account is restricted after your needs.

For more details check the click to call section.

Click to call from email signature

Just set your phone number in your email signature as a link (URL anchor) to the webphone click to dial:
http://www.yourwebsite.com/webphonedir/click2call.html?wp_serveraddress=YOURSIPDOMAIN&wp_username=USERNAME&wp_password=PASSWORD&wp_callto=YOURNUMBER
In this way the phone number in your email signature will become a clickable link which will trigger the webphone and will call your number automatically on SIP.
Instead of the click2call.html, you can also use the softhone.html (or your custom webphone html).

For account username/password you should just create a special extension on your SIP server which is not authenticated and allows unrestricted calls to local extensions only (not to outbound/paid).

More details about click to call can be found here.

Push to talk

PTT can be implemented by using the hold function.
If call hold is not well supported by your server or the remote peer, then you might use mute instead.

Other related functions and parameters which you might use are the followings:
ismuted, isonhold, automute, autohold, holdtype, muteonhold, defmute, sendrtponmuted

Clickable phone numbers on webpages

The webphone can be used to automatically recognize telephone numbers on a web page, convert them to clickable links and if the users click on the link it will dial the number.

This functionality is implemented by the linkify.js (in the /js/linkify folder).
You can find a working example for this in the samples folder linkify_example.html.

In case if you wish to implement it all-over your website, then you might insert the linkify script lines to all html files from your web server configuration (Instead of manually modifying all your html files. Most web servers has this capability) or if you are using some CRM, then that might have a functionality to insert a JS header in each file.

Floating webphone

To float the webphone skin over your web page, just set the following CSS attributes for the container HTML element of the webphone (which can be a DIV or an iframe):

```
// this aligns the webphone to the bottom-right corner of you page
z-index: 1000; position: fixed; bottom: 0px; right: 0px;
```

If you wanted for instance to set it in the top-left corner, then the CSS attributes would be:

```
z-index: 1000; position: fixed; top: 0px; left: 0px;
```

How to pass PHP session variables?

You can use this format:

```
var jsvariable='<?php echo $php_session_value;?>';
webphone_api.setparameter('parametername',  jsvariable);
```

Just make sure that the page is actually generated by php and not statically served (so the php scripts runs on the page).
You can easily verify by inspecting the page source once the page is loaded into the browsers.

How to manage multiple lines?

Multi-line means the capability to handle more than one call at the same time (multiple channels).

By default you don't need to do anything to have multi-line functionality as this is managed automatically with each new call on the first “free” line.
If you have multiple ongoing calls, then the active call will be the last one you make or pickup.

Multi-line vs Conference

When we refer to “multi-line” we mean the capability to have multiple calls in progress at the same time. This doesn’t necessarily means conference calls. You can initiate multiple simultaneous calls by just using the call API multiple times (to initiate calls to more than one user/phone), so you can talk with remote peers independently (all pears will hear you unless you use hold on some lines, but the peers will not hear each-others). You can also have multiple incoming calls at the same time in any state (ringing/connected).
To turn multiple calls into a conference, you need to use the conference API (or use the conference button from the softphone.html). When you have multiple peers in a conference, all peers can hear each-other.

User interface:

Multi line functionality is enabled by default in the webphone.
Once the enduser initiate or receive a second call, the webphone will automatically switch to multi-line mode.
If you are using the softphone skin (the softphone.html) its user interface will display the separate calls in separate tabs, so the user can easily switch between the active calls.

Actually the followings user interface elements are related to multi line:

- on the Call page, once you have a call, you can initiate more calls from Menu -> New call
- for every call session, a line button will appear at the top of the page so the users can change the active line from there
- the line buttons for managing call sessions, will also appear in case another incoming call arrives
- you can easily transfer the call from line A to line B
- you can easily interconnect the active lines (create conference calls)

Disable multi-line

You can disable multi-line functionality with the following settings:
-set the “multilinegui” webphone parameter to 0
-set the "rejectonbusy" setting to "true"

Other related parameters are the "automute" and "autohold" settings.

JavaScript library/API

When the webphone is used as an SDK, the lines can be explicitly managed by calling the setline/getline API functions:

```
- webphone_api.setline(line); // Will set the current line. Just set it before other API calls and the next API calls will be applied for the selected line
- webphone_api.getline();  //Will return the current active line number. This should be the line which you have set previously except after incoming and outgoing calls (the webphone will automatically switch the active line to a new free line for these if the current active line is already occupied by a call)
```

For example if there are multiple calls in progress and you wish to hangup one of the calls, then just call the webphone_api.setline(X) before to call webphone_api.hangup().
The active line is also switched automatically on new outgoing or incoming calls (to the line where the new call is handled).

Channels

The following line numbers are defined:

- -2: all (some API calls can be applied to all lines. For example calling hangup(-2) will disconnect all current calls)
- -1: current line (means the currently selected line or otherwise the "best" line to be used for the respective API)
- 0: undefined (this should not be received/sent for endpoints in call, but might be used for other endpoints such as register endpoints)
- 1: first channel
- 2: second channel
- …
- N: channel number X

Some behaviors will automatically change when you have multiple simultaneous calls. For example the conference API/button will automatically interconnect the existing parties or the transfer API/button will transfer the call from the current line to the other line.
Note: If you use the setline() with -2 and -1, it will be remembered only for a short time; after that the getline() will report the real active line or "best" line.

API usage example:

```
webphone_api.call('1111');  //make a call
webphone_api.call('2222');  //make second call

//setup conference call between all lines
webphone_api.setline(-2);  //select all lines
webphone_api.conference(); //interconnect current lines

//disconnect the second call
webphone_api.setline('2222');
webphone_api.hangup(true);

//put first call on hold
webphone_api.setline('1111');
webphone_api.hold(true);
```

Notes

- Alternatively, you might just run separate webphone instances to implement multiple lines as described here.
- Some API might auto-guess the correct line to use if you supply a wrong line. For example if you call hangup() on a line which doesn't have an active call then the webphone might disconnect the call on another line if any. (Auto-guess best match line)
- You can use the linetocallid and the callidtoline functions if you wish to convert between line number and SIP Call-ID.
- If your use-case requires multiple simultaneous calls, then you might wish to set the following parameters:
  usecommdevice: 0
  aec: 0
  aec2: 0
  agc: 0
- There are two related parameters to handle some workarounds and for maximum compatibility across browsers. You might change these if you don't like their defaults:
  - multilineoop
    (number)
    Specify hold-mute workaround for the WebRTC engine with multiple simultaneous calls.
    With the WebRTC engine the webphone might use mute instead of hold with multiple lines as a workaround for some multi-line hold related WebRTC issues in some browsers.
    Possible values:
    -1: always use mute instead of hold (even if there are no multiple calls)
    0: use mute instead of hold with multiple calls
    1: usually use mute with multiple calls
    2: use mute instead only in some cases for multi-line calls
    3: never use mute instead of hold
    Default is: 2
  - removemultilinestream
    (boolean)
    You might keep the media stream after hangup if there are other pending calls. Applicable only for the WebRTC engine.
    Possible values:
    false: means that the media stream can be kept (for maximum compatibility)
    true: the media stream will be always closed
    Default is: false
    When set to false, the browser might still display the microphone icon (as not released) after the hangup of the calls.

How to manage multiple accounts?

Multi-accounts means the capability to handle more than one SIP account at a time.
You can use different SIP credentials on the same server (extensions) or to different servers.

To enable multiple accounts, you can use the extraregisteraccounts parameter (if you are using fix accounts which can be set statically in the webphone_config.js or passed as URL parameter) or at run-time with the registerex API.

When you are using multiple accounts, the SIP engine will register with each account so it is capable to also receive calls via any of the accounts. For outgoing calls or chat the "main account" is used by default (which is specified by the serveraddress/username/password/etc parameters).

If you are using the softphone skin (softphone.html) then multiple accounts can be added from Settings -> Advanced Settings -> SIP Settings -> Accounts.

Another way to be registered with multiple accounts is to launch multiple webphone instances in different webpages (or in different iFrames if you wish to display them on the same page).
You can also use different profiles to completely separate the settings storage.

How can I set the engine to be used?

The best engine is selected by the webphone automatically based on circumstances (client device, OS, browser, network, server):
However the preferred engine can influenced on 3 levels:
-Choice presented to the user in some circumstances on startup (This is not always presented. The webphone will go with the best engine when there is a definitive winner, without asking the user)
-Engine settings in the user interface, so the enduser might change its own preferred engine
-Engine priority options in the configuration. You can set this in the "webphone_config.js" (enginepriority_xxx settings as discussed in this documentation Parameters section)

There should be very rare circumstances when the default engine selection algorithm should be changed. The web sip lib always tries to select the engine which will disturb the user the less (minimizing required user actions) and offers the best performance.
For example don't be inclined to disable Java for the sake of its age. Users will not be alerted to install Java by default. However if Java is already enabled in the user browser then why not to use it? Java can offer native like VoIP capabilities and there should be no reason to disable it.
We spent a considerable amount of work to always select the best possible engine in all circumstances. Don't change this unadvisedly, except if you have a good reason to use a particular engine in a controlled environment.

NS engine troubleshooting

Contrary to the WebRTC engine (where users doesn't have to download or install anything), the NS requires an one-click download/install step by the endusers.
The NS engine is a native executable running the SIP/media stack as an NT service on Windows or as a daemon on Linux, exposing its API via a local websocket (the webphone browser user interface connects to the NS engine by websocket to localhost/127.0.0.1/ns4.webvoipphone.com).
When using the NS engine, the webphone will automatically offer the relevant file (depending on the user OS) to be downloaded and installed.
Check the followings if the NS engine can't install / start or the webphone is unable to connect to the NS engine:

Windows:

1. Check if the NS engine is installed and running as an NT service
   Open the services (type "Services" or run services.msc)
   Check the "Webphone" or "Your_brand_name" entry.
2. If not installed, run the WebPhoneService_Install.exe from the /webphone/native/ folder as Administrator
3. If you have a strict firewall which might block also local connections, then add then white-list the NS engine executable (add it as an exception).
4. By default the service executable can be found at :
   C:\Program Files (x86)\WebPhone_Service\WebPhone_Service.exe
5. If still fails, then send the logs to Mizutech support (the whole content of the above folder and the browser console output as described also here)

Linux:

1. Extract the corresponding webphone_ns_lin*.zip file (usually webphone_ns_lin64.zip) from the /webphone/native/ folder
2. Launch the webphone_ns_lin*.run file with root user privileges (sudo webphone_ns_lin64.run).
3. The installer script will also auto detect local JRE if any, and should install a JRE if not found.
   Please verify if you have Java installed (a JRE or JDK) by typing java –version in your console.
   If the command fails, please install a JRE and the launch the webphone_ns_lin*.run file again.
   On linux use your package manager to install Java. For example on Ubuntu use the following command:
   sudo apt install default-jdk
4. Check if the NS engine is running:
   systemctl list-units | grep webphone (should find webphone_ns_lin64.bin.service)
   or: systemctl status webphone_ns_lin64
5. Now the webphone should be able to connect to the NS engine.
6. If it works now, then please the possible problem to Mizutech to be fixed
7. If still fails, then please send the webphone_ns_lin*.run output and all the related logs:
   - The NS engine logs can be found usually at: /opt/WebPhone_NS/mwphonedata (rarely it might be located at /home/username/mwphonedata)

- Install related logs can be found at: /opt/WebPhone_NS/installog.dat (useful if install fails, NS service can't start or the webphone can't connect to it)
- Or send the whole /opt/WebPhone_NS/ folder content of possible (zip it first)
- The webphone logs can be loaded from the browser console
- More details here

How to set the webphone parameters dynamically?

The easiest way to specify parameters for the webphone is to just enter them in the webphone_config.js file.
However if you need to integrate the webphone with your server (for example with a CRM) you might have to set different parameters regarding the session (for example different user credentials based on the currently logged-in user). There are 3 ways to do this:

1. With the client side JavaScript using the webphone setparameter API (get the parameters from you webapp or via ajax requests)
2. Just generate the URL (iframe or link) dynamically from your server side scripts with the parameters set as required (wp_username, wp_password and other URL parameters).
3. Set the "scurl_setparameters" setting to point to your server side http api which will have to return the details once called by the webphone.
   This will be called before onAppStateChange "started" event and can be used to provision the webphone from server API. The answer should contain parameters as key/value pairs, ex: username=xxx,password=yyy.

See the beginning of the parameters section for all other possibilities.

Abbreviations

Common terms used in this document:
WebPhone: JavaScript SIP library or ready to use softphone for browsers
Caller: The person making/initiating a call
Callee: The person receiving a call (called party)
SIP: The Session Initiation Protocol (SIP) is a signaling protocol used for establishing sessions in an IP network
NS engine: native VoIP client service to be installed on client PC to be used by the webphone from browsers similarly to a browser plugin
WebRTC: Web Real-Time Communication used implemented in HTML5 browsers and other clients
RTP: media channel protocol
DTLS: UDP security protocol to encrypt the RTP packets
SRTP: encrypted RTP
Websocket: TCP like connection for browsers
WS: websocket
WSS: secure websocket
TLS: transport layer security (secure TCP using a certificate for your server domain)
HTTPS: secure HTTP provided by TLS
SIP server: VoIP server, softswitch or IP PBX capable to understand the SIP protocol
WebRTC server: VoIP server, softswitch or IP PBX capable to understand the WebRTC protocol
WebRTC gateway: a software or service which can convert from WebRTC to SIP and inverse, usually implemented as a proxy, gateway or SBC
JS WebRTC SIP library / WebRTC engine: the webphone acting as a WebRTC client (websocket for the SIP signaling, DTLS/SRTP for the media)
Native NS or Java engine: the webphone acting as a native SIP client (UDP, TCP or TLS for the SIP signaling, plain RTP or SRTP for the media)
SIP proxy: SIP protocol relay at the network edge implemented as proxy, gateway or SBC
ASR: average success ratio (percent of the connected calls)
ACD: average call duration. The same as ACL
ACL or ACD: average call length. (ACD is sometime also used for Automatic Call Distributor)
TCP: is a connection-oriented internet protocol
UDP: internet core protocol for datagram packets (not reliable)
REGISTRAR: server-side component that allows SIP REGISTER requests (registrar server)
ANI / CLI: Automatic Number Identification or Caller Line Identification
IVR: Interactive Voice Recognition
SMS: short messaging service usually provided by mobile carriers (the closest term in SIP is chat SIP messages)

Errors in the browser console

There are a few common operations done by the webphone which might trigger an error or warning line in the browser console which can be safely ignored if otherwise your webphone works correctly.
The most frequent errors are the followings:

ERROR and WARNING messages

If you set the loglevel higher than 1 than you will receive messages that are useful only for debug.

Most of ERROR and WARNING message cannot be considered as faults in this case.

Some of them will appear also under normal circumstances and you should not take special attention for these messages.

If there are any issue affecting the normal usage, please send the detailed logs to Mizutech support (webphone@mizu-voip.com) in a text file attachment.

nsX.webvoipphone.com connectivity errors

The nsX.webvoipphone.com points to localhost (127.0.0.1) and it is used by the webphone to communicate with the local NS engine if any.

In the logs you might notice errors like:

POST https://ns4.webvoipphone.com:10433 net::ERR_CONNECTION_REFUSED.

or

WebSocket connection to 'wss://ns4.webvoipphone.com:10443/msstwebsock' failed: Error in connection establishment: net::ERR_CONNECTION_REFUSED

This is a normal error condition if you are not using the NS engine as at startup the webphone might try to detect the presence of the local NS engine by trying to connect to this URL.

You might also see CORS issues like this:

Cross-Origin Request Blocked: The Same Origin Policy disallows reading the remote resource at https://127.0.0.1:18520/extcmd_test.

(Reason: CORS request did not succeed)

These can be also safely ignored as the webphone will try to connect to the local NS engine by domain name or via websocket.

Uncaught TypeError: webphone_api.xxx is not a function errors

This usually happens if you try to access the webphone API before the webphone was fully loaded.

Make sure to call any API only after the onAppStateChange callback "loaded" event was triggered.

"empty-block" statements related warnings in JSLint

Our build stacks automatically removes some unneeded code at built time. These warnings can be safely ignored.

AbortError: The fetching process for the media resource was aborted by the user agent at the user's request.

Firefox sometimes throws an AbortError when an audio/video stream is attached to an <audio>/<video> HTML element.
This is an internal Promise failing in Firefox and is logged in browser console.
This error can be ignored and it doesn't affect any functionality.

Deprecation warnings

Our goal is to be compatible with all possible browsers, including old versions.
For this reason we are using also deprecated tag which might trigger a browser warning but otherwise are omitted.
Usually all these warnings can be safely ignored and they doesn't affect the webphone functionality.

RTP warning

The webphone will send a few (maximum 10) short UDP packets (\r\n) to open the media path (also the NAT if any).

For this reason you might see the following or similar Asterisk log entries:

"WARNING[8860]: res_rtp_asterisk.c:2019 ast_rtp_read: RTP Read too short" or "Unknown RTP Version 1".

or "codec_opus.c:503 opus_samples_count: Opus: Unable to parse packet for number of samples: corrupted stream"

These packets are simply dropped by Asterisk which is the expected behavior. This is not a webphone or Asterisk error and will not have any negative impact for the calls.

You can safely skip this.

You might turn this off by the "natopenpackets" parameter (set to 0). You might also set the "keepaliveival" to 0 and modify the "keepaliveival" (all these might have an impact on the webphone NAT traversal capability).

All of the above can be usually ignored. If you see other errors and the webphone doesn't work as expected, please contact our support by sending the browser console output as email text file attachment. Make sure that the console output was generated with the webphone loglevel parameter set to 5.

How to get the logs?

In short:
Make sure that the webphone loglevel parameter is set to 5 in the webphone_config.js file. Then reproduce the problem and sent the full content of your browser console by email to webphone@mizu-voip.com as a text file attachment with an accurate problem description in English (including What you did? What happened? What was expected to happen?).

Details:
The webphone can generate detailed logs for debugging purposes.
For this just set the "loglevel" setting to 5 (or enable logs from the user interface if any).
Note: by default the loglevel is already set to 5 (on) in demo and trial versions and we usually send out licensed builds with loglevel 1 (off) by default.
Once enabled, you can see the logs in the browser console or in the softphone skin help menu (if you are using this GUI). If the Java engine is being used, then the logs will appear also in the Java console. You might also use the API: getlogs() or the onEvent(callback) with "log" type events.
When contacting Mizutech support with any issue, please always attach the detailed logs: the full output of the browser console (or you can find the same from the softphone skin help menu if you are using the softphone.html).
On Firefox and Chrome you can access the logs with the Ctrl+Shift+J shortcut (or Cmd+Shift+J on a Mac). On Edge and Internet Explorer the shortcut key is F12.
In some browsers this is found under a "Developer Tools" or similar menu, "Console" tab. See this page for more help if you still can't find the browser console.
Make sure to include the logs from the very beginning (select and copy all text) as we need to check your environment and settings which is written at startup.
If there are multiple calls in the logs but not all of them has the issue you described, then please specify the SIP Call-ID of the problematic call or much better if you send a log with only a single call to avoid confusions.

Avoid sending recordings or screenshots. We always need detailed logs only and a detailed and exact issue description.
When sending logs to Mizutech support, please attach them as files (don't insert in email body).
In some circumstances you might have to send logs from the specific webphone engine as described below.

HTML/JavaScript logs

You can get this from the browser console as described above.

iOS/Safari browser console

In case if you need to access the Safari browser console from an iPhone, follow this guide.
Here is an iOS specific wiki.

Browser console log size limitation

Firefox by default trims the console logs to 200 lines.
To increase this limit type about:config into the address bar and search for "loglimit" and increase every value to 5000.
Chrome also limits the maximum lines. Set the "Preserve log" option to prevent it (from the browser console top-right gear icon)

WebRTC engine detailed logs

If the webphone is using the WebRTC engine then the browser console output will contain the most important logs.
If you are using the softphone skin, then better if you check the logs from the skin help menu because the number of lines are limited in the browser console.
If you have voice issues (no voce, one side voice, delays) then you should get a detailed log:
With Firefox this can be done by navigating to about:webrtc.
With Chrome this can be done by navigating to chrome://webrtc-internals
For more internal details, you might enable WebRTC tracing:
With Chrome this can be done by launching it like:
*"C:\Program Files (x86)\Google\Chrome\Application\chrome.exe" --enable-logging --v=4 --vmodule=*libjingle/source/talk/*=4 --vmodule=*media/audio/*=4*
Then you can find the logs at: C:\Users\USER\AppData\Local\Google\Chrome\User Data\chrome_debug.log
(replace USER with your windows username or rewrite the path to match your user directory)

Java engine detailed logs

If the webphone is using the Java engine, then a log window will appear if the "loglevel" is set to "5" and the "canopenlogview" to "true".
Grab the logs also from this log window (Ctrl+A, Ctrl+C, Ctrl+V) or from the Java console.

NS engine detailed logs on Windows

If the webphone library is using the NS engine on Windows, then some more detailed logs can be obtained from:
C:\Program Files (x86)\WebPhone_Service\WebPhone_Servicelog.dat
and C:\Program Files (x86)\WebPhone_Service\content\native\mwphonedata\webphonelog.dat.

The folder name might contain your brandname instead of "WebPhone" such as YourBrand_Service. In this case the
C:\Program Files (x86)\YourBrandName_Service will be the default data directory or it might be different on your PC.
It might be located in the C:\Users\USER\AppData\Roaming\WebPhone_Service directory if the account doesn't have write access to Program Files).
*If there is no *log.dat file, just send the "wphoneout.dat" or*
*send all the *.dat files from the NS folder and it's subfolders if you are not sure (from both the app directory and from /content/native/mwphonedata folder).*

*NS engine install related logs (relevant only if install fails) can be found at: C:\Users\USER\AppData\Local\Temp\i4j_nlog_ **

Ideally you should stop the webphone service before to grab the logs (from cmd command line: net stop Webphone).
You can restart it after you copied the logs with the "net start Webphone" command.
The NS engine log level can be also set separately with the nsloglevel parameter.

NS engine Linux logs

The NS engine logs can be found usually at: /opt/WebPhone_NS/mwphonedata (rarely it might be located at /home/username/mwphonedata)
Install related logs can be found at: /opt/WebPhone_NS/installog.dat (useful if install fails, NS service can't start or the webphone can't connect to it)

NS engine MacOS logs

The (now deprecated) NS engine logs can be found at: /home/username/mwphonedata or at: /Applications/WebPhone_NS/mwphonedata

Server side logs

If the problem is triggered by your SIP server, you should look after the reason in the server log first. Make sure to increase your server log/debug/trace level to maximum to also include the SIP signaling.

Mobile browsers

If you are used with a mobile (Android/iOS) browser for testing, then you should try to replicate the same from a desktop browser for easy access to the browser console content. Otherwise you might have to connect to your phone to get the logs as described for example here.

Working with logs

Logs are generated to help you in troubleshooting.
To understand the log files, you need some basic SIP protocol understanding. Here are some basic message flow examples.
In case if the softphone doesn't connect or register, look for WebSocket (ws/wss) connectivity issues or register issues (search for "REGISTER sip").
In case if there is some call related problem, find the call in the log (search for "INVITE sip") and then go trough the call by searching after the Call-ID field what you found in the first INVITE.
If your server disconnected/rejected the call or the registration, then check your server log about the same call or registration.
If the problem is not obvious, send the relevant logs to mizutech support.

How to find which engine was used?

To find all engine related log, like which engines are supported, selected/recommended engine, just search for "engine".
Also, before every engine start, all the engine priorities are logged, search for: "enginepriority"
To find out which engine was actually started, search for: "start engine:" in the browser console output.

If WebRTC engine is selected, how to find the websocket URL, sip server and ice settings?

Search for: "Webrtc connection details:". There you will find all the above details.

"Send log to support" functionality

The softphone skin (softphone.html) has a "Send log to support" option in its main menu where users can comfortably upload log if they encounter any issue.
This is a simple HTML form, which sends a POST request to a specified URL. By default this is sent to Mizutech support.
You can completely remove this functionality or rewrite the target URL to yours:
-"logform_action" (String) the action URL of the form where the POST request is sent
-"logform_filename" (String) this will set the "filename" hidden input parameter of the form. This filename can be used to save the log file on server side.
Note: you need to write a small server side scripts to handle the POST and save it to file for this functionality to work.

Sending logs to MizuTech technical support:

- *Always generate the logs with the loglevel parameter set to 5 (set this in the webphone_config.js file)*
- *Send the full log, which includes both the webphone startup and the problematic session. If the problem is call related, then make sure that the log contains also the disconnect of your call.*
- *Make your session as short as possible reproducing the problem (for example only one call if possible).*
- *If your log contains multiple calls then make it clear which one is the problematic one (send it's SIP Call-ID and/or make sure that it is the very last call in your log)*
- *Send the log as email attached file (not as screenshot or copy-pasted to email body)*

Resources

Homepage: https://www.mizu-voip.com/Software/WebPhone.aspx
Download: https://www.mizu-voip.com/Portals/0/Files/webphone.zip
Pricing: https://www.mizu-voip.com/Support/Webphonepricing.aspx
Contact: webphone@mizu-voip.com

Copyright © 2008-2026 Mizutech SRL