Exotel Voice Websdk

Integration Guide

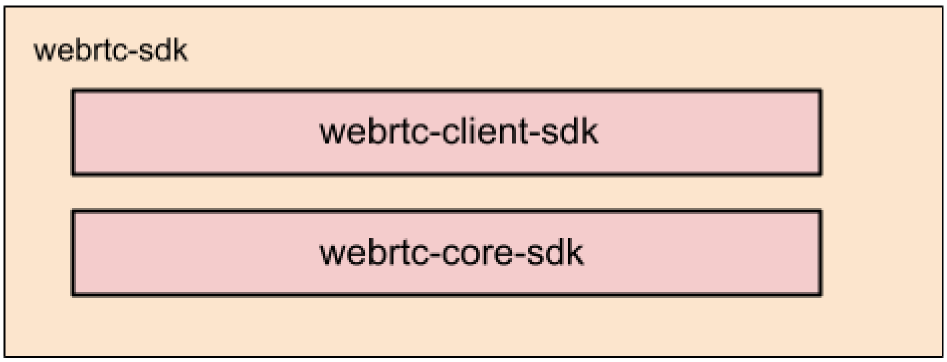
Version	Date	Changes
1.0	30-03-2022	Initial Upload
1.1	02-05-2022	Added Web Client APIs
1.2	15-06-2022	Removed Core SDK. Updated Flows.
1.3	30-12-2022	Exotel SDK bundle integration and demo
1.4	01-Oct-2024	Added data types info for function arguments

Content

1. Introduction

Exotel Webrtc SDK library enables you to add the voip calling feature into your web app. This document outlines the integration and usage. The Exotel Webrtc SDK package is layered. We have web-client-sdk that provides higher level APIs and callbacks to register/unregister, receive calls and operate on calls like mute/unmute, hold/unhold. And we have webrtc-core-sdk that provides the underlying state machine and SIP protocol stack.

The current document provides API and Callback details for the web-client-sdk.



2. Licensing

You need an [exotel](#) account to use the voip calling functionality with this websdk. Contact [exotel support](#) for demo and account creation.

Exotel organization npm account : **@exotel-npm-dev**

For using the above organizational account for publishing , an invitation will be required. And for the same please contact the account manager or hello@exotel.com

3. Glossary

Terminology	Description
App	Web Application
VOIP	Voice over IP

Client	User / Subscriber / Client signing up to use the web app
Customer	Exotel's customer licensing the SDK

4. Getting Started

4.1. Software Package

The Exotel Webrtc SDK software package includes

- @exotel-npm-dev/webrtc-client-sdk library from npm repository
- Integration Guide
- Sample application for reference

4.2. Add Web Client SDK Library to Project

Install the compatible node.js and npm versions. For example, on ubuntu 20.04, following versions are compatible

- nodejs version: >=14.x
- npm version: 6.14.4

Once nodejs is installed, download the packages from the repository and install them in your npm modules directory. To download the library from the repository follow the steps given below,

Step 1: After successful login, install the webrtc-sdk library.

```
npm install @exotel-npm-dev/webrtc-client-sdk
```

Step 2: Verify if node_modules are created inside the sample app folder. Start the npm

Step 3: In order to launch the application, run the following command. Application should start on https://localhost:3000

```
npm start
```

4.3. Supported Browsers

The SDK has been verified with the following browsers.

OS	Chrome	Firefox	Safari (> V11)	Edge
Windows				
Linux				
MacOS				

Browser not compatible with OS
WebSDK supported

5. Web Client SDK APIs and Integration Workflow

Initialize	SDK module needs to be loaded and initialized with callback listeners. SDK will callback web-app to deliver various events. Initialization is done during page load.
Register	WebSDK registers with exotel platform using credentials to make / receive the calls. This is usually done when user logs into the CRM and is ready to take calls.
Calls	After successful registration, SDK is ready to take calls. Using APIs, it can control the call operations like reject, terminate, mute, unmute, hold, unhold. SDK emits appropriate events based on which call icons can be updated.
Unregister	To stop receiving calls, unregister the sdk from the exotel platform. This is usually done when user logs off or when they can't take calls.

5.1. Initialize Library

WebRTC client SDK library needs to be initialized prior to using. A sample code snippet to do so is as below.

Step1: Import the exotel client library as below.

```javascript
import { ExotelWebClient as exWebClient } from
'@exotel/webrtc-client-sdk';
```

Step2: Initialize the Exotel Web Client with SipAccountInfo and Callbacks using the API initWebrtc.

```javascript
// Initialise sipAccountInfo  dictionary
 const sipAccountInfo = {
   'userName': 'Username',
   'authUser': 'Username',
   'sipdomain': 'Domain',
   'domain': 'HostServer' + ":" + Port
   'displayname': 'DisplayName',
   'secret': 'Password',
   'port': 'Port',
   'security': 'wss',
 }

// Initialize Callbacks:
exWebClient.initWebrtc(sipAccountInfo,
                   RegisterEventCallBack,
                   CallListenerCallback,
                   SessionCallback)
```

This API also takes as input three callbacks:

- RegisterEventCallBack handles registration states as they change.
- CallListenerCallback handles call events as they occur.
- SessionCallback handles notifications for multiple tab sessions.

The details of the callbacks are explained in the corresponding flows.

Argument Details

Args	DataType
`sipAccountInfo`	object
`RegisterEventCallBack`	Callback function
`CallListenerCallback`	Callback function
`SessionCallback`	Callback function

sipAccountInfo		
authUser	String	SIP username to register.
userName	String	Used as a unique map index for phones. Same as authUser.
displayName	String	Local Displayname on Dialer.
secret	String	SIP Password
sipdomain	String	SIP public domain
security	String	“wss”/”ws” typically “wss”
port	String	443 for Websockets.

5.2. Opus Codec Preference - Optional

5.2.1. To enable opus codec, it can be enabled from Exotel Voip domain settings, for that we can raise a request to hello@exotel.com to enable the opus codec support.

5.2.2. Once opus codec is enabled, then and if browser is not preferring opus codec in SIP 200 OK, then we have an API to make opus codec as preferred codec.

```javascript
exWebClient.setPreferredCodec("opus");
```

5.3. Download Logs - Optional

5.3.1. To help with debugging or sharing logs with support, whenever user will face the issue, we can invoke downloadLogs method.

5.3.2. This will download a ".txt" file (e.g. "webrtc_sdk_logs_2025-04-03.txt") containing 1000 logs stored in the browser's localStorage.

```javascript
            exWebClient.downloadLogs();
```

5.4. Register the SIP Phones

Before receiving / making any call, the SIP phone needs to be registered; and the API for this purpose is "DoRegister".

API Name	ExotelWebClient.DoRegister
Args	None
Exported To	Application UI

```javascript
//Ensure that initWebRTC is invoked before calling DoRegister
exWebClient.DoRegister();
```

Once the registration has been done, the response comes in the registrationCallback with the events, “registered ”/ ”terminated” as below.

API Name	RegisterEventCallBack		
Args	**Params**	**Type**	**Values**
	state	String	"registered" / “terminated” / “sent_request” / “unregistered”
	phone	String	Username
Exported To	Application UI		

```javascript
function RegisterEventCallBack (state, phone){
    if (state === 'registered') {
      // Successful registration
      setRegState(true)
    } else if (state === 'unregistered') {
      // Successful unregistration
      setRegState(false)
    } else if (state === 'terminated')  {
      // Registration/Unregistration failed
      setRegState(false)
    } else if (state === 'sent_request')  {
      // Registration/Unregistration Request Sent
      if (unregisterWait === "true") {
        unregisterWait = "false";
        setRegState(false)
      }
    }
  }
```

5.5. UnRegister the SIP Phones

To stop getting calls anymore, the SIP phones need to be unregistered. The API to do so is “UnRegister”.

API Name	ExotelWebClient.UnRegister
Args	None
Exported To	Application UI

```javascript
//Ensure that initWebRTC is invoked before calling DoRegister.
exWebClient.UnRegister();
```

The response to unregistration comes in the registrationCallback as below.

<table>
<tr><td>API Name</td><td colspan="3">RegisterEventCallBack</td></tr>
<tr><td rowspan="3">Args</td><td>Params</td><td>Type</td><td>Values</td></tr>
<tr><td>state</td><td>String</td><td>"registered" / "unregistered" / “terminated” / “sent_request”</td></tr>
<tr><td>phone</td><td>String</td><td>Username</td></tr>
<tr><td>Exported To</td><td colspan="3">Application UI</td></tr>
</table>

```javascript
  function RegisterEventCallBack (state, phone){
    if (state === 'registered') {
      // Successful registration
      setRegState(true)
    } else if (state === 'unregistered') {
      // Successful unregistration
      setRegState(false)
    } else if (state === 'terminated')  {
      // Registration/Unregistration failed
      setRegState(false)
     } else if (state === 'sent_request')  {
      // Registration/Unregistration Request Sent
      if (unregisterWait === "true") {
```

```
            unregisterWait = "false";
            setRegState(false)
          }
        }
      }
```

Note 1: If you have more than one phone, the second argument “phone” would give the indication as to which phone got unregistered.

Note 2: In some cases there won't be any successful response for the “unregister”. In such situations, it is advised to handle “unregistration” flow with the “sent_request” event as shown in the above example.

5.6. Receive Calls

Once the registration has happened, and when there is an incoming call, the callback registered for “Call Events” would get an event along with the details of the incoming call.

API Name	CallListenerCallback		
Args	**Params**	**Type**	**Values**
	callObj	Object	{ callId, callState, callDirection, callStartedTime, remoteDisplayName } callId {String}: sip call-id callState {String}: incoming callDirection {String}: incoming callStartedTime {String}: call start time remoteDisplayName {String}: callee name
	eventType	String	incoming : shows incoming call message connected : open dialer callEnded : close dialer activeSession: session continuity
	phone	String	Username identifying the phone to which the call is coming.
Exported To	Application UI		

In the following snippet,the UI states are appropriately modified based on the call events.

```javascript
function CallListenerCallback(callObj, eventType, phone) {
    if (eventType === 'incoming') {
      // Incoming Call
      setCallComing(true)
    }  else if (eventType === 'connected') {
      // Call in connected state
      setCallComing(false)
      setCallState(true)
    }  else if (eventType === 'callEnded') {
      // Call ended
      setCallComing(false)
      setCallState(false)
    } else if (eventType === 'terminated') {
      // Call terminated
      setCallComing(false)
      setCallState(false)
    }
}
```

5.7. Accept Calls

Once the incoming call event is received, based on the user action the call can be accepted. The API to do so is “Call.Answer” as below. The call object could be obtained by invoking the “getCall()” method in exWebClient object.

API Name	Call.Answer
Args	None
Exported To	Application UI

```javascript
  function acceptCallHandler() {
    call =  exWebClient.getCall()
    call.Answer();
```

```
}
```

The response event to accept calls comes in “CallListenerCallback”. Successful acceptance results in a “connected” event. Other possible events are:

“callEnded” : Call ended locally.
“terminated” : Call terminated remotely.

JavaScript

```javascript
function CallListenerCallback(callObj, eventType, phone) {
   if (eventType === 'incoming') {
     // Incoming Call
     setCallComing(true)
   }  else if (eventType === 'connected') {
     // Call in connected state
     setCallComing(false)
     setCallState(true)
   }  else if (eventType === 'callEnded') {
     // Call ended
     setCallComing(false)
     setCallState(false)
   } else if (eventType === 'terminated') {
     // Call terminated
     setCallComing(false)
     setCallState(false)
   }
}
```

5.8. Hangup / Reject Calls

Once the incoming call event is received, based on the user action the call can be rejected. The API to do so is “Call.Hangup” as below.

API Name	Call.Hangup
Args	None
Exported To	Application UI

```javascript
  function rejectCallHandler() {
    call =  exWebClient.getCall()
    call.Hangup();
  }
```

For call hangup by local user, the response comes as a “callEnded” event. For call hangup by remote user,, the event comes as “terminated”.

```javascript
function CallListenerCallback(callObj, eventType, phone) {
    if (eventType === 'callEnded') {
      // Call ended
      setCallComing(false)
      setCallState(false)
    } else if (eventType === 'terminated') {
      // Call terminated
      setCallComing(false)
      setCallState(false)
    }
 }
```

5.9. Mute / Unmute Calls

Once the call is in progress, based on the user action the call can be muted/unmuted. The APIs to do so are “Call.Mute” and “Call.UnMute” as below.

API Name	Call.Mute
Args	None
Exported To	Application UI

API Name	Call.UnMute
Args	None

Exported To	Application UI

You can also use a single API to toggle the mic using Call.MuteToggle.

API Name	Call.MuteToggle
Args	None
Exported To	Application UI

```
JavaScript
function muteHandler() {
   call =  exWebClient.getCall()

   //call.MuteToggle(); // or
   if (!callOnMute) {
     call.Mute()
     callOnMute = true
   } else {
     call.UnMute()
     callOnMute = false
   }
 }
```

There is no callback event for mute. Call remains “connected” but with the mic muted. This is a toggle operation.

5.10. Hold / Resume Calls

Once the call is in progress, based on the user action the remote user can be set on hold/unhold. The API to do so is “Call.Hold” and “Call.UnHold” as below.

API Name	Call.Hold
Args	None
Exported To	Application UI

API Name	Call.UnHold
Args	None
Exported To	Application UI

You can also use a single API to hold the remote caller using Call.HoldToggle.

```javascript
function holdHandler() {
    call =  exWebClient.getCall()

    //call.HoldToggle(); // or
    if (!callOnHold) {
      call.Hold()
      callOnHold = true
    } else {
      call.UnHold()
      callOnHold = false
    }
  }
```

There is no callback event for call hold. Call remains in “connected” but in sendonly mode. This is a toggle operation.

5.11. Send DTMF

Once the call is in progress, based on the user action the remote user can be sent DTMF digits. The API to do so is “Call.sendDTMF” as below

API Name	Call.sendDTMF		
Args	**Params**	**Type**	**Values**
	digit	String	0-9, A-D, *, #

Exported To	Application UI

```javascript
call = exWebClient.getCall()
if (call) {
call.sendDTMF(digit);
}
```

5.12. Multitab Scenarios

When the webrtc sdk is loaded in multiple tabs then all the instances will register with the backend and receive incoming call alerts. There are two ways to avoid this,

- Maintain a single login session for the user in your webapp so that only one instance of the webrtc sdk is loaded. This is preferred.
- Maintain a parent-child relationship across the tabs in your webapp so that call is handled only in the parent tab.

Exotel Webrtc-SDK supports multiple tab sessions using Broadcast Channel. In this feature, session listener and session callbacks can be used to get the indication on child tabs. A SessionListener creates a broadcast channel and sends a broadcast event to each child tab for the following events, incoming / connected / callEnded / re-register. When a child tab (as per the logic maintained by the “Application Backend”.) receives a session event, it may notify but not handle the callback.

When the parent tab is destroyed, a re-register event comes to child tabs, based on the logic of the client, a child can opt to be the new parent.

Note 1: The logic of maintaining the parent and child tabs has to be in “Application Backend” by the “Customer”.

Note 2: If multitab scenario is not used, there is no need to handle the events in the session callback.

API Name	SessionListener
Args	None

Exported To	Application UI

```JavaScript
SessionListener();  // To be called during initialization.
```

API Name	SessionCallback		
Args	**Params**	**Type**	**Values**
	callState	String	incoming / connected / callEnded / re-register incoming : child tab to show a notification message connected : child tab to close the notification message callEnded : child tab to close the notification message re-register : child tab to register when parent tab is closed ice_gathering_state_<state>: ICE gathering state changed. <state> will be the new ICE gathering state (e.g., new, gathering, complete). ice_connection_state_<state>: ICE connection state changed. <state> will be the new ICE connection state (e.g., new, checking, connected, disconnected, failed, closed). Media_permission_denied: User denied media (microphone/camera) permissions.

	phone	String	username identifying the phone to which the call is coming.
Exported To	Application UI		

A sample code snippet for session callback is as below.

```javascript
function SessionCallback(callState, phone) {
      /**
       * SessionCallback is triggered whenever an incoming call arrives
       * which needs to be handled across tabs
       */
      switch(callState){
        case 'incoming':
            console.log('incoming call' + phone)
            /**
             * Display a different notification popup in case of child
tabs
             */
            if(window.sessionStorage.getItem('activeSessionTab') !==
      'parent0'){
            const message = 'Incoming call from ' + phone + ' ,Switch
tab to find dialpad'
            setMessage(message);
          }
        break;
        case 'callEnded':
          /**
           * When call is either accepted or rejected then this is gets
shutdown
           */
          console.log('call ended' + phone)
          setOpen(false);
        break;
        case 'connected':
          /**
           * When call is connected close the notification popup on
child tabs
           */
          console.log('call connected' + phone)
          setOpen(false);
        break;
```

```
        case 're-register':
          /**
           * In case if the main/parent tab is closed then make the
subsequent tab in the tab list as the parent tab
           * and send register for the same and also make that tab as
the master
           */
      // ICE Gathering State Events
        case 'ice_gathering_state_new':
        case 'ice_gathering_state_gathering':
        case 'ice_gathering_state_complete':
                 console.log('ICE state change:', callState, 'for call
from:', phone);
            // Handle ICE state changes as needed
            break;

        // Media Permission Error
        case 'media_permission_denied':
                 console.log('Media permission denied for call from:',
phone);
              showErrorMessage('Microphone access is required for calls.
Please allow microphone permissions.');
            break;
            window.sessionStorage.removeItem('activeSessionTab');
            window.sessionStorage.setItem('activeSessionTab',
'parent0');
            sendAutoRegistration();
          }
          break;
      }
  };
```

5.13. Device and Network Diagnostics

Initialize diagnostics.

Diagnostics can be initialized by passing two callbacks, one troubleshooting logs and one to get the responses of diagnostics back.

API Name	ExotelWebClient.initDiagnostics		
Args	**Params**	**Type**	**Values**
	diagnosticsReportCallback	Object	Defined below
	keyValueSetCallback	Object	Defined below
Exported To	Application UI		

The signature of the callbacks are as below.

diagnosticsReportCallback is for logs

```
JavaScript
function diagnosticsReportCallback(logStatus, logData) {
 // logStatus : Additional information on the logs, can be ignored as of now.
 // logData : Troubleshooting log to save in a file..
}
```

diagnosticsKeyValueCallback is for test responses, described in the following sections.

```
JavaScript
function diagnosticsKeyValueCallback(key, status, description) {
  // key : Indicates the type of response
  // status: the value/status specific to key
  // description : description specific to key
}
```

Immediately after invoking the initDiagnostics, three parameters are returned through the diagnosticsKeyValueCallback callback.

browserVersion	browserName/browserVersion. Eg. Chrome/101.0.0.0
micInfo	Mic name returned by the browser. Eg. “Built-in Audio Analog Stereo”
speakerInfo	Speaker Name returned by the browser. Eg. “Built-in Audio Analog Stereo”

Speaker Test

Speaker test can be started by invoking the API “startSpeakerDiagnosticsTest”.

API Name	ExotelWebClient.startSpeakerDiagnosticsTest
Args	None
Exported To	Application UI

This starts a test by playing a ringtone. And as the ring tone gets played, the volume levels are passed through the callback function “diagnosticsKeyValueCallback” which can then be used to render the UI to show volume meter.

```javascript
function diagnosticsKeyValueCallback(key, status, description) {
  //key:“speaker”
  //status: a floating point value
  //description : "speaker ok"/”speaker error”
}
```

Once the user response is captured, it can be passed back to the API, “stopSpeakerDiagnosticsTest " as below. The responses are passed as arguments. THe

API Name	ExotelWebClient.stopSpeakerDiagnosticsTest
Args	Optional, if present, “yes”/”no”.
Exported To	Application UI

The response “yes” indicates that the user has heard the speaker's sound. “No” indicates that the user did not hear the speaker's sound. This response is further used to update the troubleshooting logs.

If no arguments are passed, only the test is terminated. No updates would be made to troubleshooting logs.

Mic Test

Mic test can be started by invoking the API “startMicDiagnosticsTest”.

API Name	ExotelWebClient.startMicDiagnosticsTest
Args	None
Exported To	Application UI

This starts a test by capturing the audio spoken on the mic. And as the audio is analyzed, the volume levels are passed through the callback function “diagnosticsKeyValueCallback” with key as “mic” which can then be used to render the UI to show mic volume meter.

JavaScript
```
function diagnosticsKeyValueCallback(key, status, description) {
  //key:“mic”
  //status: a floating point value
  //description : "mic ok"/”mic error”
}
```

Once the user response is captured, it can be passed back to the API, “stopMicDiagnosticsTest " as below. The responses are passed as arguments. THe

API Name	ExotelWebClient.stopMicDiagnosticsTest
Args	Optional, if present, “yes”/”no”.
Exported To	Application UI

The response “yes” indicates that the user has been captured by the mic. “No” indicates that the user's voice could not be captured. This response is further used to update the troubleshooting logs.

If no arguments are passed, only the test is terminated. No updates would be made to troubleshooting logs.

Network Diagnostics

Network diagnosis can be started by invoking the API "startNetworkDiagnostics".

API Name	ExotelWebClient.startNetworkDiagnostics
Args	None
Exported To	Application UI

This API starts network operations testing. The callback "diagnosticsKeyValueCallback" is called with appropriate keys after each test completion.

a. Web Socket Connection Callback

Returns a WSS url with status "connected" on successful connectivity.

```JavaScript
function diagnosticsKeyValueCallback(key, status, description) {
  // key:"wss"
  // status = connected/disconnected
  // description = WSS URL
}
```

b. User Registration Status Callback

Returns a status "connected" on successful registration of the configured "username". This is the callback from the background registration requests. No explicit registration requests are sent specifically for diagnostics purposes.

```JavaScript
function diagnosticsKeyValueCallback(key, status, description) {
  // key:"userReg"
  // status - "registered"/"unregistered"
  // description - userName
}
```

c. TCP connectivity callback

Returns a key value “tcp”with ice candidate information as description.

```JavaScript
function diagnosticsKeyValueCallback(key, status, description) {
    // key:“tcp”
    // status: connected/disconnected
    // description : ice candidate line for tcp connectivity/empty string
  }
```

d. UDP connectivity callback

Returns a key value “udp” with ice candidate information as description.

```JavaScript
  function diagnosticsKeyValueCallback(key, status, description) {
    // key:“udp”
    // key: connected/disconnected
    // description : ice candidate line for udp connectivity/empty string
  }
```

e. Host connectivity callback

Returns a key value “host” with ice candidate information as description for internal network.

```JavaScript
 function diagnosticsKeyValueCallback(key, status, description) {
       // key:“host”
       // key: connected/disconnected
       // description : ice candidate for the host connectivity (local
facing)/empty string
  }
```

f. Reflexive connectivity callback

Returns a key value “srflx” with ice candidate information as description for external network.

```javascript
 function diagnosticsKeyValueCallback(key, status, description) {
       // key:“srflx”
       // key: connected/disconnected
       // description : ice candidate for the reflex connectivity (remote
facing)/empty string
  }
```

5.14. Auto Reconnect

Sometimes due to network issues, websocket connections get disconnected. In that case the application has to retry the connection. To implement it we can store the state for shouldAutoRetry, and during doRegistration it could be set as true, and during explicit unregistration it could be set as false, and based on an unregistered event we can invoke doRegister API.

```javascript
       var shouldAutoRetry = false;
       function registerToggle() {
          if (document.getElementById("registerButton").innerHTML ===
       "REGISTER") {
              shouldAutoRetry = true;
              UserAgentRegistration();
          } else {
              shouldAutoRetry = false;
              exWebClient.unregister();
          }
       }
```

```javascript
       function RegisterEventCallBack(state, sipInfo) {
          document.getElementById("status").innerHTML = state;
          if (state === 'registered') {
              document.getElementById("registerButton").innerHTML =
       "UNREGISTER";
```

```
            } else {
                document.getElementById("registerButton").innerHTML = "REGISTER";
                if (shouldAutoRetry) {
                    exWebClient.DoRegister();
                }
            }
        }
```

5.15. Check SDK Readiness

- To check the SDK readiness, whether SDK is ready to receive a call or not. We can invoke checkClientStatus API with a callback method.
- First it checks if the microphone is available or not, then it checks whether the websocket is connected or not, then it checks if the user is registered or not.

Args	Datatype
clientStatusCallback	Callback function with status as String

Event	**Event Decription**
media_permission_denied	either media device not available, or permission not given
not_initialized	sdk is not initialized
websocket_connection_failed	websocket connection is failing, due to network connectivity
unregistered,terminated	either your credential is invalid or registration keepalive failed.
initial	sdk registration is progress
registered	Ready to receive the calls
unknown	something went wrong
disconnected	websocket is not connected
connecting	Trying to connect the websocket

```JavaScript
exWebClient.checkClientStatus(function (status) {
        console.log("SDK Status " + status);
 });
```

5.16. Audio Device Selection

- To get the device ID when the default device got changed, we can register callbacks
- `registerAudioDeviceChangeCallback function` Argument
    -

Argument	type	
audioInputDeviceChangeCallback	function	manadatory
audioOutputDeviceCallback	function	mandatory
onDeviceChangeCallback	function	optional

    - In case we dont pass onDeviceChangeCallback then sdk will internally try to change the default input/output device
    - If onDeviceChangeCallback is passed as third argument then sdk will not try to change the default audio/input device internally, however, OS may have change the default device at OS level.

```JavaScript
exWebClient.registerAudioDeviceChangeCallback(function (deviceId) {
    console.log(`demo:audioInputDeviceCallback device changed to
${deviceId}`);
}, function (deviceId) {
    console.log(`demo:audioOutputDeviceCallback device changed to
${deviceId}`);
});
```

- During the call or before the call, to change the audio output device
- You can optionally set the `forceDeviceChange` parameter to `true`. This action will bypass the system's internal auto-switching mechanisms.

```JavaScript
exWebClient.changeAudioOutputDevice(
        selectedDeviceId,
        () => console.log(`Output device changed successfully`),
        (error) => console.log(`Failed to change output device:
${error}`),
      true // optional
    );
```

- During the call or before the call, to change the audio input device
- You can optionally set the `forceDeviceChange` parameter to `true`. This action will bypass the system's internal auto-switching mechanisms.

```JavaScript
function changeAudioInputDevice() {
    const selectedDeviceId =
document.getElementById('inputDevices').value;
    exWebClient.changeAudioInputDevice(
        selectedDeviceId,
        () => console.log(`Input device changed successfully`),
        (error) => console.log(`Failed to change input device:
${error}`),
        true // optional
    );
}
```

- If you want the SDK to automatically detect and switch to newly plugged in audio input/output devices, you can enable this option by passing a 4th argument to `initWebrtc`.

```JavaScript
      exWebClient.initWebrtc(
        sipAccountInfo,
        RegisterEventCallBack,
        CallListenerCallback,
        SessionCallback,
        true // optional: Enables auto audio device change handling
      );
```

5.17. Logger Callback

To get the SDK logs item as a callback event we can register own logger

RegisterLoggerCallback args

Args	Datatype
type	string
message	String
args	Array

```JavaScript
exWebClient.registerLoggerCallback(function (type, message, args) {
    switch (type) {
        case "log":
            console.log(`demo: ${message}`, args);
            break;
        case "info":
            console.info(`demo: ${message}`, args);
            break;
        case "error":
            console.error(`demo: ${message}`, args);
            break;
        case "warn":
            console.warn(`demo: ${message}`, args);
            break;
```

```
        default:
            console.log(`demo: ${message}`, args);
            break;
    }
});
```

5.18. Audio Volume Control

- The SDK provides granular control over different audio elements including call audio, ringtone, ringback tone, DTMF tone, and beep tone volumes.
- Volume values are normalized between 0.0 (silent) and 1.0 (maximum)
- Volume settings persist during the session but reset when the page is reloaded

5.18.1. Notification Audio Volume Control

- Methods to control audio output volume for notification sounds.

API Name	Args	Returns	Description
ExotelWebClient .setAudioOutput Volume	- audioElementName (string) - value (number 0.0-1.0)	None	Set notification sound volume
ExotelWebClient .getAudioOutput Volume	- audioElementName (string)	Current volume (number 0.0-1.0)	Get notification sound volume

- Valid audioElementName values:
  - "ringtone" - Incoming call ringtone
  - "ringbacktone" - Outgoing call ringback tone
  - "dtmftone" - DTMF keypad tones
  - "beeptone" - System beep sounds

```javascript
// eg: Set ringtone volume to 50%
exWebClient.setAudioOutputVolume("ringtone", 0.5);

// Get current ringtone volume
const volume = exWebClient.getAudioOutputVolume("ringtone");
```

5.18.2. Call Audio Volume Control

- Methods to control audio output volume for call audio.

API Name	Args	Returns	Description
exWebClient.setCallAudioOutputVolume	- value (number 0.0-1.0)	None	Set call audio volume
exWebClient.getCallAudioOutputVolume	- None	Current volume (number 0.0-1.0)	Get call audio volume

- Valid audioElementName values:
    - "ringtone" - Incoming call ringtone
    - "ringbacktone" - Outgoing call ringback tone
    - "dtmftone" - DTMF keypad tones
    - "beeptone" - System beep sounds

```javascript
// eg: Set call audio volume to 80%
exWebClient.setCallAudioOutputVolume(0.8);

// Get current call audio volume
const callVolume = exWebClient.getCallAudioOutputVolume();
```

5.19. Disabling Built-in Logging

- The SDK provides a way to control all built-in logging (console output, SDK logs, and SIP.js logs) using the setEnableConsoleLogging method.
- Call the following method on your ExWebClient instance:

```javascript
// Disable all SDK and SIP.js logs
exWebClient.setEnableConsoleLogging(false);
```

- Default: Logging is enabled (true).
- Effect: When set to false, all SDK logs, SIP.js logs, and internal logger callbacks are suppressed.
- Note: This should be called before initializing or registering the client to ensure no logs are printed.

6. Integration with Exotel APIs

Refer to https://developer.exotel.com/api/make-a-call-api for exotel platform integration APIs. For example, to make a outbound call from a webclient to a pstn phone the request will be

```
curl -s -X POST
https://<your_api_key>:<your_api_token><subdomain>/v1/Accounts/<your_account_sid>/Calls/connect -d "From=<sip_user_id>" -d
"CallerId=<caller_id>" -d "To=<phone number>"
```

7. Support Contact

Please write to hello@exotel.in for any support required with integration.