WebSBC Client

API User Manual

FastFind Links

Sansay, Inc.
4350 La Jolla Village Dr. Ste. 888
San Diego, CA 92122

Copyright © 2018 Sansay, Inc.
All Rights Reserved

This document is specific to the WebSBC API platform. The licensed product described in this document and all licensed materials that are available for it are provided by Sansay under terms of the Sansay Software License Agreement.

Sansay periodically makes additions, deletions, or changes to the information in this document. Before you use this document, consult Sansay or your distributor for the most recent edition. The author and publisher have made reasonable efforts to ensure the accuracy and timeliness of the information in this document. However, neither the author nor the publisher shall have any liability with respect to loss or damage caused or alleged to be caused by reliance on any information in this document.

Sansay may have patents or pending patent applications covering material in this document. Furnishing this document does not of itself constitute a grant of any license or immunity under any patents, patent applications, trademarks, copyrights, or other rights of Sansay, or of any third party, or any right to refer to Sansay in any advertising or other marketing activities. Sansay assumes no responsibility for any infringement of patents or other rights of third parties that may result from use of the material in this document or for the manufacture, use, lease, or sale of machines or software programs described herein, outside of any responsibilities assumed in the original or subsequent purchase or lease agreements.

This document may contain information about, or make reference to, Sansay products, programming, or services that are not available in your country. This information must not be construed to mean that Sansay intends to make available such products, programs, or services in your country. Sansay and the Sansay logo are registered trademarks, and WebSBC is a trademark, of Sansay, Inc. All other brands, product names, trademarks, or service marks are property of their respective owners.

WebSBC Client API User Manual
Document version: 1.0
March 19, 2018

CONTENTS

1 API OVERVIEW

The Sansay WebSBC™ client is a simple JavaScript API that implements a WebRTC client engine. Objects instantiated from this class are complete WebRTC clients that work with Sansay's WebSBC server (a WebRTC gateway powered by Sansay's VSXi). WebRTC clients are objects that can be used for secure web communications between one another. They can also be coupled with Sansay's VSXi offnet routing capabilities.

A `sansayWebSBCClient` object can make and receive calls, as well as handle mid-call actions such as dual-tone multi-frequency signaling (DTMF), mute, unmute, hold, and unhold calls. In addition, it provides callback hooks that applications can use to add dynamic user interfaces (UIs) or any custom handling to various real-time events supported by the client engine.

As a first step:

- You should include the Sansay WebSBC library in your project.
- Make sure the Sansay WebSBC client is implementing a WebRTC client engine. Objects instantiated from this class are a complete WebRTC client that works with Sansay's WebSBC server (a WebRTC gateway).
- The `sansayWebSBCClient` object must be instantiated with the URL of its corresponding WebSBC server. For example:

```
client = new SansayWebSBCClient('wss://sansay-websbc.sansay.com');
```

- The `sansayWebSBCClient` object takes an optional parameter for specifying a DTMF method to use. For example:

```
client = new SansayWebSBCClient('wss://sansay-websbc.sansay.com', true);
```

  If this parameter is omitted or set to 'false', the object defaults to RFC2833. If this parameter is included (set to 'true'):

  – The object uses the INFO method for DTMF, and

  – The WebSBC should have trancoding enabled to support INFO-to-2833 conversion.

  Note that INFO from the WebRTC client supports all browsers, while 2833 might not be supported on all browsers.

1.1 Audience

This user manual is intended for software developers and application and Web development engineers who create application programs designed to integrate into or interface with the Sansay WebSBC client.

1.2 Assumptions

This user manual assumes you have a general understanding of JavaScript APIs, along with development tools and environments necessary to test and create applications.

1.3 Methods At-a-Glance

Table 1-1 summarizes the supported methods.

Table 1-1.Supported Methods

Query	Description	See Section
Registration Methods		
login	Connects the client to the WebSBC server.	2.1.1
logout	Disconnects the client from the WebSBC server.	2.1.2
Call Methods		
startRTCSession	Places an outgoing call.	2.2.1
endRTCSession	Hangs up the call	2.2.2
Mid-call Methods		
sendDTMF	Performs out-of-band DTMF dialing.	2.3.1
setHoldState	Sets a call's hold state.	2.3.2
SetMuteState	Sets a call's mute state.	2.3.3
Client Status Methods		
goBusy	Sets the client status flag to BUSY.	2.4.1
goActive	Sets the client status flag to IDLE.	2.4.2
Configuration Methods		
setStunServers	Selects a Session Traversal Utilities for NAT (STUN) server other than the default.	2.5.1
setRingTones	Adds a ring and ringback audio document object module (DOM) element to the WebSBC client.	2.5.2
setMediaElements	Allows creation of media (audio, video) DOM elements.	2.5.3

1.4 Related Documents

In addition to this user manual, you may find the following helpful:

Document/URL	Description
VSXi User Guide	Refer to this document for information about VSXi functions.
https://webphone-demo.sansay.com	Provides additional information about Sansay's WebRTC APIs, along with demos and downloadable SDKs for all Sansay WebRTC-supported features.

1.5 Document Revision Level

Revision	Date	Description
A	3/19/2018	Initial release

2 INSTANCE METHODS

Topics:

This chapter describes the instance methods.

2.1 Registration Methods

2.1.1 login()

Description

The `login()` method connects the client to the WebSBC server. It registers event handlers for the application, which the client engine will invoke for the supported events.

Syntax

```
login({
    user_id: ,
     secret:  ,
     cname:   ,
     logger:   ,
     notifier:   ,
     domain:  ,
     login_fail_cb: ,
     media:  {
                incoming_session_cb: ,
                incoming_cancel_cb:  ,
                end_session_cb:              ,
                error_cb:
                }
     })
```

Parameters

Parameter	Type	Description	Required / Optional?
user_id	Integer	ID of the user. This can be either a phone number or a SIP user ID that identifies the user in the SIP network.	Required
secret	String	Authentication password for the user to the SIP network.	Required
cname	String or Integer	Caller ID of this user that appears as the caller on the remote end. This can be a string (Joe Smith) or a phone number (8005550000).	Required
logger	Function	Callback function for third parties to add their own logging mechanism in addition to using the console port (for example, for debugging and troubleshooting).	Optional
notifier	Function	By default, alert messages are in the form of a pop-up. This function re-routes alert messages to a third party's custom notification mechanism.	Optional
domain	String	Domain to which this user belongs. This is used on the WebSBC server to determine how to route the call.	Required

Parameter	Type	Description	Required / Optional?
login_fail_cb	String	Called when login to WebSBC fails. Possible causes are invalid user_id, secret and/or domain.	Required
media	Media object	The media object contains callbacks used as hooks for real-time handling of call setup and teardown.	Required
incoming_session_cb (caller, session_id, call_type, cb)	Callback function	Called when the client receives an incoming call. Typically used to display visual notification to the user and take input from the user to determine whether to answer the call. This callback function has the function signature -function (caller, session_id, call_type, cb). The parameters of this callback carry information that the client engine (sansayWebSBCClient) passes to the application. An application can select how to use these parameters as it sees fit, except for 'cb'.	Required
caller	String	• caller = string containing the caller ID, phone number, or SIP user ID of the remote end.	Required
session_id	String	• session_id = unique string that identifies the call.	Required
call_type	String	• call_type = type of call the caller is making: – 'VIDEO' = video call. – 'AUDIO' = audio call. (*default*)	Required
cb	Boolean	• cb = function that the callback must invoke to inform the client engine how to handle the call. The cb takes a true or false parameter: – True = client call engine answers the call. – False = client call engine rejects the call.	Required
incoming_cancel_cb	Callback function	Invoked by the client engine at the completion of the rejection of an incoming call. Used to clean up the incoming call notification prompt and stop ringing. incoming_cancel_cb takes no arguments.	Required
end_session_cb	Callback function	Invoked by the client engine when either end of an inbound or outbound call hangs up. end_session_cb takes no arguments.	Required
error_cb	Callback function	Called by the client engine when it encounters an error in the signaling process. This callback function has the following parameters:	Optional
	String	• state = call state when the error occurred. Possible values are: – 'CS_ALERTING' – 'CS_OFFERED' – 'CS_ACCEPTED' – 'CS_CONNECTED' – 'CS_IDLE'	Required*
	String	• error_code = one of the following error codes: – 'NOMATCH' – 'TIMEOUT' – 'REFUSED' – 'CONFLICT' – 'DOUBLECONFLICT' – 'FAILED' * The parameters state and error_code are required if error_cb is used. For common error_cb cases, see Table 2-1.	Required*

Table 2-1. Common error_cb Cases

Case	state	error code	Mapping to SIP Code
Outbound call rejected or refused	'CS_ALERTING'	'REFUSED'	486 BUSY HERE
Call cannot be completed as dialed	'CS_DIALING'	'NOMATCH'	404 NOT FOUND
Active call fails	'CS_CONNECTED'	*	While in the 'CS_CONNECTED' state, a timeout occurs before the callee receives the acknowledgement of the caller's receipt of SIP 200, OK.

Example

```
this.login = function(user, pwd, cname, domain) {
    // connection related
    function _login_fail() {
      console.log("login fail")
      $('.webphone').css('display', 'none');
    }
    function _conn_status_change(status) {
      _vSansayClientConnUp = (status == 'connected') ? true : false;
      console.debug("conn status change: " + status);
      if (_vSansayClientConnUp) {
        clearTimeout(_vSansayClientConnTimer);
        $('.webphone').css('display', '');
      }
      else
        $('.webphone').css('display', 'none');
    }

    // call related
    function _incoming_call(caller, session_id, ctype, cb) {
      _fSansayUiAnswerPrompt(function(ans, caller, ctype) {
        if (ans) {
          _vSansayUiVideoCall = (ctype == 'VIDEO') ? 1 : 0;
          _fSansayUiCallPad(1, caller);
          _oSansayClient.goBusy();
        }
        cb(ans);
      }, caller, ctype);
    }
    function _incoming_call_cancel() {
      _fSansayUiCancelAnswerPrompt();
    }
    function _hangup() {
      if (_vSansayClientCallError)
        return;

      _fSansayUiCallPad(0);
      _oSansayClient.goActive();
    }
    function _error(state, etype) {
      var busy;
      if (state === "CS_ALERTING" && etype === "REFUSED") {
        console.debug("call is REFUSED");
        return
      }

      if (_eSansayClientAudioFastBusy != null) {
        busy = document.getElementById(_eSansayClientAudioFastBusy);
        busy.currentTime = 0;
        busy.play();
      }
      _vSansayClientCallError = true;
    }

    _oSansayClient.login({
                        user_id:        user,
                        secret:         pwd,
                        cname:          cname,
                        domain:         domain,
                        login_fail_cb:  _login_fail,
                        conn_status_cb: _conn_status_change,
                        media: {
                          incoming_session_cb: _incoming_call,
                          incoming_cancel_cb:  _incoming_call_cancel,
                          end_session_cb:      _hangup,
                          error_cb:            _error
                        }});
    _vSansayClientConnTimer = setTimeout(_login_fail, 5000);
  };
```

2.1.2 logout()

Description

The `logout()` method disconnects the client from the WebSBC server.

Syntax

```
logout( )
```

Parameters

None

Example

```
_oSansayClient.logout( );
```

2.2 Call Methods

2.2.1 startRTCSession()

Description

The `startRTCSession()` method places an outgoing call.

Syntax

```
startRTCSession( user_id )
```

Parameters

Parameter	Type	Description	Required / Optional?
user_id	Integer	User ID of the remote client. This can be either a phone number or an extension.	Required

Example

```
//Place call to callee
function _fSansayUiCallStart(callee) {
   console.log("start call");
   _oSansayClient.startRTCSession(callee);
   _oSansayClient.goBusy();
  }
```

2.2.2 endRTCSession()

Description

The `endRTCSession()` method hangs up the call.

Syntax

```
endRTCSession()
```

Parameters

None

Example

```
function _fSansayUiCallEnd() {
    if (_vSansayClientCallError) {
      _vSansayClientCallError = false;
      if (_eSansayClientAudioFastBusy != null)
        document.getElementById(_eSansayClientAudioFastBusy).pause();
    }
    else {
      console.log("end call");
      _oSansayClient.endRTCSession();
    }
    _fSansayUiCallPad(0);
  }
```

2.3 Mid-call Methods

2.3.1 sendDTMF()

Description

The `sendDTMF()` method performs out-of-band DTMF dialing.

Syntax

```
sendDTMF(character)
```

Parameters

Parameter	Type	Description	Required / Optional?
character	Char	A single character of value 0 to 9, *, or #, typical of those on a telephone keypad.	Required

Example

```
//For UI Dial Pad
function _fSansayUiSendKey(key) {
  console.log("key: " + key);
  _oSansayClient.sendDTMF(key);
}
```

2.3.2 setHoldState()

Description

The `setHoldState()` method sets the call's hold state.

Syntax

```
setHoldState(hold_state)
```

Parameters

Parameter	Type	Description	Required / Optional?
hold_state	Boolean	Determines whether the call is placed on hold. • True = place call on hold. • False = resume the call.	Required

Example

```
//Place call on hold from UI button click
function _fSansayUiSetHold(hold) {
  console.log("set hold to " + hold);
  _oSansayClient.setHoldState(hold);
}
```

2.3.3 setMuteState()

Description

The `setMuteState()` method sets the call's mute state.

Syntax

```
setMuteState(mute_state)
```

Parameters

Parameter	Type	Description	Required / Optional?
mute_state	Boolean	Determines whether the call is muted. • True = mute the call. • False = unmute the call.	Required

Example

```
//Mute the call from UI button click
function _fSansayUiSetMute(mute) {
  console.log("set mute to " + mute);
  _oSansayClient.setMuteState(mute);
}
```

2.4 Client Status Methods

2.4.1 goBusy()

Description

The `goBusy()` method sets the client status flag to BUSY.

Syntax

```
goBusy()
```

Parameters

None

Example

```
//Place call to callee
function _fSansayUiCallStart(callee) {
  console.log("start call");
  _oSansayClient.startRTCSession(callee);
  _oSansayClient.goBusy();
}
```

2.4.2 goActive()

Description

The `goActive()` method sets the client status flag to 'IDLE'.

Syntax

```
goActive()
```

Parameters

None

Example

```
//Hangup current active call and change from 'BUSY' status to 'IDLE'
function _hangup() {
  if (_vSansayClientCallError)
    return;

  _fSansayUiCallPad(0);
  _oSansayClient.goActive();
}
```

2.5 Configuration Methods

2.5.1 setStunServers()

Description

The default STUN server is the WebSBC server; however, you can select a different STUN server by calling the `setStunServers()` method. This method requires an array of JSON objects as its parameter. The JSON objects must have the field 'urls'.

Syntax

```
client.setStunServers([{urls: url1}, {urls: url2}]);
```

Parameters

Parameter	Type	Description	Required / Optional?
urls	String	URLS field within a JSON object. The JSON object must be in an array.	Required

Example

```
// external api websbc
this.init = function(server) {
  if (typeof server == "undefined" || server == null) {
    console.error("PROGRAMMING_ERROR: No WebSBC server provided");
    return;
  }

  _oSansayClient = new SansayWebSBCClient(server);
  _oSansayClient.setStunServers([{urls: 'stun:' + server}]);
  // we could use google's stun server here instead of that in websbc
  //_oSansayClient.setStunServers([{urls: 'stun:' + 'stun.l.google.com:19302'}]);

  // load the audio files
  _fSansayUiAudios();
  _oSansayClient.setRingTones('webphone-ring-tone', 'webphone-ringback-tone');
  _eSansayClientAudioFastBusy = 'webphone-busy-tone';
};
```

2.5.2 setRingTones()

Description

The `setRingTones()` method adds ring and ringback audio elements by ID from the DOM to the WebSBC client. The audio elements are invoked and played by the client engine at the appropriate state.

Syntax

```
client.setRingTones(ring_audio_dom, ringback_audio_dom)
```

Parameters

Parameter	Type	Description	Required / Optional?
ring_audio_dom	String	ID of the ring audio element in the DOM.	Required
ringback_audio_dom	String	ID of the ringback audio element in the DOM.	Required

Example

```
this.init = function(server) {
  if (typeof server == "undefined" || server == null) {
    console.error("PROGRAMMING_ERROR: No WebSBC server provided");
    return;
  }

  _oSansayClient = new SansayWebSBCClient(server);
  _oSansayClient.setStunServers([{urls: 'stun:' + server}]);
  // we could use google's stun server here instead of that in websbc
  //_oSansayClient.setStunServers([{urls: 'stun:' + 'stun.l.google.com:19302'}]);

  // load the audio files
  _fSansayUiAudios();
  _oSansayClient.setRingTones('webphone-ring-tone', 'webphone-ringback-tone');
  _eSansayClientAudioFastBusy = 'webphone-busy-tone';
};

function _fSansayUiAudios() {
    var path = "/assets/";

    var ring = document.createElement('audio');
    ring.setAttribute('id', 'webphone-ring-tone');
    ring.setAttribute('preload', true);
    ring.setAttribute('autoplay', false);
    ring.setAttribute('loop', true);
    ring.src = path + 'skype-ring.wav';
    ring.load();
    ring.pause();
    document.body.appendChild(ring);

    var ringback = document.createElement('audio');
    ringback.setAttribute('id', 'webphone-ringback-tone');
    ringback.setAttribute('preload', true);
    ringback.setAttribute('autoplay', false);
    ringback.setAttribute('loop', true);
    ringback.src = path + 'sansay-ringback.wav';
    ringback.load();
    ringback.pause();
    document.body.appendChild(ringback);

    var busy = document.createElement('audio');
    busy.setAttribute('id', 'webphone-busy-tone');
    busy.setAttribute('preload', true);
    busy.setAttribute('autoplay', false);
    busy.setAttribute('loop', true);
    busy.src = path +'sansay-fastbusy.wav';
    busy.load();
    busy.pause();
    document.body.appendChild(busy);
  }
```

2.5.3 setMediaElements()

Description

By default, the WebSBC client engine has a built-in audio DOM that handles voice-only calls without requiring an application to create media DOM elements for audio, video, or both. If an application requires a video call, it invokes the `setMediaElements()` method to inform the client engine about the DOM ID of the local and remote video DOM elements that correspond to their respective video streams.

Syntax

```
client.setMediaElements(local_video_dom_id, remote_video_dom_id);
```

Parameters

Parameter	Type	Description	Required / Optional?
local_video_dom_id	String	ID of the local video DOM element.	Required
remote_video_dom_id	String	ID of the remote video DOM element.	Required

Example

```
//In function that handles UI manipulation upon receipt of video calls
//_vSansayUiVideoCall == true if video call, false if audio only
if (_vSansayUiVideoCall) {
  _oSansayClient.setMediaElements("local-side", "remote-side");

  // reset aspect ratio
  _vSansayUiLocalAspectRatio = _vSansayUiRemoteAspectRatio = 0.0;

  // place video panel in the middle of the window
  $('#sansay-video-panel').css('top', Math.max(0, (($(window).height() - 500)/2) + $(window).scrollTop()) + "px");
  $('#sansay-video-panel').css('left', Math.max(0, (($(window).width() - 800)/2) + $(window).scrollLeft()) + "px");

  $('#sansay-video-panel').css('display', '');
}
```

Sansay, Inc.
4350 La Jolla Village Dr. Ste. 888
San Diego, CA 92122
Phone: 888.889.8906 • 858.754.1100
Fax: 858.550.2044
www.sansay.com