Omni Channel Management (OCM)

Tetherfi Communication Web API Reference Document

Document Version : 2.19.09.13

Last Updated Date : 2019 September 13

Version	Edited By	Date	Change
1.0	*Nuwan*	*11/2016*	*Initial draft*
1.1	*Vineeth*	*11/2016*	*Review, restructure and approve.*
1.2	*Nuwan*	*17/03/2017*	*Added cross and quick references. Further formatting.*
1.3	*Nuwan*	*15/06/2017*	- *Session defined* - *OnEcho() added* - *OnEnd() added* - *OnRemoteEndClosed() deleted* - *OnServerStateChanged() added*
1.4	*Nuwan*	*13/07/2017*	*RequestCallback() added.*
1.5	*Nuwan*	*31/07/2017*	*Customer info updated.* *Chat History as parameter to Start* *Added 2 error codes* *Health check section added*
1.6	*Nuwan*	*05/09/2017*	*Reason parameter added to End and OnEnd such that End(reason), OnEnd(Initiator, reason)*
1.7	*Vineeth*	*10/10/2017*	*'Secure deployment model' section added*
1.8	*Nuwan*	*26/12/2017*	*AccessToken added and customer details extended.* *API renamed to livechat.js (backword compatible with libtextchat.js).*
1.9	*Nuwan, Sharadhi*	*02/01/2018*	*Fileupload function added.* *AppMessage type added.* *Secure Deployment model updated.*
2.0	*Saman, Nuwan*	*03/01/2018*	*Audio Video related functions added*
2.18.09	*Sandeep*	*18/09/2018*	*Captcha integration for web api.*
2.18.10	*Sandeep*	*24/10/2018*	1. *Agent details in session description.* 2. *Log To Server function added.*
2.19.01	*Shanaka*	*12/01/2019*	*Update Audio/Video API documentation*
2.19.02.18	*Sandeep*	*18/02/2019*	*File transfer functions added.*
2.19.03.07	*Adhip*	*07/03/2019*	*Added initial co-browse documentation (sequence diagram, scripts size, pre-requisites)*
2.19.04.03	*Sandeep, Adhip*	*03/04/2019*	*Merged engine.js, __textchat.js and livechat_http.js into livechat.js* *Added extra parameters to SetCobrConfigs and added description for the same.* *Updated Co-Browsing Initial Description*
2.19.04.11	*Adhip, Vineeth*	*11/04/2019*	*Updated Co-browse scripts size* *Segregated scripts section to clearly mention the scripts that developers need to use when integrating the Web API interface.*

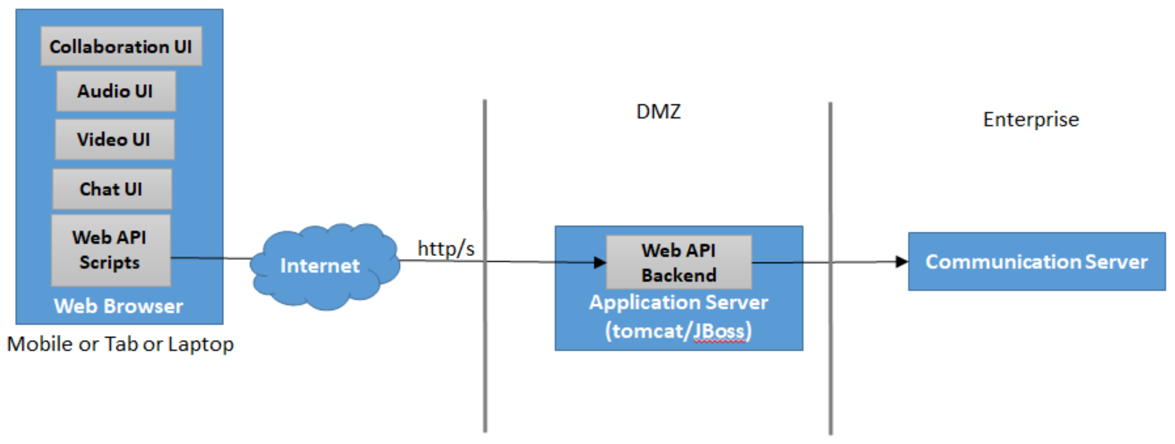
			Updated the written statements in the Introduction, Components, status check and WebAPI sections, added a diagram in the Introduction section and renamed certain headings. Changed the document name to “Tetherfi Communication Web API Reference Document”
2.19.04.22	*Sandeep*	*22/04/2019*	*Added error code - MaxConcurrentSessionCountReached*
2.19.05.30	*Sandeep*	*30/05/2019*	*Added API call flows for Text chat and Audio-Video.*
2.19.06.25	*Sandeep*	*25/06/2019*	*1. Updated Text chat and Audio-Video API call flow diagrams. 2. Added events. - OnAppMessageReceived - OnConferenceUserLeft - OnHistoryReceived*
2.19.08.02	*Sandeep*	*02/08/2019*	*Added a synonymous event to OnCallbackAutoRegistered.*
2.19.08.09	*Sandeep*	*09/08/2019*	*Added error code - AlreadyEndedSession*
2.19.08.20	*Sandeep*	*20/08/2019*	*Added function and event for Create Campaign Contact.*
2.19.09.13	*Sandeep*	*10/09/2019*	*Multi-chat functionality section added.*

Contents

Reference

Introduction

This document is prepared to help developers to integrate with Tetherfi Communication Library to implement Chat, Audio, Video & Collaboration features with Mobile devices (Native Mobile) or web browsers (Web Interface).



Components

Below are the components involved in the Communication Channel implementation:

1) **'Communication Server'** windows service (service to handle communication sessions).
2) **'Web API Backend'** Http add-on Web Application (for http interface).
3) Mobile Libraries (for developing native mobile applications).
4) **'Web API Scripts'** Java Script Library (for web interfaces).

Web API

This document covers the **'Web API'** interface, which is used for web browser based integration. This interface requires a servlet container (e.g. tomcat or JBoss) to host Http add-on. Developers are required to include below scripts in their web applications. The browser should have cookies enabled for communication.

This is a complete API to add web communication support for any website with minimum effort. It suits any kind of GUI.

Scripts

Mandatory Scripts - Following scripts are essential to be included for communication:

```
<host>/tetherfi/livechat/livechat.js
<host>/tetherfi/livechat/interface/__clientlog.js
```

Following scripts can be **optionally included** as follows:

1) If **audio/video calling** is to be integrated, then in addition to the above (essential scripts) the following scripts are to be included as well:

   ```
   <host>/tetherfi/adaptor.mini.js
   <host>/tetherfi/livechat_video.mini.js
   ```

2) If **co-browsing** is to be integrated, then in addition to the above (essential scripts) the following scripts and styles are to be included manually on the co-browse page:

   ```
   <link href="Styles/fontawesome-all.min.css" rel="stylesheet">
   <link href="Styles/tetherfi-cobrowse.min.css" rel="stylesheet">
   <script type='text/javascript' src='Scripts/cobrowse-configurations.js'></script>
   <script type='text/javascript' src='Scripts/tetherfi-cobrowse-plugins.min.js'></script>
   <script type='text/javascript' src='Scripts/tetherfi-cobrowse.min.js'></script>
   ```

3) If **multi-chat functionality** is to be integrated, then in addition to the above (essential scripts) the following scripts are to be included as well:

   ```
   <host>/tetherfi/livechat/livechat.js?multichat=true
   ```

Classes

Customer Details

```
function CustomerDetails()
{
    this.eKCIN = "";
    this.eRegStatus = ""
    this.eCustName = "";
    this.pChannel = "";
    this.pPageTag = "";
    this.pTopic = "";
    this.eCin = "";
    this.eAuthStatus = "";
    this.eAuthLevel = "";
```

```
    this.pKaiSid = "";
    this.eCustSeg = "";
    this.eMobileNo = "";
    this.eEmail = "";
    this.pBranding = "";
    this.pCountry = "";
    this.pLang = "";
    this.pAccessToken = "";
}
```

Class to hold customer information as well as initial chat session history.

Session Details

```
function SessionDetails ()
{
    this.eCustDetails;

    this.ToString = function () {
         this.lib = "web";
        this.comm_path = "sg_pweb";
       this.vachatid = this.pKaiSid;
       this.enable_autocb = false;
       var d = new Date();
      this.pClientTime = d.getHours() + ":" + d.getMinutes() + ":" +
                  d.getSeconds() + ":" + d.getMilliseconds();
      this.eEnteredCaptcha = "";
       return JSON.stringify(this);
      }
}
```

Usage:

```
var details = new SessionDetails();
details.eCustDetails =  "..";  or { ... };
details.eEnteredCaptcha = "<user's input>;
livechat.Start ( details, "History" );
```

Objects

livechat

libtextchat (synonymous with livechat for backward compatibility)

Predefined objects containing functions and variables application is expected to use.

Session

Dynamic object which carry information about an already opened session in server. This object is sent by the server as a parameter of OpenSuccess event.

```
{
    agents(optional) : [
      { name : “”, id : “” , index: “”},
      { name : “”, id : “” , index: “”}
    ],
    trans : [
        { type:”msg”, by : “c”, msg: “”, time : “” },
         { type:”msg”, by : “a”, msg: “”, time : “” },
         { type:”connect”, id : “”, name: “”, time : “” },
         { type:”disconnect”, id : “”, name: “”, time : “” }
    ]
}
```

`agents(optional):` Details of the CSO agent who is handling this session at the moment. This attribute will not present or null before CSO connect for the session (CSO answer is Informed to application by RemoteUserConnected event). Attributes of this object are same as arguments of `RemoteUserConnected` event.

`index` : agent’s position In trans. (e.g. if index=2 , this agent haven’t seen 1st 2 messages as he joined later but 3rd message onwards).

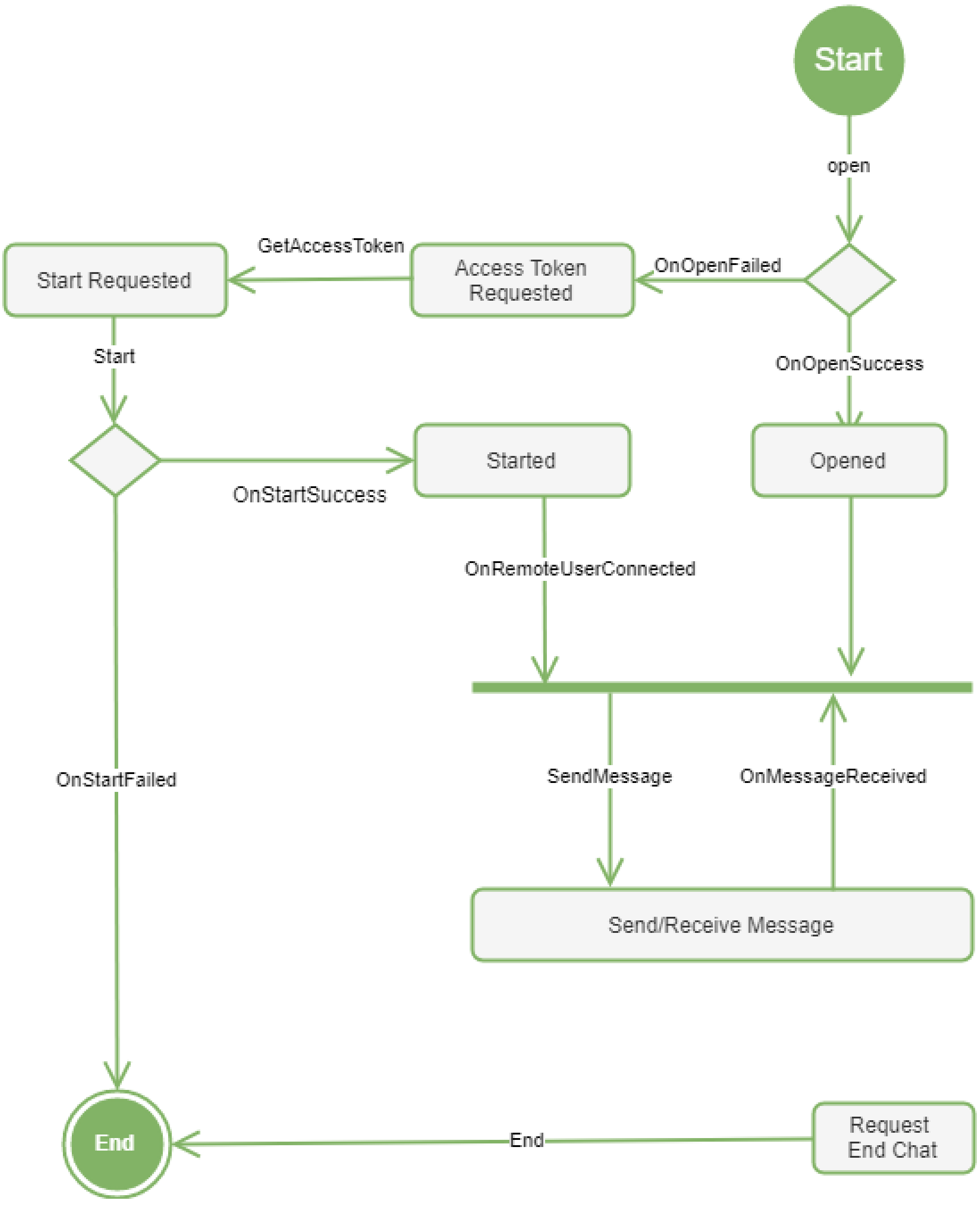
`trans` : Array of message records exchanged within the session. Each object has following attributes.

`by` : Possible values `‘c’`(to indicate customer) or `‘a’`(to indicate CSO agent).

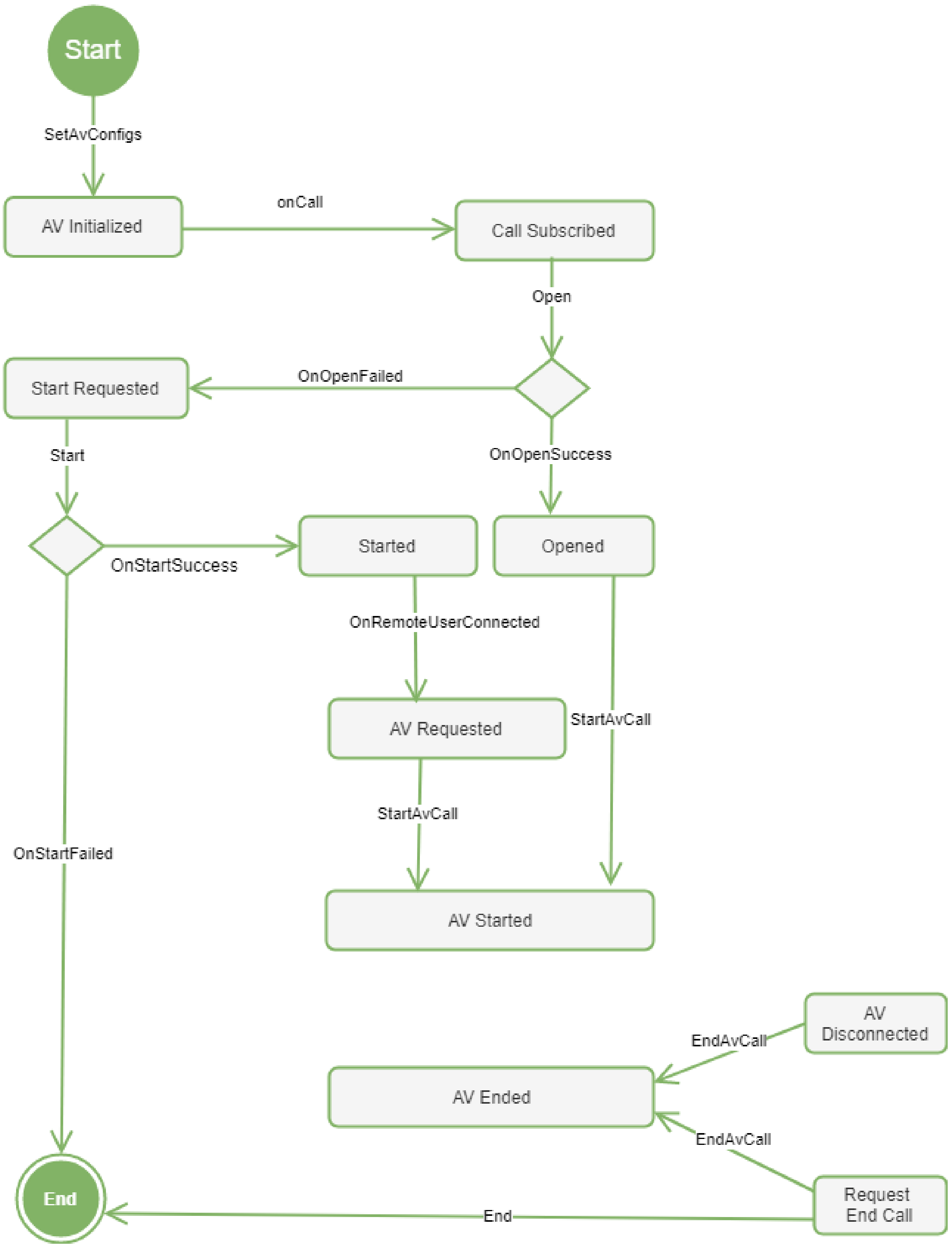
`msg` : URLEncoded message text.

`time` : Server’s time of message accept.

API call flow for Text Chat



API call flow for Audio-Video



Basic Communication Functions

```
livechat.GetAccessToken( onsuccess, [onerror] );
```

Session token to implment additional security to block XSRF and XSS attacks. Session token is unique for the session and expected to add into customer details if details are to be encrypted. Either `onsuccess` or `onerror` will be invoked to indicate status. `onsuccess` will carry returned access token in it's 1st parameter.

```
onsuccess = function( token ){ .. }

onerror = function(err){ .. }
```

```
livechat.Start( custDetails, VaSession );
```

Request to starts a chat session with specified customer details. Either `StartSuccess` and `StartFailed` events are fired to indicate the status.
In implmenting floating chat, `Open` must be called prior to calling this function. Please read on `Open` to find criteria to call `Start` in a floating chat sessions.
custDetails – Either an object of CustomerDetails or an encrpted JSON string of the same.
VaSession - String represenation of Vachat session history

```
livechat.End( reason );
```

Terminates the chat session. CSO is notified about the end of the chat session.
Parameter reason : Any string passed by the application which will come back to app in the second parameter of `OnEnd` event.

```
livechat.Open( );
```

Opens an existing on going chat session. This is used implement floating chat capability in a browser session. In such a case web client application must invoke this function before it invoke `Start`. Status will be informed to the application either in `OnOpenSucess` or `OnOpenFailed(errorCode).` If the error code `OnOpenFailed` carry is `NoSessionFound`, then application can proceed to invoke `Start()` and begin a new chat session.
If `OpenSuccess` callback is invoked or `OnOpenFailed` with a different error code, Application should NOT proceed to `Start`.

```
livechat.Close();
```

Application must call this function as it closes browser tab or unload the page without wanting to end chat session. E.g. inavigating sequence to a different page.
At page1 – invoke `Close()`
At page2 – invoke `Open()` and handle `OpenSuccess` event.

```
livechat.SendMessage( msg );
```

Sends the message *msg* to Customer service officer. A echo of the message is sent back by server on `OnEcho` event for all pages with active chat sessions (of same brower session).
Id – Any id to track the status of message delivery.

```
livechat.SendAppMessage( type, obj );
```

Sends the message object *'obj'* of type '*type'* to TMAC.

`type` – Any application defined string
`obj` – Any Javascript object

```
livechat.NotifyTypingStateChange( state );
```

Informs typing state events to CSO. State is an integer such that, 0-Typing started, 1-Typing stopped. Refer `OnTypingStateChanged.`

```
livechat.RequestCallback( contact );
```

Register a callback request with Tetherfi callback service. `OnCallbackRequestStatus` event will return information about success or faliure. *Note:* OnCallbackRequestStatus event can be fired even without application calling this function : Please refer `OnCallbackRequestStatus` for more details.
contact – Phone number to contact back customer.

Audio – Video Communication Functions

Audio and video related functions

```
livechat.SetAvConfigs( config );
```

Sets Audio/Video configuration parameters.

`config` – A JavaScript object with following format.

```
var config = {
    MediaTransport : {
            iceServers : [
                    {urls: '<IceUrl>', credential: '<IceId>', username: '<IceName>'}
            ],
    },
    MediaCapture : {
            audio: {
                    enabled : <AudioEnabled>,
            },
            video: {
                    enabled : <VideoEnabled>,
                    width : <VideoWidth>,
                    height : <VideoHeight>,
            }
    },
    Display : {
            OnConnected : <OnConnectCallbackFn(remoteStreams, localStreams)>
            OnDisconnected : <OnDisconnectCallbackFn()>
    }
}
```

`IceUrl` – The WebRTC AVProxy server URL.
`IceName` – A valid user name from the AVProxy.
`IceId` – A valid ID from the AVProxy for the user.

`AudioEnabled/VideoEnabled` – Boolean values specifying the call should be audio call or video call with voice. When both values are set to true, the call will be a video call with voice.
`VideoWidth/VideoHeight` – The resolution for the video to be sent when video call.

`OnConnectCallbackFn(remoteStreams, localStreams)` – A JavaScript function that takes two arguments. Both arguments are Array of MediaStream objects. You can set a MediaStream object to the srcObject attribute of HTML5 audio/video element to play them. This will be called when the call is connected.

`OnDisconnectCallbackFn()` – A JavaScript function without any arguments. This will be called when the call gets disconnected.

```
livechat.StartAvCall( sid );
```

Starts a webRTC audio / video call opening video views at the same time

`sid` – A String representing the session Id of the video call.

`livechat.EndAvCall();`
Ends webRTC audio-video call

`livechat.Pause( audio, video );`
Pause or Unpause audio and video `audio` - If set mute microphone, unmute otherwise `video` - If set pause camera capture, resume otherwise

Co - Browsing Communication Functions

Before enabling Co-Browsing or Screen-sharing in an application, there are few pre-requisites needed as follows:

- Co-browsing or Screen-sharing makes use of Tetherfi Live chat and Tetherfi Multimedia Agent Client, as the underlying frameworks for communication. It is required to have Tetherfi Communication platform installed first before starting Co-browsing.
- To start Co-browsing or Screen-sharing the user needs to be connected to an CSO first.
- Since Tetherfi's Screen-sharing and Co-browsing technology captures the entire DOM of user's web page, **Content Security Policy must be disabled for the pages** which require co-browsing and sharing.

There are few ways of how co-browsing can be enabled in a web site:

- If the user wants co-browsing to happen on the current window, both live chat and co-browse JS files have to be added to the web page.
- If the user wants co-browse to happen on a different web page other than the live chat integrated web page, live chat scripts will be loaded on the chat page and the co-browse scripts will be loaded on the other window.

`livechat.SetCobrConfigs(config, domControlList, isMousePointerCapture, isAnnotationEnabled);`
Set configurations related to co-browsing.

config – A JavaScript object with following format.

```
   var config = {
       MediaTransport : {
                MediaTransport: {
        iceServers: [
            {urls: '<IceUrl>', credential: '<IceId>', username: '<IceName>'}
        ],
        iceTransportPolicy: "all"
    },,
                MediaCapture: {
        audio: {
            enabled: false,
            autoGainControl: true,
            noiseSuppression: true,
            echoCancellation: true
        },
        video: {
            enabled: true,
            width: 1920,
            height: 1080,
            frameRate: 6.0
        }
    },,
                Signaling: {
        iceRestart: true
    },
                Network: {
                    Bandwidth: {
        SingleAudioTrackLimit: 8,
        SingleVideoTrackLimit: 2048,
        CumulativeSessionLimit: 5000

    }
                },
                Display: {
                    OnConnected: function (remoteStreams, localStreams) { },
                    OnDisconnected: function () { },
                    OnAcquireCustomCanvasStream: function (stream) { }
                }
            };
```

IceUrl – The WebRTC AVProxy server URL.
IceName – A valid user name from the AVProxy.
IceId – A valid ID from the AVProxy for the user.

OnConnected – Method called when co-browsing starts. remoteStreams and localStreams can be ignored.
OnDisconnected - Method called when co-browsing is ended.
OnAcquireCustomCanvasStream – Before co-browsing starts, the screen that needs to be shared should be streamed by returning a capture stream.

domControlList – List of elements along with their id's that needs to be passed. Currently only textbox, radiobutton, checkbox and dropdown are supported.
Format is as shown below:

```
{
    textboxId: ["textbox1", "textbox2"],
    radioButtonId: ["radiobutton1", "radiobutton2"],
    checkboxId: ["checkbox1", "checkbox2"],
    dropdownId: ["dropdown1", "dropdown2"]
}
```

Here textbox1, radiobutton1, checkbox1, dropdown1 etc. are the id's of textbox, radiobutton, checkbox and dropdown respectively.
If atleast **any one of the DOM Controls** on the page **are not to be monitored** then the array for that element can be kept as a blank array.
If **none of the DOM Controls** on the page are to be monitored then the `domControlList` parameter can be passed as blank.

`isMousePointerCapture` – If set then the Customers Mouse pointer will be captured and sent to Agent

`isAnnotationEnabled` – If set then Customer will be shown the annotation toolbar menu and will be allowed to annotate.

```
livechat.StartCobr();
```

Starts a new co-browsing session

```
livechat.EndCobr(triggerDisconnect, sendEndMessage);
```

Ends current co-browsing session

`triggerDisconnect` - If set then cobrowse session will be disconnected (usually set to true).
`sendEndMessage` - If set then livechat will inform CSO side of disconnect.

Captcha integration

Captcha integration related functions

```
livechat.setCaptchaImage (img);
```

Sets the url to the <img>. In this url, query param **reset = false.**
Url is mapped to CaptchaImageServlet and which loads the captcha image.

```
livechat.resetCaptchaImage (img);
```

Resets the captcha image by setting url query param, **reset = true.**
Fetches the new captcha image from CaptchaImageServlet and updates the image.

Log to server

Log related function.

```
livechat.slog (type, unit, msg);
```

Logs the client log messages in proxy server.
type - “info”, “warn”, “error”.
unit – calling method name.
msg – Log message.

File Transfer

File transfer related functions.

```
livechat.prepareDownloadURL(fileName);
```

Prepares the URL to download the file.
fileName – Name of the file to download

```
livechat.uploadFiles (fileList);
```

Uploads the files to the Server.
fileList– List of files

Create Campaign Contact

Functions to create new contact for the campaign.

```
livechat.CreateCampaignContact(data);
```

Creating new contact for the campaign
data – request payload in json format

Events

Below are the list of events webpage receives from server. They are **folked** for all open pages (which has active chat) in the same web session. Each opened page is required to handle these events.

```
livechat.OnStartSuccess = function( ){ .. }
```

Indicates success of `Start`. Application will start receiving further events. Next event will be `OnCallQueued` [happy path] or `OnSystemNotification` [error path].

```
livechat.OnStartFailed = function(errorCode){ .. }
```

Indicates the failure of `Start`. Possible Error codes are :

```
StartError
ServerInternalConnectivityError,
AlreadyStartedAndInSession,
CookiesDisabled
UnOpenedSessionPerformedStart
CaptchaDoesNotMatch
CaptchaNotFound
CaptchaIsEmpty
MaxConcurrentSessionCountReached
AlreadyEndedSession
```

```
livechat.OnOpenSuccess = function( Session ){ .. }
```

Indicates the success of `Open`.
Parameter `Session` contains information about on going session.

```
livechat.OnOpenFailed = function(errorCode){ .. }
```

Indicates the failure of `Open`. Possible error codes are :

```
CookiesDisabled
NoSessionFound
AlreadyEndedSession
```

The code `'NoSessionFound'` is actually not an error. On this code application can proceed to call `Start` to begin a new chat session.

```
livechat.OnEnd= function( initiator, reason ){ .. }
```

Indicate end of the chat session.
Parameter initiator :
`server` – session end has been initiated by server side.
`client` – session end has been initiated by client side.
`lib` – Not possible to determine which side initiated and client library terminated client session.
Parameter reason :
when initiator = 'client' : Reason is any content passed by application to the End function
when initiator = 'server' : reason is undefined in this version
when initiator = 'lib' : reason is SessionLostWhileOffline'

```
livechat.OnRemoteUserConnected= function( name, agentId, sessionId ){ ..
}
```

Customer Service Officer connected to the chat session
name – Name of the CSO login
agentId – Id of the CSO login

This event can be fired multiple times in same page as CSO changes in transfers.

```
livechat.OnSystemNotification = function(reasonCode){ .. }
```

```
livechat.OnCallbackAutoRegistered = function(reasonCode){ .. }
```

(Synonym with livechat.OnSystemNotification for backward compatibility.)

Callback request was recorded by the server due to the reason mentioned in reasonCode. In most of the errors a callback is recorded prior to termination. Possible reason codes strings are :
`ErrorInGetAgentDetails` – Can't retrive CSO information.
`ErrorAgentDetails` – Error in CSO information.
`NoStaffedAgents` – No CSOs are login to the system at the moment.
`QueueIsTooBig` – There are CSOs logged-in but too many unanswered calls are in the queue.
`TimedOutInQueue` – Configured time elapsed in the queue but could not reach a CSO.
`InternalServerError` – Callback recorded due to Internal server error.

```
livechat.OnCallbackRequestStatus = function(isCustomerReq, status ){ ..
}
```

Indicates the status of the call back request made. Call back could have been requested either by CSO or user of the application which is indicated in the parameter isCustomerReq.
isCustomerReq= true : Request has been made by this application
isCustomerReq=false : CSO has requested the system to record a call back request.
status=true : Successful

```
livechat.OnInternalServerError= function(errorCode){ .. }
```

Unexpected Internal server error. Errorcode is a string which is not defined here. Application should terminate the session in this case.
errorCode :
`UnOpenedSessionNotAllowed`

```
livechat.OnCallQueued = function( queuePosition ){ .. }
```

This is the next event application receive after OnStartSuccess. Note : Only for sessions started by calling Start receives this event. Sessions started by calling Open and receiving OpenSuccess will not receive this event as application is not creating making new chat call in such cases. (Chat sessions are routed as phone calls in the internal system)

```
livechat.OnMessageReceived = function( msg ){ .. }
```

Text message received from CSO.

```
livechat.OnEcho = function( msg ){ .. }
```

Server's echo of messages client sent. This is used to display messages between multiple tabs. Application should use this event to display user's messages so that multiple brower tabs can be synchronized.

```
livechat.OnTypingStateChanged = function(state){ .. }
```

Typing `state` changed at CSO. 0-Typing started, 1-Typing stopped. Please refer `NotifyTypingStateChange`

`livechat.OnAppMessageReceived = function(agent, msg){ .. }`
Receives text message from TMAC. agent – Name of the CSO login msg – Text message

`livechat.OnConferenceUserLeft = function(agent, msg){ .. }`
Customer Service Officer leaves the conference chat session agent – Name of the CSO login msg – Text message

`livechat.OnHistoryReceived = function( history ){ .. }`
Application receives customer chat history. History – Chat history

`livechat.OnServerStateChanged = function(state){ .. }`
Indicate server online state changed. Possible string values of `state` are as below : `offline` – Server became un-reachable within last 9 seconds. `online` – Contacted server back. Hint: Application may show a message to user upon each case and disable/enable text input. E.g. offline - "We are trying to contact server" , online - "Hi, We are back online.." Note: Online state event can immediately be followed by `OnEnd('lib', 'SessionLostWhileOffline')` event if session has been ended by the time user get network connection back.

`livechat.OnResetCaptcha= function(){ .. }`
URL query param, **reset** indicates whether captcha text has to be generated newly or retrieved from session. This value will be true for reset captcha functionality. This event can be fired to update the captcha image.

`livechat.OnUploadComplete= function(response){ .. }`
Receives the file name, file size and type as response upon file upload complete.

`livechat.OnCampaignContactCreated= function(response){ .. }`
Receives resultCode and resultMessage as response in json format.

System status check

Below request can be used for, continuous or on-demand system checks.

Request

Url for system connectivity and health-check is as below.

URL : `<host>/tetherfi/livechat/status`
Type: `GET`

Response

```
Http Status : 200,
Response Message : ONLINE
```

Indicate system can accept chat.

```
Http Status : 200,
Response Message : OFFLINE (message)
```

Indicate that the system can’t accept chat sessions due to a technical limitation.

This does not include business limitations like, CSO’s not being staffed, End of business hours, etc. Those scenarios are handled differently within the chat session.

```
Http Status : 4xx,
```

Application is down or not reachable via reverse-proxies.

Multi-chat Functionality

Multi-chat functionality can be used when one customer wants to chat with multiple agents using different chat sessions for each agent in the same browser/tab. Below mentioned changes are required to support this functionality.

Mandatory Scripts

Following script is essential to be included:

```
<host>/tetherfi/livechat/livechat.js?multichat=true
```

Objects

multilivechat

Predefined object containing functions and variables application is expected to use.

Session

Dynamic object array which carry information about already opened sessions in server. This object array is sent by the server as a parameter of OnOpenSuccess event.

```
[
        {
            agents(optional) : [
              { name : “”, id : “” , index: “”},
              { name : “”, id : “” , index: “”}
            ],
            trans : [
                { type:”msg”, by : “c”, msg: “”, time : “” },
                 { type:”msg”, by : “a”, msg: “”, time : “” },
                 { type:”connect”, id : “”, name: “”, time : “” },
                 { type:”disconnect”, id : “”, name: “”, time : “” }
            ],
             clientId: “”
        },
        {
            agents(optional) : [
              { name : “”, id : “” , index: “”},
              { name : “”, id : “” , index: “”}
            ],
            trans : [
                { type:”msg”, by : “c”, msg: “”, time : “” },
                 { type:”msg”, by : “a”, msg: “”, time : “” },
                 { type:”connect”, id : “”, name: “”, time : “” },
                 { type:”disconnect”, id : “”, name: “”, time : “” }
            ],
            clientId: “”

        }
]
```

Basic Communication Functions

All the functions marked below has an additional parameter **Sid** (web chat session id).

SendMessage, SendAppMessage, NotifyTypingStateChange

Below mentioned function has an additional parameter **clientId** (unique client id)

Start

Events

All the events marked below has an additional parameter **Sid** (web chat session id).

OnStartFailed, OnEnd, OnCallbackAutoRegistered, OnCallbackRequestStatus, OnCallQueued, OnMessageReceived, OnEcho, OnTypingStateChanged, OnAppMessageReceived, OnConferenceUserLeft, OnHistoryReceived

Below mentioned events has an additional parameter **clientId** (unique client id)

OnRemoteUserConnected, OnStartSuccess

Communication model for floating web-chat

Floating chat UI is a popped-up chat GUI component that stays above all web page the user can see, within the same website. Popped-up GUI can be placed anywhere within user's view and usually is either can be dragged or can be minimized to avoid obstructing the website view.

The Web API exposed here, in Http/https mode provides additional 2 functions to join for an ongoing session and leave a session without ending (within the same browser session). Related functions are `Open` and `Close` respectively. Session leaving and Joining has to take place within the same browser session (e.g. opening a new tab). No details are passed to the `Open` function as generally new page is not aware about details of the previous page.

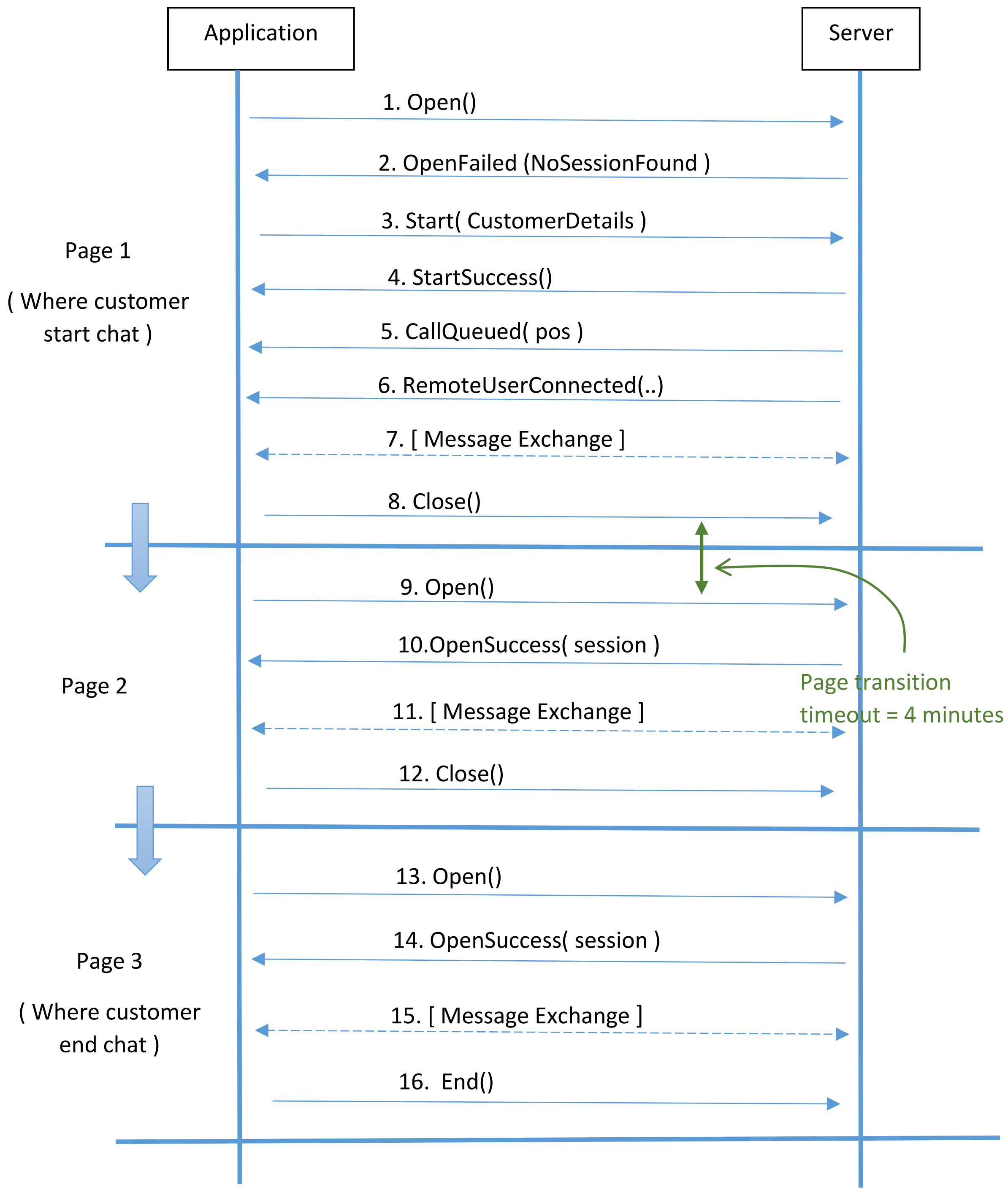
These two functions are used to implement floating chat feature where we address below cases.

- Navigating to a different page of the same website by clicking a link.
- Opening a new tab for a page in the same website.

The diagram at next page shows a scenario where user is changing the page while in the chat session.

Scenario: User start chat session in Page 1 and then navigate to Page 2 and 3 while chatting with a CSO. On page 3, he decide to end the chat.

[To next page]


Application
Server
1. Open()
2. OpenFailed (NoSessionFound)
3. Start(CustomerDetails)
4. StartSuccess()
5. CallQueued(pos)
6. RemoteUserConnected(..)
7. [Message Exchange]
8. Close()
Page 1
(Where customer start chat)
9. Open()
10.OpenSuccess(session)
11. [Message Exchange]
12. Close()
Page 2
Page transition timeout = 4 minutes
13. Open()
14. OpenSuccess(session)
15. [Message Exchange]
16. End()
Page 3
(Where customer end chat)

[From previous page]

According to above scenario, customer's side performed the session end.

In a similar scenario If CSO perform the termination, the application will receive `OnEnd` event at anypoint after `4.StartSuccess` . In such a case, application is not expected to call anything but merely handle the event.

In the diagram above, it has been marked 2 messages exchanged before and after the page transition. The session end timer starts as soon as Close is invoked (If there are no other tabs accessing same chat session).

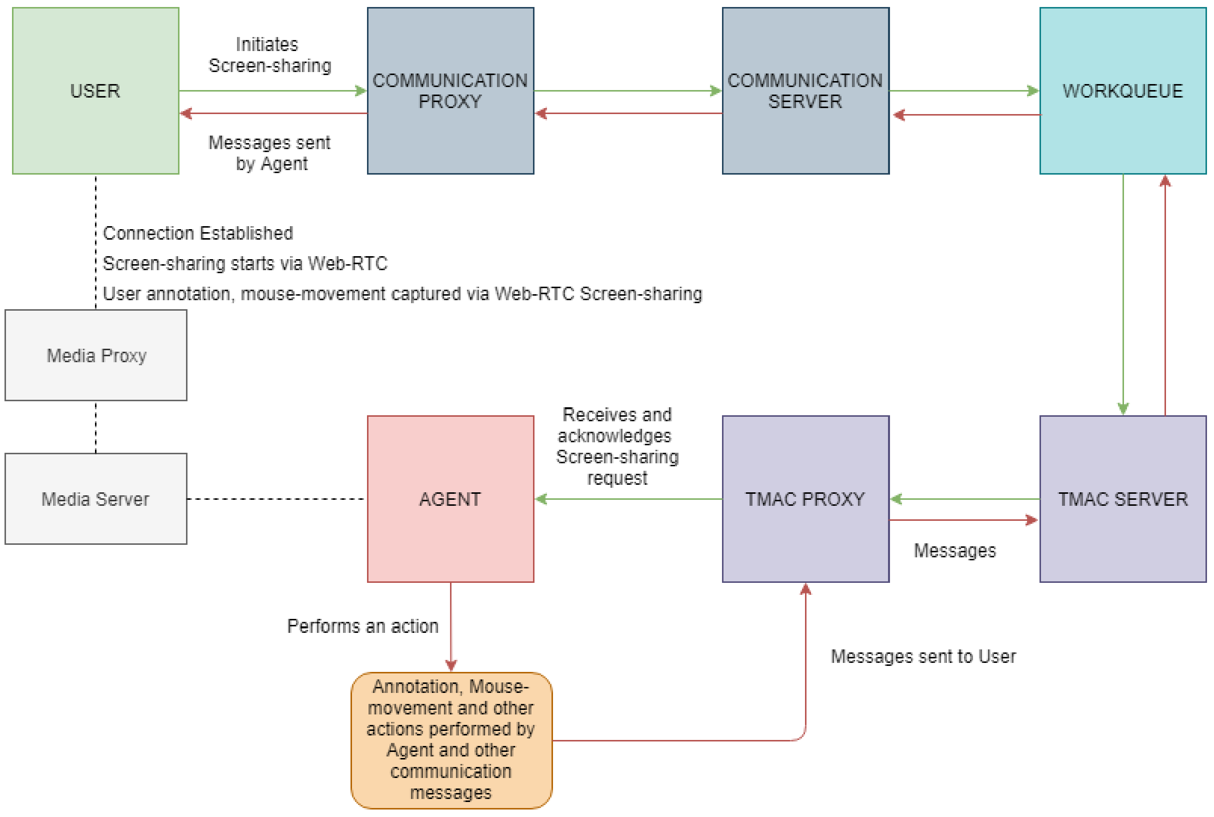
Page 1, 2 and 3 in above scenario are assumed included with following scripts

```
<host>/tetherfi/livechat/livechat.js ( 61.9 KB )
<host>/tetherfi/livechat/interface/__clientlog.js ( 1 KB )
<host>/tetherfi/adaptor.mini.js ( 90 KB )
<host>/tetherfi/livechat_video.mini.js ( 32 KB )

Total script size = 180 KB
```

Design Diagram for Screen-sharing

Following Diagram illustrates shows a simple flow along with the components used of Screen-sharing on the web between the Agent and the User:



USER
Initiates Screen-sharing
Messages sent by Agent
COMMUNICATION PROXY
COMMUNICATION SERVER
WORKQUEUE
Connection Established
Screen-sharing starts via Web-RTC
User annotation, mouse-movement captured via Web-RTC Screen-sharing
Media Proxy
Media Server
AGENT
Receives and acknowledges Screen-sharing request
TMAC PROXY
TMAC SERVER
Messages
Performs an action
Messages sent to User
Annotation, Mouse-movement and other actions performed by Agent and other communication messages

```
Scripts used for Co-browsing/Screen-sharing:
```

1) Following scripts are used for communication i.e. sending messages, logging and other communication.

   ```
   <host>/tetherfi/livechat/livechat.js (61.9 KB)
   <host>/tetherfi/livechat/interface/__clientlog.js (1 KB)
   ```

2) Following styles are used for buttons created used for annotation, setting the styles of canvas and other components used.

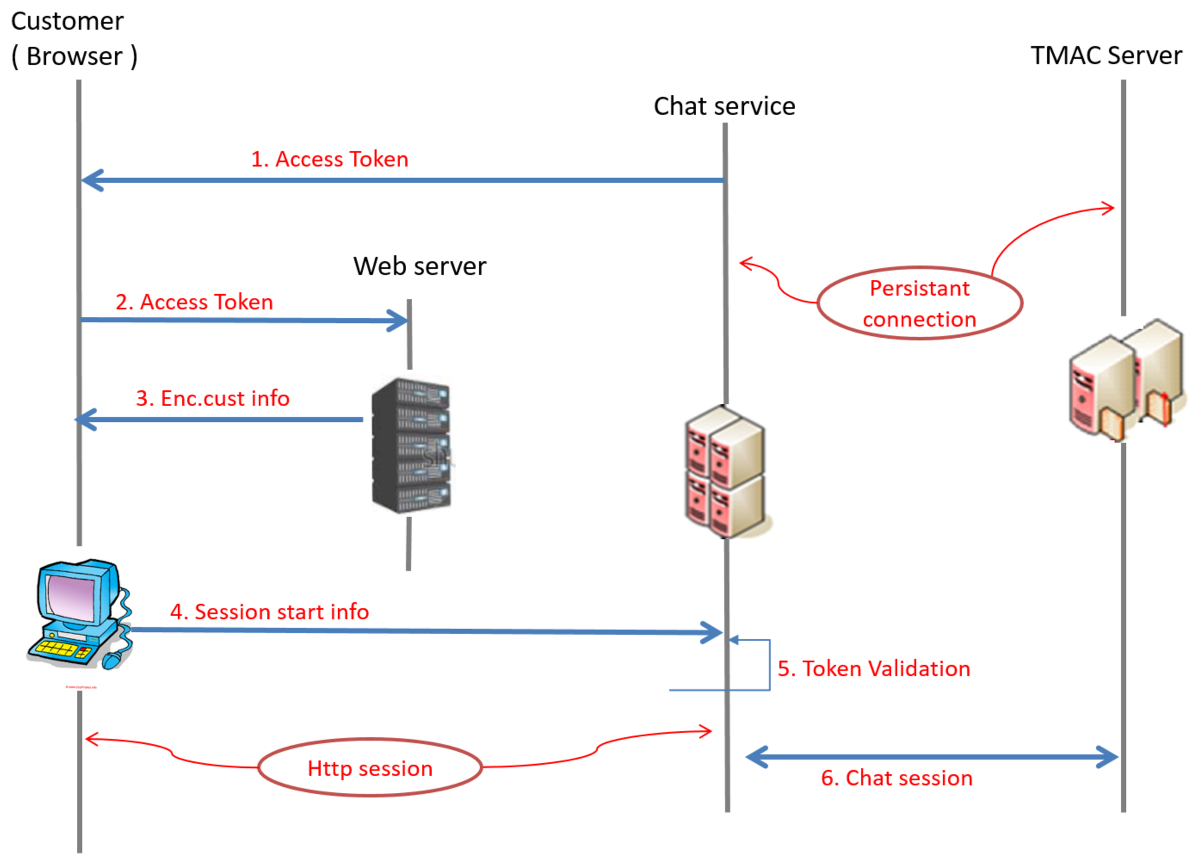
   ```
   <link href="Styles/fontawesome-all.min.css" rel="stylesheet">
   (106KB)
   <link href="Styles/tetherfi-cobrowse.min.css" rel="stylesheet">
   (8KB)
   ```

3) Following scripts are used for monitoring the customer screen, mouse pointers, configuring and 3rd party plugins used to achieve screen-sharing and co-browsing.

   ```
   <script type='text/javascript' src='Scripts/cobrowse-
   configurations.js'></script> (3KB)
   <script type='text/javascript' src='Scripts/tetherfi-cobrowse-
   plugins.min.js'></script> (214KB)
   <script type='text/javascript' src='Scripts/tetherfi-
   cobrowse.min.js'></script> (113KB)
   ```

Total script size = 507 KB

Secure Deployment Model



Above is one of the generic deployment models used for a secure deployment with ‘Chat Proxy’, which validates sessions using an access token. Chat feature is allowed from customer’s existing web site/mobile app, for which the customer’s web page/mobile app needs to be modified.

- Customer logs into Customer’s INB Server with 2FA and is now inside the secure site hosted by customer.
- The webAPI is loaded to customer’s browser, once the customer request for chat, audio or video communication.
- For every active session with INB Server there will be a token generated from the chat service.
- WebAPI will establish maintation the communication with chat service
- The chat service would validate the token with against encrypted data created by web server.
- Session initiation will be aborted in case of a token mismatch.