JavaScript XMPP Client Documentation

Release 4.0.0-alpha

JSXC Team and Contributors

Aug 25, 2021

Contents:

JSXC is a free XMPP client licensed under the MIT license.

This version of the documentation covering JSXC 4.0.0-alpha has been rendered at: Aug 25, 2021

CHAPTER 1

Getting Started

1.1 Requirements

Obviously you need a running web server and a XMPP server with BOSH support.

1.1.1 Web server

Apache

If not already done, install apache on your Debian-based distribution:

```
sudo apt install apache2
```

Enable the apache proxy module:

```
sudo a2enmod proxy
sudo a2enmod proxy_http
```

Add the proxy definition to your vhost configuration of your host-application server (e.g. `/etc/apache/sites-available/default`):

```
ProxyPass /http-bind/ http://localhost:5280/http-bind/
ProxyPassReverse /http-bind/ http://localhost:5280/http-bind/
```

Reload apache configuration:

```
sudo service apache2 reload
```

Finished!

Lighttpd

Enable the `mod_proxy` module and add the proxy definition in `/etc/lighttpd/lighttpd.conf`:

```
server.modules += ( "mod_proxy" )
proxy.server = (
    "/http-bind" => (
        ( "host" => "127.0.0.1", "port" => 5280 )
    )
)
```

Here is an alternative example of a vhost config.

In the `lighttpd.conf` enable the *mod_proxy* module:

```
server.modules = (
    ...
    "mod_proxy"
    ...
)
...
# include all files that are under vhosts
include_shell "cat /etc/lighttpd/vhosts/*.conf"
```

and in `vhosts/cloud.myserver.com.conf`:

```
$HTTP["host"] == "cloud.myserver.com" {
        ...
        proxy.server = (
                "/http-bind" => (
                        ( "host" => "127.0.0.1", "port" => 5280 )
                )
        )
}
```

Nginx

Enable the proxy pass in your Nginx vhost config:

```
server {
        ...
        location /http-bind {
                proxy_pass http://127.0.0.1:5280;
                proxy_set_header Host $host;
                tcp_nodelay on;

        }
}
```

1.1.2 XMPP server

Prosody

If you encounter problems with BOSH, please add these lines to `prosody.cfg.lua`:

```
consider_bosh_secure = true
cross_domain_bosh = true
```

Ejabberd

1.2 Installation

1.2.1 JSXC for Nextcloud

Warning: JSXC assumes that you are using the same credentials for your Nextcloud and XMPP server.

Requirements

TODO

Get it

Go to your app store and enable the **JavaScript XMPP Client**.

Configure it

Go to the Nextcloud admin page:

BOSH URL The URL to your bosh server (e.g. `/http-bind/`). Please be aware of the same-origin-policy. If your XMPP server doesn't reside on the same host as your OwnCloud, use the Apache ProxyRequest as described in our prepare Apache guide.

XMPP domain The domain of your Jabber ID.

XMPP resource The resource of your JID. If you leaf this field blank a random resource is generated.

TURN url The url to your TURN server. You get a free account on http://numb.viagenie.ca

TURN username If no username is set, the TURN REST API is used.

TURN credential If no credential is set, the TURN REST API is used.</dd>

TURN secret Secret for TURN REST API.

TURN ttl Lifetime of credentials.

Internal JSXC XMPP server

OJSXC implements a minimal XMPP server, just enough such that JSXC works. It is meant as a starting point, as long as you only run JSXC on Nextcloud. As soon as you require more features (external clients, server-to-server communications, . . .) you should install a full-fledged XMPP server (they are pretty easy to install).

1.2.2 JSXC for WordPress

In addition to the WordPress installation you need a running XMPP server with BOSH support. Next you have to configure your XMPP server, maybe activate your BOSH module and to make sure, that your BOSH Server is reachable by your website. If your XMPP server supports CORS everything should be fine.

1. Install web server, xmpp server, (bosh server)
2. Download and extract [jsxc](https://github.com/jsxc/jsxc/releases) to a folder of your choice inside of your WordPress theme folder (e.g. /data/mydomain.com/webroot/wp-content/themes/mytheme/jsxc).
3. Create a `jsxc_client.js` file in your scripts directory (e.g. /data/mydomain.com/webroot/wp-content/themes/mytheme/js) and insert the following code into it:

```
$(function($) {
jsxc.init({
    loginForm: {
        form: '#page-login-form',
        jid: '#page-user-login',
        pass: '#page-user-pass'
    },
    logoutElement: $('#logout-element'),
    root: '../jsxc/',
    displayRosterMinimized: function() {
        return true;
    },
    xmpp: {
        url: 'http://YOUR_XMPP_SERVER.com:7070/http-bind/',
    domain: 'YOUR_DOMAIN',
        resource: 'jsxc'
    }
});
});
```

   Replace `YOUR_DOMAIN` with the domain of you XMPP server and `http://YOUR_XMPP_SERVER.com:7070/http-bind/` with a correct URL of your BOSH server. Also replace `#page-login-form`, `#page-user-login`, `#page-user-pass` and `#logout-element` with correct id's of your site.

4. Create a `jsxc-custom.css` file inside of the styles directory of your site (e.g. `/data/mydomain.com/webroot/wp-content/themes/mytheme/css/`).
5. Insert the following lines of code into some shared page (e.g. `header.php`):

```
<link href="<?php echo get_template_directory_uri(); ?>/jsxc/css/jquery-ui.min.css
↪" media="all" rel="stylesheet" type="text/css" />
<link href="<?php echo get_template_directory_uri(); ?>/jsxc/css/jsxc.css" media=
↪"all" rel="stylesheet" type="text/css" />
<link href="<?php echo get_template_directory_uri(); ?>/jsxc/css/jsxc.webrtc.css"␣
↪media="all" rel="stylesheet" type="text/css" />
<link href="<?php echo get_template_directory_uri(); ?>/css/jsxc-custom.css"␣
↪media="all" rel="stylesheet" type="text/css" />
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jquery.min.js">
↪</script>
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jquery.ui.min.js
↪"></script>
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jquery.colorbox-
↪min.js"></script>
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jquery.
↪slimscroll.js"></script>
```

(continues on next page)

(continued from previous page)

```
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jquery.
↪fullscreen.js"></script>
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/lib/jsxc.dep.js"></
↪script>
<script src="<?php echo get_template_directory_uri(); ?>/jsxc/jsxc.min.js"></
↪script>
<script src="<?php echo get_template_directory_uri(); ?>/js/jsxc_client.js"></
↪script>
```

Verify that jquery is not loaded twice, otherwise the client will not work correctly. 6. Open the main page of your site and notice the jsxc is working. 7. Investigate the CSS styles of jsxc and overwrite the required styles in `jsxc-custom.css` to match your site's UI.

1.2.3 JSXC for Ilias

Warning: The Ilias version of JSXC is currently unmaintained.

Get the code

Packed version

Download the latest version from *releases <https://github.com/jsxc/jsxc.ilias/releases>* and extract it to `ILIAS_DIR/Customizing/global/plugins/Services/UIComponent/UserInterfaceHook/`.

Development version

cd ILIAS_DIR/Customizing/global/plugins/Services/UIComponent/UserInterfaceHook/ git clone https://github.com/sualko/jsxc.ilias ijsxc cd ijsxc git submodule update –init –recursive

Configuration

Rename `config.inc.php.sample` to `config.inc.php` and adjust the values for XMPP server, BOSH url and XMPP domain and the values for WebRTC.

1.2.4 JSXC for SOGo

Warning: The SOGo version of JSXC is currently unmaintained.

Get the code

Packed version

Download the latest version from releases and extract it to `/usr/lib/GNUstep/SOGo/WebServerResources/`.

Development version

```
cd /opt
git clone https://github.com/jsxc/jsxc.sogo sjsxc
cd sjsxc
git submodule update --init --recursive
ln -s /opt/sjsxc /usr/lib/GNUstep/SOGo/WebServerResources/
```

JSXC Configuration

It is good to first make sure you can connect and established XMPP communications before configuring the WebRTC part (which should anyway work most of the time right off with default configuration).

Rename `sjsxc/js/sjsxc.config.sample.js` to `sjsxc/js/sjsxc.config.js` and adjust the values for xmpp server, bosh url and xmpp domain.

Example

Here below is an example of a working config file with few options although sufficient to get JSXC plugin to connect to your XMPP server.

File `sjsxc.config.js`:

```
/**
* feel free to browse through ./jsxc/src/jsxc.options.js to see possible␣
↪configurations options
**/

var sjsxc = {};
sjsxc.config = {
    /** enable chat by default? */
    enable: true,

    /** JSXC options. */
    jsxc: {
        xmpp: {
            /** url to bosh server binding. Adapt port to the one you declared in␣
↪your XMPP server's config */
            url: 'https://xmpp.example.com:7443/http-bind/',

        /** domain part of your jid */
            domain: 'example.com',

            /** which resource should be used? Blank, means random. */
            resource: '',

            /** Allow user to overwrite xmpp settings? */
            overwrite: true,

            /** Should chat start on login? */
            onlogin: true
        }
    }
};
```

Note: it might be a good idea to use a reverse proxy for reaching the BOSH server (*url* parameter then becoming *https://xmpp.example.com/http-bind/*, otherwise tight-up networks would prevent the user to connect to XMPP's port.

SOGo configuration

Add `SOGoUIAdditionalJSFiles` to your sogo config (`/etc/sogo/sogo.conf`).

Packed version

```
{
    [...]

    SOGoUIAdditionalJSFiles = (
        "sjsxc/js/lib/jquery.min.js", // only SOGo v3
        "sjsxc/js/lib/jquery.ui.min.js",
        "sjsxc/js/jsxc/lib/jquery.slimscroll.js",
        "sjsxc/js/jsxc/lib/jquery.fullscreen.js",
        "sjsxc/js/jsxc/lib/jsxc.dep.min.js",
        "sjsxc/js/jsxc/jsxc.min.js",
        "sjsxc/js/sjsxc.config.js",
        "sjsxc/js/sjsxc.js"
    );

    [...]
}
```

Development version

```
{
    [...]

    SOGoUIAdditionalJSFiles = (
        "sjsxc/js/lib/jquery.min.js", // only SOGo v3
        "sjsxc/js/lib/jquery.ui.min.js",
        "sjsxc/js/jsxc/dev/lib/jquery.slimscroll.js",
        "sjsxc/js/jsxc/dev/lib/jquery.fullscreen.js",
        "sjsxc/js/jsxc/dev/lib/jsxc.dep.js",
        "sjsxc/js/jsxc/dev/jsxc.js",
        "sjsxc/js/sjsxc.config.js",
        "sjsxc/js/sjsxc.js"
    );

    [...]
}
```

Restart sogo service

```
sudo service sogo restart
```

Debug

First off, make sure your BOSH server is accessible, and that the certificate is valid for HTTPS. One way to do that is prepare a generic test file (_testBosh.txt_) that you'll call via **curl** command :

Content of the file `testBosh.txt`:

```
<body content='text/xml; charset=utf-8'
    from='user@localhost'
    hold='1'
    rid='1573741820'
    to='localhost'
    wait='60'
    xml:lang='en'
    xmpp:version='1.0'
    xmlns='http://jabber.org/protocol/httpbind' xmlns:xmpp='urn:xmpp:xbosh'/>
```

Save the file locally on your computer and run:

```
curl -X POST -d@testBosh.txt  https://xmpp.example.com:7443/http-bind/
```

If you get this kind of output, you are good to go as far as BOSH access to XMPP server is concerned, with valid certificate for HTTPS access.

```
<body xmlns="http://jabber.org/protocol/httpbind" xmlns:stream="http://etherx.jabber.
↪org/streams"
    from="example.com" authid="55j3i8xlx2" sid="55j3i8xlx2" secure="true"
    requests="2" inactivity="30" polling="5" wait="60">
    <stream:features>
        <mechanisms xmlns="urn:ietf:params:xml:ns:xmpp-sasl">
            <mechanism>PLAIN</mechanism>
        </mechanisms>
        <bind xmlns="urn:ietf:params:xml:ns:xmpp-bind"/>
        <session xmlns="urn:ietf:params:xml:ns:xmpp-session">
            <optional/>
        </session>
    </stream:features>
</body>
```

Finally, JSXC core being all about **Javascript**, you need to open your browser console (Ctrl-Shift-i on Firefox) and filter all JS information.

Next

Once JSXC is well anchored in your SOGo interface and you have a fully functionnal OTR-capable integrated chat, it is now time to test and use **video chat**.

1.2.5 Overview

1. Install web server, xmpp server, (bosh server)
2. Add jQuery to your site
3. Download and extract jsxc (you have to use the tar file) to a folder of your choice (e.g. `jsxc.example`)
4. Create two folders in your directory `css` and `js`

5. Create a file in each folder: `jsxc.example.css` in `css` and `jsxc.example.js` in `js`
6. Adjust permissions

Your folder structure should now look like:

```
- jsxc.example/
    - jsxc/
        - css/
            - jsxc.example.css
        - js/
            - jsxc.example.js
```

1.2.6 Include

Now include all these files in your template:

```
<!-- Javascript -->
<script src="/jquery.min.js"></script>
<script src="jsxc.example/jsxc/jsxc.bundle.js"></script>
<script src="jsxc.example/js/jsxc.example.js"></script>

<!-- Stylesheets -->
<link href="jsxc.example/jsxc/styles/jsxc.bundle.css" media="all" rel="stylesheet"␣
↪type="text/css" />
<link href="jsxc.example/css/jsxc.example.css" media="all" rel="stylesheet" type=
↪"text/css" />
```

1.2.7 Configure

Add the following lines to our `jsxc.example.js`:

```
$(function() {
    let jsxc = new JSXC({
        loadConnectionOptions: function(username, password) {
            return Promise.resolve({
                xmpp: {
                url: '/http-bind/',
                domain: 'localhost',
                }
            });
        }
    });

    let formElement = $('#form');
    let usernameElement = $('#username');
    let passwordElement = $('#password');

    jsxc.watchForm(formElement, usernameElement, passwordElement);
});
```

Adjust the values according to your application.

1.2.8 Customize style

TODO

1.2.9 Enjoy

Now you should be ready to go.

CHAPTER 2

Contributor Guid

There are plenty of ways to give something back to the open source community. Following you find a couple of them for JSXC and no matter which you choose, we appreciate every help.

2.1 Report a bug

You found a bug in a stable release of JSXC? Please open a new issue for it on our issue tracker. You want to minimize the number of bugs in stable releases? Just watch our repository or follow us on twitter to receive notifications about new prereleases.

Please provide the following points in your issue: - Which jsxc version number do you use? - In which system do you use jsxc? (e.g. nextcloud, sogo, ...) - Can you reproduce your issue? If yes, how? - Which browser and browser version do you use? - Do you have any browser plugins enabled? - Do you see any errors in your javascript console (ctrl+shift+I)? - Which messages appear in your javascript console if you enable debugging? - Do you have any related entries in your XMPP server log or your host system (e.g. nextcloud, sogo, ...)?

2.2 Contribute code

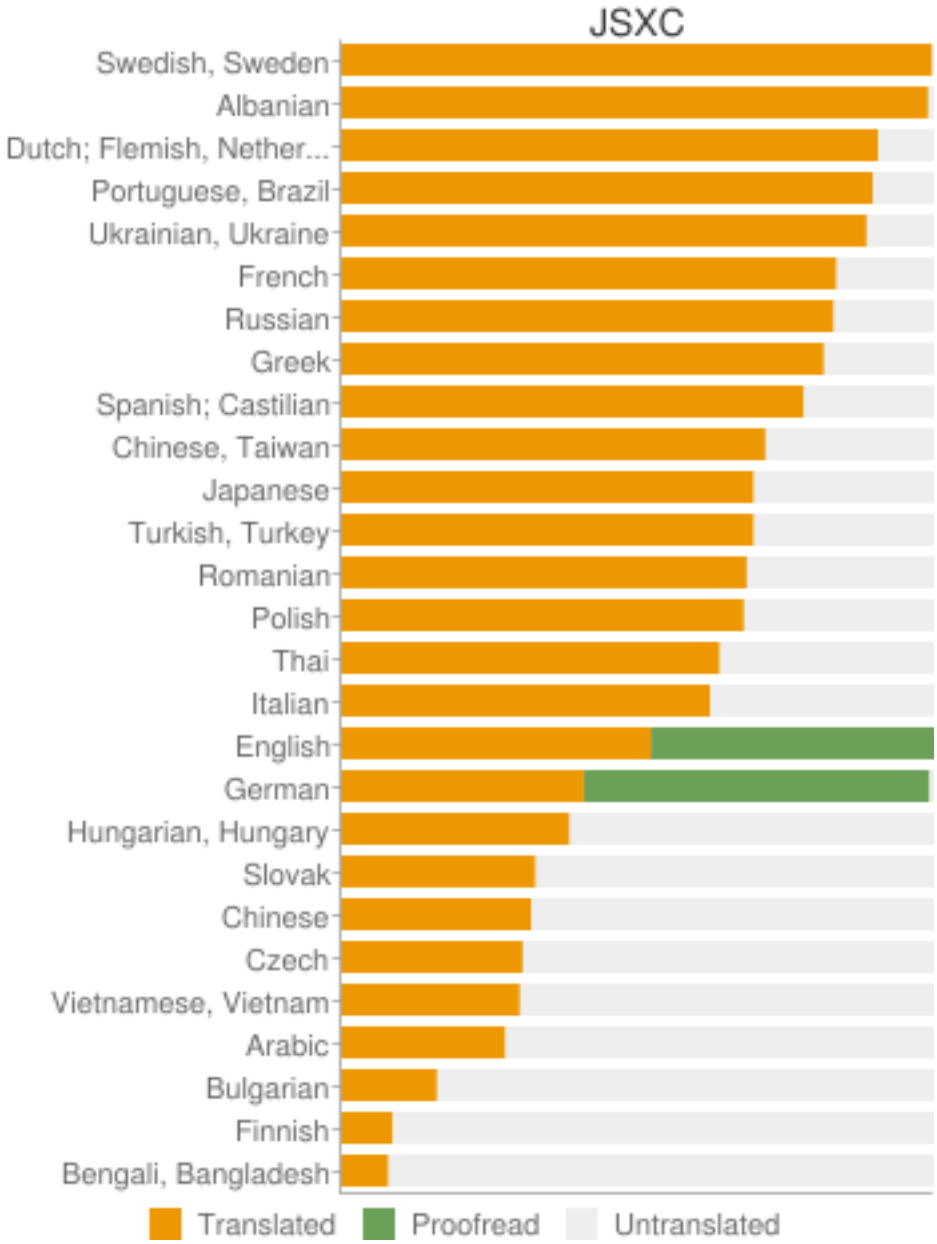
You find detailed information in our *developer notes*, but the following steps describes the general process.

1. Fork the corresponding repository
2. Create a local branch for your fix
3. Commit your changes and push your created branch to your fork
4. Open a new pull request into our master branch

A good way to get familiar with JSXC to work on a starter issue.

2.3 Translate JSXC into your language

JSXC is currently translated into 13 languages and we are always looking for new translators, which add new or complete existing languages. Interested? Request an invitation.



2.4 Announce

You like JSXC and you have a blog? It would be great if you could write a small post about JSXC.

CHAPTER 3

API

Note: We use a Typescript-like syntax to describe our API. For example a following question mark means that this argument or property is optional.

3.1 General

3.1.1 <constructor> JSXC(options?)

You have to initialize JSXC with some options so it can learn about your environment. If you have already some established sessions, those will be restored.

Arguments

- **`options.appName?` (string)** Name of container application (e.g. Nextcloud or SOGo).

  Default: `"web applications"`

- **`options.lang?` (string)** Default language.

  Default: `"en"`

- **`options.autoLang?` (boolean)** Auto language detection.

  Default: `true`

- **`options.rosterAppend?` (string)** Query string for element which should contain your contact list.

  Default: `"body"`

- **`options.rosterVisibility?` ("shown"|"hidden")** Default roster visibility.

  Default: `"shown"`

- **`options.hideOfflineContacts?` (boolean)** Set to true if you want to hide offline contacts.

  Default: `false`

- **`options.loadOptions?` (jid: string, password: string) => Promise<{[id: string]: {[key: string]: any}}>** If you store option changes with `options.onOptionChange` you probably want to restore them on the next login. Just return an object with all id's and the corresponding values.

  Default: `undefined`

- **`options.loadConnectionOptions?` (username: string, password: string) => Promise<ISettings>** This option can provide connection options for every form login (form watcher, login dialog).

  Default: `undefined`

- **`options.onOptionChange?` (id: string, key: string, value: any, exportId: () => any) => void** This function is called every time the user changes a option.

  Default: `undefined`

- **`options.getUsers?` (search: string) => Promise<{[uid: string]: string}>** If you like to provide auto suggestions if the user is adding a contact, you should add this function. Key has be a JID or UID and value a display name.

  Default: `undefined`

- **`options.RTCPeerConfig.ttl?` (number)** Time-to-live for config from url.

  Default: `3600`

- **`options.RTCPeerConfig.iceServers?` (array)** ICE servers like defined in http://www.w3.org/TR/webrtc/#idl-def-RTCIceServer;

  Default: `[{ urls: 'stun:stun.stunprotocol.org' }]`

- **`options.onlineHelp?` (string)** Default: `http://www.jsxc.org/manual.html`

- **`options.storage?` (localStorage|sessionStorage)** Storage backend. Has to implement the Storage interface of the Web Storage API.

  Default: `localStorage`

- **`options.disabledPlugins?` (array<string>)** Default: `[]`

- **`options.connectionCallback?` ((jid: string, status: number, condition?: string) => void)** Default: `undefined`

- **`options.onUserRequestsToGoOnline?` (() => void)** If the user requests to go online again, this function is called. Default: The login dialog is shown.

  Default: `loginDialog`

3.1.2 jsxc.numberOfCachedAccounts: number

Number of restored connections.

3.1.3 jsxc.start(boshUrl, jid, sid, rid)

With JSXC you can also continue a previous established BOSH session.

Arguments

- **`boshUrl` (string)** The URL of your BOSH service.
- **`jid` (string)** Your Jabber Id with or without resource. E.g. `klaus@jsxc.org` or `klaus@jsxc.org/desktop`.
- **`sid` (string)** Your Session ID from your pre-bind session.
- **`rid` (string)** Your Request ID from your pre-bind session.

Returns

`Promise<void>` Promise is resolved if the connection is established and the UI loaded.

3.1.4 jsxc.start(boshUrl, jid, password)

Start a new connection with the given service and credentials.

Arguments

- **`boshUrl` (string)** The URL of your BOSH service.
- **`jid` (string)** Your Jabber Id with or without resource. E.g. `klaus@jsxc.org` or `klaus@jsxc.org/desktop`.
- **`password` (string)** Corresponding password to your Jabber Id.

Returns

`Promise<void>` Promise is resolved if the connection is established and the UI loaded.

3.1.5 jsxc.start()

Show an empty contact list which allows the user to connect to a service by itself.

Returns

`Promise<void>` Promise is resolved if the UI is loaded.

3.1.6 jsxc.startAndPause(boshUrl, jid, password)

Same as `jsxc.start`, but just connects to the XMPP and stops afterwards. Can be used to connect before a form is submitted, or similar.

Returns

`Promise<void>` Promise is resolved if the connection is established.

3.1.7 jsxc.watchForm(formElement, usernameElement, passwordElement)

Watch a login form and use credentials to establish an XMPP connection. You can provide options with `options.loadConnectionOptions`.

Arguments

- **`formElement` (JQuery)** Form element which should be watched for a submit event.
- **`usernameElement` (JQuery)** If the form is submitted get the username from this element.
- **`passwordElement` (JQuery)** If the form is submitted get the password from this element.

3.1.8 jsxc.watchLogoutClick(element)

Watch a logout element and disconnect JSXC before the original click action is processed.

Arguments

- **`element` (JQuery)** Logout element which should be watched for a click event.

3.1.9 jsxc.showLoginBox(username?)

Opens a login modal with field for username and password.

Arguments

- **`username?` (string)** Username for login can be predefined.

3.1.10 jsxc.disconnect()

Disconnect all accounts.

Returns

`Promise<void>` Promise is resolved if all accounts are disconnected.

3.2 User interface

3.2.1 jsxc.addMenuEntry(options)

Add a new entry to the main menu.

Arguments

- **`options.id` (string)** ID of your menu entry.
- **`options.handler` ((ev) => void)** This handler is called if the user clicks on the menu entry.
- **`options.label` (string)** Every menu entry needs a text label.
- **`options.icon?` (string)** If you provide a URL or Base64 encoded image, an icon is shown beside the label.
- **`options.offlineAvailable?` (boolean)** If your entry should also be clickable while the user is offline, set this to `true`. Default: `false`

3.2.2 jsxc.toggleRoster()

Show or hide the contact list.

3.3 Development

3.3.1 jsxc.enableDebugMode()

Enable debug mode for more log messages.

3.3.2 jsxc.disableDebugMode()

Disable debug mode.

3.3.3 jsxc.deleteAllData()

Delete all data stored by JSXC in your data backend.

> **Warning:** This function is only available in debug mode.

Returns

`number` Number of deleted items.

3.3.4 jsxc.getAccount(uid)

Arguments

- **`uid` (string)** UID of account. Usually the bare jid.

Returns

`Account` Account object.

3.4 Account

3.4.1 account.getContact(jid)

Arguments

- **`jid` (string)** Bare JID of contact.

Returns

`Contact|MultiUserContact` Depending on the kind of user, a contact or multi-user contact is returned.

3.4.2 account.createMultiUserContact(jid, nickname, displayName?, password?)

Creates a new MUC room.

Arguments

- **`jid` (string)** Bare JID of new multi-user contact.
- **`nickname` (string)** Desired nickname in MUC room.
- **`displayName?` (string)** Contact name in roster.
- **`password?` (string)** Protect room with password.

Returns

`MultiUserContact` Multi-user contact is returned.

3.5 Contact

3.5.1 contact.openChatWindow()

Opens the chat window.

3.5.2 contact.openChatWindowProminently()

Opens the chat window and highlights it.

3.5.3 contact.addToContactList()

Adds contact to roster or bookmark storage.

3.6 MultiUserContact extends Contact

3.6.1 multiUserContact.join()

Join the given room.

3.6.2 multiUserContact.leave()

Leave the given room.

3.6.3 multiUserContact.destroy()

Destroy the given room.

3.6.4 multiUserContact.getRoomConfigurationForm()

Returns

`JSON` Room configuration form as json.

3.6.5 multiUserContact.submitRoomConfigurationForm(form)

Arguments

- **`form` (JSON)** The filled room configuration form previously retrieved via `multiUserContact.getRoomConfigurationForm()`.

3.7 Services

3.7.1 JSXC.register(service, domain, callback?)

Arguments

- **`service` (string)** The URL of your BOSH service.
- **`domain` (string)** Register a new user with this domain.
- **`callback?` ((form: Form) => Promise<Form>)** If you like to display a custom form, provide a callback.

Returns

`Promise<void>` Promise is resolved if the user was successfully registered.

3.7.2 JSXC.testBOSHServer(url, domain)

Allows you test if a BOSH server is reachable and serving the given domain.

Arguments

- **`url` (string)** URL which you like to test.
- **`domain` (string)** Domain which you like to test.

Returns

`Promise<string>` If the BOSH server is reachable the promise resolves with a constant success string.

In the error case the promise is resolved with an error object. You can call `toString()` to get the error message in english or `getErrorCode()` to get a more generic error code. You find a list of all messages and codes in `src/api/v1/testBOSHServer.ts`.

3.7.3 JSXC.version

Shows the current version of JSXC.

Returns

`string` Version number of JSXC.

CHAPTER 4

How to's

4.1 Screen Sharing

Screen sharing is only supported by Chrome and Firefox (before version 52) through a domain-specific extension. This means, that every domain which tries to access the desktop needs to release his own extension. There are example implementations for Firefox and Chrome which are easy to adapt.

Starting from version 52, Firefox doesn't require an extension for screen sharing.

Screen sharing also requires an encrypted connection (https), otherwise the browser will reject any screen media request.

4.2 WebRTC how to

4.2.1 Why STUN/TURN?

A STUN (Session Traversal Utilities for NAT) server will allow our two end-nodes to know what their public IP is, as we most often sit behind a router in a LAN with private IP addresses. It is only used for that "getting to know each other" phase used to gather all informations about how (if proven possible) establishing our peer-to-peer WebRTC communication.

A TURN (Traversal Using Relay NAT) server is a relay server used when peer-to-peer connection cannot be established. In that case your only option is to use an external server as a relay. Symmetrical NAT, encountered more often in corporate networks, is typically the kind of NAT for which a TURN server will be used as a workaround to establish communication.

4.2.2 Public STUN/TURN Servers

stun.stunprotocol.org (STUN only) is used by default in JSXC. Is known to be prone to DDoS attacks.

> **Warning:** For Russia and parts of Ukraine, this server resolves to 127.0.0.1 (probably as a result of anti-DDoS actions taken by server maintainers). If you experience problems with WebRTC connectivity in JSXC, first of all check the STUN server availability.

URL: `stun:stun.stunprotocol.org`

Google STUN server (STUN only)

URL: `stun:stun.l.google.com:19302`

numb.viagenie.ca (STUN/TURN) is a public STUN/TURN server. Requires registration.

> **Warning:** This server is known not to work with Firefox (fails to collect srflx and relay entries).

URL: *turn:numb.viagenie.ca*

4.2.3 Private STUN/TURN Server

Install and configure coturn

Below commands are for Debian, adjust to you distribution's package manager. We need openssl version 1.0.2 minimum to support DTLS (Datagram TLS). I used the testing repo to install recent versions of both **openssl** and **coturn**:

```
apt install openssl/stretch coturn/stretch
```

You should get validated checks regarding security protocols supported when restarting coturn:

```
0: TLS supported
0: DTLS supported
0: DTLS 1.2 supported
0: TURN/STUN ALPN supported
0: Third-party authorization (oAuth) supported
0: GCM (AEAD) supported
0: OpenSSL compile-time version: OpenSSL 1.0.2j  26 Sep 2016 (0x100020af)
```

Now for the configuration (in `/etc/turnserver.conf`), a minimum working conf dims down to:

```
listening-port=3478
tls-listening-port=5349
alt-listening-port=3479
alt-tls-listening-port=5350
listening-ip=PUB.IP.NUM.1
listening-ip=PUB.IP.NUM.2
relay-ip=PUB.IP.NUM.1
min-port=49152
max-port=65535
verbose
fingerprint
use-auth-secret
static-auth-secret
userdb=/var/lib/turn/turndb
realm=example.com
cert=/etc/ssl/certs/coturn_ca.crt
pkey=/etc/ssl/private/coturn.key
```

(continues on next page)

(continued from previous page)

```
dh-file=/etc/turn/dhparam.pem
no-stdout-log
log-file=/var/log/turn/turn.log

no-sslv3
no-tlsv1
```

This is set up to enable STUN features as well as TURN features. The TLS key/certs have to be fully functional, as TLS is a requirement and not an option (works perfectly fine with Letsencrypt certificate). You need 2 public IPs for STUN to do its magic.

coturn authentication

For TURN, because it handles the whole stream, we want it to only accept relaying authenticated nodes. To do that, you need to create at least a **user account** and a **shared secret**. They both will be used for ephemeral credential validation. In your terminal, generate an account and declare your shared secret via **turnadmin** command:

```
turnadmin -a --db /var/lib/turn/turndb -u username -r example.com -p XXXXXX
turnadmin --db /var/lib/turn/turndb -r example.com --set-secret=XXXXXXXXXXXX
```

Note: from what I gathered, the user password is not used for ephemeral credentials, only for long-term credentials, which are **not secure**.

4.2.4 Configure STUN/TURN server in JSXC

Static

You can pass your STUN/TURN configuration directly to JSXC as init option with the key *RTCPeerConfig.iceServers* using the format described in the W3C WebRTC working draft:

```
RTCPeerConfig: {
    /** ICE servers like defined in http://www.w3.org/TR/webrtc/#idl-def-RTCIceServer␣
↪*/
    iceServers: [{
        urls: ['stun:stun.domain1.org', 'stun:stun.domain2.org']
    }, {
        urls: 'turn:turn.domain3.org',
        username: 'user',
        credential: 'pass'
    }]
}
```

Dynamic

If you like to generate STUN/TURN parameters on the fly, just pass `RTCPeerConfig.url` as init option to JSXC. The endpoint has to respond with a JSON encoded representation of RTCPeerConfig:

```
RTCPeerConfig: {
    url: '/getWebRTCConfig.php',
```

(continues on next page)

(continued from previous page)

```
    /** If true, jsxc send cookies when requesting RTCPeerConfig from the url above */
    withCredentials: false,
}
```

4.2.5 Resources

WebRTC Troubleshooter tests all your WebRTC parameter, like Camera, Network or Connectivity.

Trickle ICE can be used to check general STUN/TURN server availability and functioning.

NAT-Analyzer is a tool to check your NAT type (full cone, symmetric etc.) Requires Java.

4.3 File Transfer

4.3.1 Methods

In general JSXC supports currently data channels (WebRTC) and http upload for file transfer. If http upload is enabled it is the preferred method. If the file is larger as the maximum upload limit, JSXC will fall back to data channels if available.

HTTP Upload

To use http upload your server should support CORS. If it doesn't you have to proxy those request and add the CORS header by yourself as in the example shown.

ejabberd Configuration

Ejabberd mod_httpupload:

```
listen:
...
-
    port: 5443
    module: ejabberd_http
    tls: true
    certfile: "/etc/ejabberd/certificate.pem"
    request_handlers:
    ...
    "upload": mod_http_upload
    ...
...

modules:
...
mod_http_upload:
    docroot: "/ejabberd/upload"
    put_url: "https://@HOST@:5443/upload"
    custom_headers:
    "Access-Control-Allow-Origin": "*"
```

(continues on next page)

(continued from previous page)

```
    "Access-Control-Allow-Methods": "OPTIONS, HEAD, GET, PUT"
    "Access-Control-Allow-Headers": "Content-Type"
...
```

Warning: Ejabberd ignores custom headers if your put url is different from your xmpp domain. See processone/ejabberd#1482.

Prosody configuration

Prosody mod_http_upload:

```
Component "localhost" "http_upload"
http_external_url = "https://EXTERNAL_URL/"

http_upload_file_size_limit = 10485760
```

Apache configuration:

```
<VirtualHost *:443>
        ServerName EXTERNAL_URL

        ...

        <Location /upload/>
        # Allow cross site requests
        Header always set Access-Control-Allow-Origin "*"
        Header always set Access-Control-Allow-Headers "Content-Type"
        Header always set Access-Control-Allow-Methods "OPTIONS, PUT, GET"

        RewriteEngine On

        # modify status code of preflight request
        RewriteCond %{REQUEST_METHOD} OPTIONS
        RewriteRule ^(.*)$ $1 [R=200,L]
        </Location>

        SSLProxyEngine on

        # Just for testing
        #SSLProxyVerify none
        #SSLProxyCheckPeerCN off
        #SSLProxyCheckPeerName off
        #SSLProxyCheckPeerExpire off

        ProxyPass /upload/ https://localhost:5281/upload/
</VirtualHost>
```

WebRTC data channel

See our special *WebRTC page*.

4.4 Using JSXC on Windows with IIS

I'm not going to go in full details yet on how to fully setup each point.

I actually have a domain, and SSL certificates that I created using Let's Encrypt, and I'm using https for my entire setup. So this might make things tricky for you if you are only using http with no actual domain at all, but http://localhost/ should just be fine I think??? lol.

I will assume for the most part you do have some knowledge and experience with IIS, php, MySQL, permissions, configuration files, and etc... You can search online using Google or other for "How to secure IIS", "How to secure wtv".

My setup:

- OS: Windows Server 2016 Data Center (Can be used with any OS that supports IIS)
- WWW Server: IIS 10 (Assumed installed)
  - IIS Web Platform Installer v5 so you can install next two easy
  - IIS Application Request Routing for reverse proxy
  - IIS URL ReWrite for reverse proxy
- DNS Server: BIND9
- php: v7.2.2
- XMPP Server: ejabberd v18.01
- DB: MySQL

4.4.1 Step 1

Make sure your OS, IIS, php, MySQL, Web Application Platform, Application Request Routing, and URL ReWrite module are installed, configured, and working. If you plan on actually testing this outside of localhost, don't forget to open your firewall ports on the server, port forwarding in your router, and to actually get a domain instead of making fake internal domains local to the server using BIND9 DNS server, any other DNS server, or even just using the host file.

Install ejabberd XMPP Server on the same machine, just follow the steps in the installer, once it ask you for host name you can put domain.com or wtv domain you plan on using for testing. Run the "Start ejabberd shortcut" created by the installer on the desktop. Your web browser will open and say the ejabberd has started if all went well. Login to the admin page http://localhost:5280/admin/ with the user/pwd you typed in the ejabberd installer. You can use this admin page to create more users but for now we can do testing with your admin@domain.com account created during installation of ejabberd.

Important: Once ejabberd is installed and running, open up your web browser and test "http://localhost:5280/http-bind/" or "http://localhost:5280/bosh/" (they both do the same thing) to confirm they are accessible and working. This is what we are going to create a URL ReWrite rule or aka alias for using reverse proxy".

4.4.2 Step 2

Make a folder called www on any drive outside Inetpub. This will make your life more easy for editing instead of having to be in admin mode all the time, security, and backing up.

Change security permissions to allow the user IIS_IURS Read and Execute permissions, depending on what you are doing some subfolders or files might need write access to.

4.4.3 Step 3

Decide now if you want JSXC to be a domain or subdomain. Just to be clean, I name my folders like domain.com for a folder/domain or sub.domain.com for a folder/subdomain aka "C:wwwdomain.com" or "C:wwwsub.domain.com". I'll be using domain.com for the example.

Create a folder called domain.com in your www folder and dump the JSXC files in this folder. Configure your JSXC install. I'm only using the internal database for the ejabberd server.

Important: Very important create another folder called bosh or http-bind in your domain.com folder, this will be used to create the reverse proxy to allow JSXC XMPP Client to talk with the ejabberd XMPP Server. It will also store a Web.config file that actually has the settings to do the reverse proxy. So your link would be http://domain.com/bosh/ or http://domain.com/http-bind/" wtv you decided ejabbered works with both links using /bosh/ or /http-bind/ same as JSXC does.** – I would like to thank @skyfox675 for the help on this one and showing his config file sample, because I had never done proxies using IIS before.

By now you should of setup something like roundcube webmail, nextcloud, or owncloud to allow you to use JSXC as a plugin/addon, if not you will have to make your own little html or php app to allow you to even use JSXC. Just follow the JSXC installation section, and pay attention to the "Do you want to integrate JSXC into your own web application".

4.4.4 Step 4

Important: Enabling IIS as a proxy server using ARR.

You have to click on the server itself in the IIS Manager on the left, if you have installed Application Routing and Request you should see Application Routing and Request Cache as an icon on the right, double click the icon.

Now that you have Application Routing and Request Cache open, look to the right you should see "Server Proxy Settings" so single click it.

Don't worry about all the settings on this page, they can be left to default, all you need to do is enable it and click apply on the right.

4.4.5 Step 5

Important: Setting up the Web.config file with rules for URL ReWrite.

You can do this two ways either using IIS Manager, expand your website/domain.com and click on the subfolder bosh or http-bind wtv you decided to name it, and using the GUI to create the reverse proxy url rewrite.

Or just creating a text file using notepad and saving it in your bosh or http-bind folder calling Web and saving it as a config file not a txt file making it be "Web.config".

I used the GUI to make the server variables needed for this ReWrite to work, and used notepad to actually write the config file provided by @skyfox675 in the provided sample.

You need to create some server variables: 1. HTTP_X_FORWARDED_PROTO 2. HTTP_X_FORWARDED_HOST 3. HTTP_X_FORWARDED_PORT 4. ORIGINAL_HOST21

So now travel to Windows File Explorer to "C:wwwdomain.comhttp-bind" or C:wwwdomain.combosh" and create a web.config file if there is none. You can use notepad these files are just text files in the end. Put this text in your new config file, if there was one already created just add this into it by modifying what IIS generated. I hope you know what config files are and how to use them they end up just being xml being just text lol. Don't forget to edit the host variable to be your domain, in the example I have it set to domain.com on the inboundrule, and for the outboundrule there is a "/(http-bind)" you could do "/(bosh)" if you wanted, same goes for the localhost:5280 section:

```
<!-- THANKS TO @skyfox675 for providing a sample -->
<!-- I modified the sample to fit my needs and versions -->
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
    <system.webServer>
        <rewrite>
            <rules>
                <rule name="ReverseProxyInboundRuleBOSH" stopProcessing="true">
                    <match url="(.*)" />
                    <action type="Rewrite" url="http://localhost:5280/http-bind/{R:1}
↪" appendQueryString="true" />
                    <serverVariables>
                        <set name="ORIGINAL_HOST21" value="{HTTP_HOST}" />
                        <set name="HTTP_X_FORWARDED_PROTO" value="https" />
                        <set name="HTTP_X_FORWARDED_PORT" value="5443" />
                        <set name="HTTP_X_FORWARDED_HOST" value="domain.com" />
                    </serverVariables>
                </rule>
            </rules>
            <outboundRules>
                <rule name="ReverseProxyOutboundRuleBOSH" preCondition=
↪"ResponseIsHtml1">
                    <match filterByTags="A, Form, Img" serverVariable="RESPONSE_
↪LOCATION" pattern="^http://[^/]+/(.*)" />
                    <action type="Rewrite" value="http://{ORIGINAL_HOST21}/{C:1}/{R:1}
↪" />
                    <conditions>
                        <add input="{ORIGINAL_HOST21}" pattern=".+" />
                        <add input="{URL}" pattern="^/(http-bind)" />
                    </conditions>
                </rule>
                <preConditions>
                    <remove name="ResponseIsHtml1" />
                    <preCondition name="ResponseIsHtml1">
                        <add input="{RESPONSE_CONTENT_TYPE}" pattern="^text/html" />
                        <add input="{RESPONSE_STATUS}" pattern="3\d\d" />
                    </preCondition>
                </preConditions>
            </outboundRules>
        </rewrite>
        <httpErrors errorMode="DetailedLocalOnly" />
        <directoryBrowse enabled="false" />
    </system.webServer>
</configuration>
```

What all this does is convert http://domain.com/http-bind/ from the web server side to http://localhost:5280/http-bind" on the ejabberd xmpp server side on the same machine, could be a different machine if you wanted. This also terminates https if you are using it at the IIS server, but that's another issues for another time.

So for every single application you use JSXC client in with ejjaberd server, you would tell it your xmpp server is located at http://domain.com/http-bind/ so the local machine and remote machines can find the xmpp server, if you would not do this and only did localhost, the remote machines would never find the xmpp server because locally to themselves there is no server.

Doing this also allows you to not have to create more bindings in IIS for your website aka virtual-host causing other issues. My only bindings on IIS for the site I have JSXC running on is port 80 and 443, and I redirect all to 443.

And you are done and things should be working.

CHAPTER 5

Development

5.1 Developer notes

5.1.1 Get code

Please execute the following commands to get a copy of the code:

Core

```
git clone https://github.com/jsxc/jsxc.git
```

Apps

```
git clone https://github.com/nextcloud/jsxc.nextcloud.git
# and/or
git clone https://github.com/jsxc/jsxc.sogo.git
# and/or
git clone https://github.com/jsxc/jsxc.ilias.git
```

5.1.2 Install yarn

We use yarn as our package manager. Please follow there docs to install it.

Warning: We used npm and grunt in the passed to concatenate, convert, and validate our source files. It's likely that you still find it at some places.

First install all dependencies with *yarn install*. To build the project, just run `yarn start` in its top-level directory. If you like to start a development server and watch all files for changes, some project offer you *yarn dev*.

Now you are ready for development.

5.1.3 Workflow

- run `yarn watch` in the project root folder
- make your modifications in *src/* and *scss/*
- check your results on *example/index.html*
- `git add ...`
- `git commit` (should run our pre-commit hook)
- `git push`

5.1.4 Notes

Yarn commands

A few helpful yarn commands you should know to simplify your life.

- `start` create a production ready build
- `watch` create a development build and rerun on file changes
- `dev` similar to `watch`, but it will start a development server
- `fix` beautifies all files and makes sure the pre-commit hook is satisfied

To run a command, execute *yarn COMMAND*.

Core Structure

TODO

Debugging

Open your Javascript console (e.g. ctrl+shift+I) and enter *jsxc.storage.setItem('debug', true)* to obtain debug messages.

5.1.5 Locales

We use webtranslateit.com to organize our translations and wti to synchronise those.

5.2 Creating a release

- update change log
- update translation `wti pull --all`
- update example

- create build
  - increase version number in `package.json` - run `node scripts/build-release.js` (use `--release` for stable releases)
- update documentation
  - wiki
  - website
- publish to app store, npm and similar
- announce new release
  - blog post
  - twitter
  - mailing list

CHAPTER 6

References

6.1 External REST specification

External REST API of the Nextcloud JSXC app.

6.1.1 Version 1

Schema

All API access should over HTTPS, and accessed the `https://YOUR_CLOUD/index.php/apps/ojsxc/ajax/`. All data is sent and received as JSON. Every request has to contain a signature to verify the request.

Client Errors

500 Internal Server Error: `{"result": "error", "data": {"msg":"An error occured"}}`

Header Signature

Every request to an API endpoint needs a valid X-JSXC-SIGNATURE header of the form `HASH_ALGO=hmac(HASH_ALGO, REQUEST_BODY, API_SECRET)`.

Endpoints

Check Password

```
POST /externalApi.php
```

Parameters

Name	Type	Description
operation	"checkPassword"	
username	string	
password	string	
domain	string	The domain is used to check additionally the password for `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "noauth"
}
```

```
Status: 200 OK

{
  "result": "success",
  "data": {
    "uid": "foobar"
  }
}
```

User exists

```
POST /externalApi.php
```

Parameters

Name	Type	Description
operation	"isUser"	
username	string	
domain	string	The domain is used to test additionally the user `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "success",
```

(continues on next page)

(continued from previous page)

```
  "data": {
    "isUser": true
  }
}
```

Get shared roster

```
POST /externalApi.php
```

Parameters

Name	Type	Description
operation	"sharedRoster"	
username	string	
domain	string	The domain is used to get the shared roster for `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "success",
  "data": {
    "sharedRoster": {
      "fritz": {
        "name": "Fritz Froh",
        "groups": ["group1", "group2"]
      },
      "georg": {
        "name": "Georg Geizig",
        "groups": ["group3"]
      }
    }
  }
}
```

6.1.2 Version 2 (draft)

Warning: This version is still in development!

Schema

All API access should over HTTPS, and accessed the `https://YOUR_CLOUD/index.php/apps/ojsxc/api/v2/`. All data is sent and received as JSON. Every request has to contain a signature to verify the request.

Client Errors

500 Internal Server Error: `{"result": "error", "data": {"msg":"An error occured"}}`

Header Signature

Every request to an API endpoint needs a valid X-JSXC-SIGNATURE header of the form `HASH_ALGO=hmac(HASH_ALGO, REQUEST_BODY, API_SECRET)`.

Endpoints

Check Password

```
POST /checkPassword
```

Parameters

Name	Type	Description
username	string	
password	string	
domain	string	The domain is used to check additionally the password for `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "noauth"
}
```

```
Status: 200 OK

{
  "result": "success",
  "data": {
    "uid": "foobar"
  }
}
```

User exists

```
POST /isUser
```

Parameters

Name	Type	Description
username	string	
domain	string	The domain is used to test additionally the user `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "success",
  "data": {
    "isUser": true
  }
}
```

Get shared roster

```
POST /sharedRoster
```

Parameters

Name	Type	Description
username	string	
domain	string	The domain is used to get the shared roster for `username@domain` (optional)

Response

```
Status: 200 OK

{
  "result": "success",
  "data": {
    "sharedRoster": {
      "fritz": {
        "name": "Fritz Froh",
        "groups": ["group1", "group2"]
      },
      "georg": {
        "name": "Georg Geizig",
        "groups": ["group3"]
      }
    }
  }
}
```