StarLeaf Cloud Room Systems

Endpoint Control API Guide

22 January 2019

Contents

1. Introduction to endpoint control

The Endpoint Control API is a protocol used over IP or serial connection to a StarLeaf endpoint. This protocol allows both control of the StarLeaf touchscreen and information on the state of the system. This allows implementing an external controller such as those produced by Crestron.

This API is available for all StarLeaf Cloud hardware.

To use the API, StarLeaf Support need to enable the Endpoint API Option for the endpoint to which you want to connect.

There are modules available for Crestron, Extron, and AMX that use this StarLeaf Endpoint Control API (ECAPI). To access these modules, go to: the Endpoint Control support page.

2. Connecting over IP

Each StarLeaf Cloud endpoint runs an HTTP server and supports HTTP 1.0 and 1.1.

You may prefer to use HTTP 1.1 because it allows multiple requests per connection to the server.

The HTTP connection uses port 23456.

2.1. IP authentication

To use this API over an IP connection, you must first authenticate.

To authenticate:

1. Fetch the path **`/auth`**, which returns a JSON object.
2. Extract the **`salt`** and **`iterations`** fields.
3. Convert the **`salt`** (which is a hex string) to binary.
4. Derive the key from the password along with the **`salt`** and **`iterations`**

   ```
   key = PBKDF2(HMAC-SHA256, password, salt, iterations, 256)
   ```

5. Derive the response from the derived key and the **`challenge`** field:

   ```
   response = HMAC-SHA256(key, challenge)
   ```

6. Fetch **`/auth`** again, with **`challenge`** and **`response`** as arguments.
7. If successful, the boolean field **`authenticated`** in the returned object is **`true`** and string field **`session`** is present.

Refer to Example Python key derivation script for an example of how to derive the key.

When authentication is successful and the **`session`** value obtained, it can be provided as a parameter to all future requests requiring that authentication level.

A session lasts indefinitely while in use, but times out one hour after the last request. A **`challenge`** string times out after one minute, and can only be used once (after making an incorrect response, a new challenge must be fetched to try again).

Note that the **`salt`** and **`iterations`** fields are constant for each set password, so the key need only be derived once.

2.1.1. HTTP cookies

The returned **`session`** field when authenticating is also set as a session cookie, allowing browser-style clients to make use of the authenticated state without further effort.

Fetching the path **`/auth`** with no parameters (other than the cookie) returns the above object indicating whether the user is authenticated.

2.2. Sending requests over IP

Use either POST and JSON or GET/HEAD and URI queries.

If you are connecting over IP, any request can be given the additional boolean option **`pretty`** to enable pretty-printing of the JSON output. This adds whitespace to make the output easier to read while not changing semantics.

2.2.1. Using POST and JSON queries

The request parameters are supplied in JSON format as the body of a HTTP POST request.

For example:

```
POST /action HTTP/1.0

{
  "action": "dial",
  "number": "1234"
}
```

2.2.2. Using GET/HEAD and URI queries

The request parameters are rendered inside the GET or HEAD query string.

For example:

```
HEAD /action?action=dial&number=1234 HTTP/1.0
```

Query values can only contain integers, strings and true boolean values. Integers and strings are passed directly (no quoting is required with strings). Boolean values are set by passing them without any value. Boolean values cannot be set to **`false`** through this interface – if that is required, JSON must be used instead.

```
HEAD /action?action=audio_mute&on HTTP/1.0
```

3. Connecting over serial

3.1. Serial port

The serial port on the Group Telepresence 3330 (GT Mini) is labeled **COM**. The serial port on the Group Telepresence 3351 is labeled **COM 2**.

The settings for the port are: "115200,8,N,1".

The description given here presumes that you are connecting the room system to an RS-232 DTE (a PC for example).

3.1.1. Pinout

PC end	DTE signal names	Room system end
2	RXD	3
3	TXD	2
5	GND	4
		Do not connect Pin 1

3.2. Sending requests over the serial connection

Each request consists of a JSON-formatted object, terminated by a newline. The object must contain another object called request, which is processed in exactly the same way as the content of the HTTP POST request above.

For example, to dial the number 1234 send the following:

```
{"request":{"action":"dial","number":"1234"}}\n
```

To request the current call state, send:

```
{"request":{"action":"state","filter":"calls"}}\n
```

3.3. Example responses using the serial connection

For example, if the endpoint receives the request:

```
{"id":42,"request_example":"one","request"
{"action":"dial","number":"1234"}}\n
```

It might send the response:

```
{"id":42,"request_example":"one","response":null}\n
```

Which indicates success in that the endpoint has received the request and returned a response. For actions, this is usually returned immediately.

State updates work similarly, though may be returned at some point in future. If the endpoint receives a request like:

```
{"id":17,"request_example":"two","request":{"action":"state","filter":
"calls","counter":567}}\n
```

The endpoint does not send a response until the call state updates. If an incoming call arrives, this is updated and the endpoint sends the response:

```
{"id":17,"request_example":"text","response":{"counter":578,"calls":
{"counter":91,"list":[{"id":90123,"state":3,"call_time":0,"start_time"
:0,"participants":[{"id":90124,"state":2,"name":"FooBar","number":
"1234"}]}]}}\n
```

And an example of a failing request:

```
{"id":23,"request_example":"three","request":{"action":"hold"}}\n
```

With response:

```
{"id":23,"request_example":"three","response":{"error_code":7,"error_
message":"Invalid state for action hold: no usable default call for
action."}}\n
```

3.3.1. Example serial-connection session

Below is an example session showing a request on the state of the audio and also switching the audio mute on. The example session shows you how you might use unique IDs per request. You can see that this is useful, because in the example the responses are not in the same order as the requests, but are identifiable by the ID. The example session also shows correct usage of the counter.

Send `{"id":1,"request":{"action":"state","filter":"audio"}}\n`
Receive: `{"id":1,"response":{"counter":385,"audio":{"counter":10,"mute:
false,"incall_volume":0,"ringer_volume":5,"adjunct_volume":6}}\n`
Send: `{"id":2,"request":{"action":"state","filter":"audio","counter":
385}}\n`
Send: `{"id":3,"request":{"action":"audio_mute","off":true}}\n`
Receive: `{"id": 3,"response":null}\n`
Send: `{"id":4,"request":{"action":"audio_mute"}}\n`
Receive: `{"id":4,"response":null}\n`
Receive: `{"id":2,"response":{"counter":394,"audio":{"counter":11,"mute:t
rue,"incall_volume":0,"ringer_volume":5,"adjunct_volume":6}}\n`

4. Request parameter types

- **boolean** – Boolean value (true or false).
- **integer** – 53-bit signed integer.
  Note that this need not fit in either a 32-bit (signed or unsigned) integer nor a IEEE 754 single-precision floating point value.
- **float** – Floating point value. IEEE 754 double-precision.
- **string** – Textual string.
  Non-ASCII values are UTF-8 encoded. Escape sequences for higher values (including the UTF-16 surrogate pairs \ud8XX, \udcXX) are never generated, but are accepted on input.
- **object** – Set of name/value pairs.
- **array** – Ordered list of values.
- **null** – Null value (null).

5. Responses

5.1. Response body

All response bodies contain text of type *application/json* encoded in UTF-8. For requests with nontrivial responses, this is a JSON object encoding the requested detail. For requests that do not require a response (most actions), this is the single element null. If the request was not successful, the JSON in the body is usually an object containing the two fields **`error_code`** and **`error_message`**. Some fundamental errors (malformed requests, for example) have an empty response body.

The response also echoes all elements of the request that are not otherwise used. This can be used to distinguish different requests or streams of requests that arrive out of order.

5.2. HTTP header response codes

If you are connecting over IP, the response header includes one of the following codes:

- **200 OK—Success**.
  Note that this does not necessarily mean that the action was valid and was performed, only that it was not determined to be invalid by the endpoint.
- **400 Bad Request** — Request was malformed in some way.
  Includes problems like POSTing invalid JSON and offering invalid arguments to actions.
- **405 Method Not Allowed**— Unsupported HTTP method. Only GET, HEAD, POST and OPTIONS are allowed.
- **409 Conflict** — The request is not valid in the current state.
  For example, trying to hang up a call when not in a call. This may not be returned in all possible cases, because it has to be determined by the endpoint.
- **413 Request Entity Too Large**— The POST request body was too big. The body is limited to 64 kilobytes; no valid request should need this much.
- **414 Request-URI Too Long**— The GET or HEAD query string was too long. The query string is limited to 64 kilobytes; no valid request should need this much.
- **500 Internal Server Error**— Contact StarLeaf Support with details of the request.
- **503 Service Unavailable**— There are too many outstanding requests against this server.

5.3. Example responses using IP connection

For example, if the endpoint receives the request:

```
{
"action":"dial",
"number":"1234"
}
```

It might send the response:

```
{
null
}
```

Which indicates success in that the endpoint has received the request and returned a response. For actions, this is usually returned immediately.

State updates work similarly, though may be returned at some point in future. If the endpoint receives a request like:

```
{
 "action":"state",
 "filter":"calls",
 "counter":567
}
```

The endpoint does not send a response until the call state updates. If an incoming call arrives, this is updated and the endpoint sends the response:

```
{
  "counter":578,
  "calls":{
   "counter":91,
   "list":[{
     "id":90123,
     "state":3,
     "call_time":0,
     "start_time":0,
     "participants":[{
       "id":90124,
       "state":2,
       "name":"FooBar",
       "number":"1234"
      }
     ]
    }
   ]
  }
}
```

And an example of a failing request:

```
{
 "action":"hold"
 }
```

With response:

```
{
 "error_code" : 7,
 "error_message" : "Invalid state for action hold: no usable default call
for action."
}
```

6. Actions

Actions are initiated by requests to the path **`/action`**.

The argument **`action`** is used to determine which action will be run.

6.1. Call control

6.1.1. Making a call

The action **`dial`** calls a number, URI, favorite, or scheduled conference. Exactly one of those arguments must be provided.

Parameter	Data type	Description
`number`	string	Number or URI to dial.
`favorite`	integer	Favorite ID to dial.
`recent`	integer	Recent ID to dial.
`scheduled_conference`	integer	Call the scheduled conference with the given ID.
	boolean	Call the next scheduled conference.

Examples:

- **`action?action=dial&number=1234`**
  Dial the number 1234.
- **`action?action=dial&number=name@example.com`**
  Dial the URI name@example.com.
- **`action?action=dial&number=%c3%b1%c3%a5m%c3%a9@example.com`**
  Dial the URI ñåmé@example.com.
- **`action?action=dial&favorite=567`**
  Dial the favorite with ID 567.
- **`action?action=dial&recent=487`**
  Dial the recent with ID 487.
- **`action?action=dial&scheduled_conference`**
  Dial the next scheduled conference (this implements the Join Now button).
- **`action?action=dial&scheduled_conference=890`**
  Dial the scheduled conference with ID 890.

6.1.2. Ending a call

The action **`hangup`** ends a call. If no **`callid`** is specified, it hangs up the current foreground call. (In general, unless you are sure that you have exactly one call then always specify the **`callid`** to avoid surprises.)

Parameter	Data type	Description
`callid`	integer	ID of call to hang up.

6.1.3. Responding to an incoming call

The actions **`answer`**, **`reject`** and **`ignore`** respond to an incoming call. All take zero (to respond to the first incoming call that has not already been responded to) or one (to respond to a specific call) arguments.

Parameter	Data type	Description
`callid`	integer	ID of call to respond to.

6.1.4. Hold

The actions **hold** and **resume** perform hold and resume on calls.

Parameter	Data type	Description
callid	integer	ID of call to to hold or resume.

6.1.5. Transfer

The actions **start_transfer** and **complete_transfer** perform transfer actions on calls. After the **start_transfer** action, a new call is immediately created. To proceed with the transfer, use **dial** to connect that call. Use **complete_transfer** to finish: if called immediately, the transfer is blind; otherwise, the consultation call connects and the transfer can be completed at any time thereafter. To cancel the transfer, use **hangup** on the new call (including in the case where it has not yet been dialed).

Parameter	Data type	Description
callid	integer	ID of call being acted on.

Example transfer sequence:

- **action?action=dial&number=1234**
  Dial the number 1234.
- **action?action=start_transfer**
  Start a transfer.
- **action?action=dial&number=5678**
  Dial the number 5678, to make a consultation call.
- **action?action=complete_transfer**
  Complete the transfer – both calls end and the remote ends connected to each other.

6.1.6. Conferencing

The actions **create_conference** and **add_to_conference** work together in the same way as **start_transfer** and **complete_transfer** to create a new ad hoc conference with three participants. When in a conference, **dial** and **add_to_conference** can be used to add more participants, again in the same way. A participant can be removed from the conference using **kick_participant**, and they can be muted and unmuted using **mute_participant** and **unmute_participant**.

Parameter	Data type	Description
callid	integer	ID of call or conference being acted on.
partid	integer	ID of participant in the conference to act on.

Example conference sequence:

1. **action?action=dial&number=1234**
   Dial the number 1234.
2. **action?action=create_conference**
   Promote the call to an ad hoc conference, and make a new call to add.
3. **action?action=dial&number=3456**
   Dial the number 3456, to connect the second call.
4. **action?action=add_to_conference**
   Add the second call into the conference.
5. **action?action=add_to_conference**
   Create a new call to add another participant.
6. **action?action=dial&number=5678**
   Dial the number 5678, to connect the third call.
7. **action?action=add_to_conference**
   Add the third call into the conference.
8. **action?action=mute_participant&partid=234**
   Mute the participant with ID 234 (find the ID numbers in the participants array in the call object).
9. **action?action=kick_participant&partid=456**
   Kick the participant with ID 456.
10. **action?action=hangup**
    Leave the conference; it continues with the two remaining participants.

Example conference sequence: Answer two incoming calls and escalate one of those to an ad hoc conference

1. **action?action=answer&callid=<1st call id>**
   Answer the first incoming call
2. **action?action=answer&callid=<2nd call id>**
   Answer the second incoming call, putting the first on hold
3. **action?action=create_conference&callid=<2nd call id>**
   Use the second call to create a conference (the first is still on hold)
4. **action?action=add_to_conference&callid=<1st call is>**
   Add the first call into the conference.
5. **action?action=hangup**
   Leave the conference; it continues with the two remaining participants.

6.1.7. Recording

The actions **`start_recording`** and **`stop_recording`** perform recording actions on a call. These actions always apply to the active call. There are no parameters.

Example recording sequence:

- **`action?action=start_recording`**
- **`action?action=stop_recording`**

For information about the recording state of a call, refer to the Call state section of Endpoint Control API state.

6.2. Camera control

6.2.1. Select camera

The **camera_select** action selects the active camera on systems with multiple cameras. The cameras are zero-indexed (for example, on a system supporting three cameras, the three cameras are 0, 1 and 2).

Parameter	Data type	Description
index	integer	Camera index to switch to.

6.2.2. Move camera

The **camera_control** action is used to move the local active camera or the remote camera (via Far-End Camera Control). Moving local cameras other than the active camera is not supported.

For example:

- **action?action=camera_control&preset=3**
  Move the camera to camera preset 3.

Parameter	Data type	Description
local	boolean	Move the local camera. This is the default.
remote	boolean	Move the remote camera.
stop	boolean	Stop moving the camera.
direction	string	Direction to move in: one of **up**, **down**, **left**, **right**, **zoom-in** or **zoom-out**.
duration	integer	Number of milliseconds to continue the move action for. Defaults to 100ms.
preset	integer	Move the local camera to a camera preset (see below). Note that camera presets are configured using the StarLeaf Portal.

Camera preset is one of:

Preset	Description
0	The default preset from the StarLeaf Portal.
1, 2, 3, 4	Presets 1-4 from the StarLeaf Portal.

6.3. Audio

6.3.1. Audio mute

The **audio_mute** action modifies the audio mute state of the endpoint.

Parameter	Data type	Description
on	boolean	Audio muted.
off	boolean	Audio unmuted.
toggle	boolean	Toggle. This is the default.

6.3.2. Volume

The **volume** action controls the volumes of the devices attached to the endpoint. Volume levels are integers 0 to 10 – attempts to move outside this range will saturate.

Parameter	Data type	Description
device	string	Device to modify volume on – see below.
absolute	integer	Set to the given absolute volume.
relative	integer	Modify the current volume by the (possibly negative) value given.
direction	string	up or down – equivalent to relative +1 and –1 respectively.

- **ringer** – The ringer volume.
- **adjunct** – The audio or video adjunct. With a GT Mini, this is usually the HDMI/TV output, but could equally be line-out or a Conference Phone 2220.
- **incall**– This is usually the HDMI/TV output, but could equally be the audio line-out. It affects whichever audio device is in-use for the call

If no device is specified, then the currently active audio device is used. (When out of call, this is the ringer.)

6.4. Video

6.4.1. PC sharing

The **share_pc** action changes whether the PC is being shared on the currently active call.

Parameter	Data type	Description
on	boolean	Switch on.
off	boolean	Switch off.
toggle	boolean	Toggle. This is the default.

6.4.2. Video mute

The **video_mute** action modifies the video mute state of the endpoint.

Parameter	Data type	Description
on	boolean	Switch on.
off	boolean	Switch off.
toggle	boolean	Toggle. This is the default.

6.4.3. Self view

The **self_view** action sets the self view display mode.

Parameter	Data type	Description
auto	boolean	Set to automatic mode.
off	boolean	Set to always off.
on	boolean	Set to always on.

6.4.4. Keypad

The **keypad** action sends digits as if typed on the keypad. This creates DTMF digits while in call. On a StarLeaf endpoint, this opens a new call and starts dialing when the endpoint is not currently in a call.

Parameter	Data type	Description
digits	string	Digits to dial.

6.4.5. Do not disturb

The **do_not_disturb** action sets the 'do not disturb' state of the endpoint.

Parameter	Data type	Description
on	boolean	Switch on.
off	boolean	Switch off.
toggle	boolean	Toggle. This is the default.

6.4.6. Call forwarding

The **forward_calls** action sets the endpoint to forward calls somewhere else. If the user has no mailbox or mobile number, trying to set the corresponding forwarding target silently fails.

Parameter	Data type	Description
off	boolean	Switch off.
target	string	Forwarding target: **off**, **voicemail**, **mobile** or **number**.
number	string	Number to forward to, when **target** is **number**.

6.4.7. Help

The **help** action enumerates the available actions, returning them as a JSON array.

7. State

The state of the endpoint is examined by requesting the path **/state**.

This returns a JSON object containing one or more subobjects corresponding to each section requested.

7.1. Sections

Specific sections can be requested via the **filter** parameter. This takes a comma-separated list of section names to examine, or the special value **all**, which requests all sections. If no **filter** is provided, then a default set of sections is returned (not necessarily all).

Some examples:

- **state**
  Fetch default state sections (time, locale, network, endpoint, line, calls, scheduled conferences, audio, video)
- **state?filter=line,calls**
  Fetch the state sections line and calls.
- **state?filter=all**
  Fetch all state sections.
- **state?filter=all&pretty**
  Fetch all state sections, and pretty-print the output to be more human-readable.

7.1.1. Version

The **version** section returns version information about the endpoint.

Parameter	Data type	Description
version_string	string	String form of the version number.

7.1.2. Time

The **time** section shows the current time in universal (UNIX / UTC), local (configured timezone) and system (monotonic) clocks. It pushes an update when the minute changes (sufficient for rendering a HH:MM clock).

Parameter	Data type	Description
system_clock	integer	Current system clock time (often seconds since boot).
unix_clock	integer	Current UNIX time (milliseconds since the start of 1970).
date_time	object	Current local time, made up of the following fields:
year	integer	Year.
month	integer	Month of year (one indexed).
day	integer	Day of month (one indexed).
hour	integer	Hours (24-hour).
minute	integer	Minutes
second	integer	Seconds.

Parameter	Data type	Description
dst	boolean	Whether DST is currently in effect.
day_of_week	integer	Day of week (Sunday as zero to Saturday as six).
day_of_year	integer	Day of year (zero indexed).

7.1.3. Line

The **line** section contains details about the current line.

Parameter	Data type	Description
id	integer	Line ID.
name	string	Display name.
number	string	Number.
uri	string	Global URI.
legacy_uri	string	Global URI usable with legacy H.323 devices.

7.1.4. Calls

The **calls** section contains details of all calls.

The top level object contains the element **list**, which is an array of call objects. Each call object has fields:

Parameter	Data type	Description
id	integer	Unique ID of this call (pass as **callid** to call-related actions).
state	integer	API call state.
call_time	integer	Time since the call was connected, in seconds.
participants	array	If this is a multiparty call, the array of participant details.
recording	integer	The recording state of the call. This is only present for organizations that have recording enabled.

Updates are pushed on changes to everything except the **call_time** field.

Each participant object has fields:

Parameter	Data type	Description
id	integer	Unique ID of this participant (pass as **partid** to conference-related actions).
state	integer	API participant call state.
name	string	Display name.
number	string	Number (or URI).

Basic call states are as follows:

	Call state	Description
0	INACTIVE	Inactive call (ignore, should never be seen).
1	DIALING	Call is currently dialing, or offhook about to dial.
2	WAITING	Outgoing call is placed and waiting for the far end.
3	RINGING	Incoming call is ringing.
4	IN_CALL	In call.

	Call state	Description
5	ON_HOLD	On hold, at the near end.
6	ENDED	Call ended.

Recording state of a call is as follows:

	Recording state	Description
0	UNAVAILABLE	Recording is not available for this call (for example, recording is not available for point-to-point calls).
1	DISABLED	This call is not currently being recorded.
2	ENABLED	This call is currently being recorded.

For information about the recording action, refer to the Call action section of Endpoint Control API actions.

7.1.5. Scheduled conferences

The **scheduled_conferences** section contains details of current and future scheduled conferences. The top level object contains the element **list**, which is an array of objects:

Parameter	Data type	Description
id	integer	Unique ID (pass as **scheduled_conference** to action **dial**).
state	integer	Scheduled conference state.
name	string	Display name.
start_time	integer	Conference start time, as UNIX time (may be in the past).
end_time	integer	Conference end time, as UNIX time.

Scheduled conference states are as follows:

	Schedule conference state	Description
1	NORMAL	The conference has been created, but is not in any of the other states.
2	SOON	The conference can now be called and joined. For example, on the StarLeaf touchscreen controller, the Join Now button displays green.
3	NOW	The conference start time has been reached. For example, on the StarLeaf touchscreen controller, the Join Now button flashes green. On the StarLeaf Phone 2120 and StarLeaf Touch 2020, the message indicator light flashes red. The NOW state continues for five minutes from the start of the conference
4	NOW QUIET	The conference has not reached its end time. For example, on the StarLeaf touchscreen controller, the Join Now button displays green, but does not flash.

7.1.6. Endpoint

The **endpoint** section details miscellaneous endpoint state.

Parameter	Data type	Description
id	integer	Endpoint ID.
dnd	boolean	Whether Do-Not-Disturb is enabled.
forwarding_availability	integer	Forwarding availability states are listed below.
forwarding_state	integer	Forwarding states are listed below.
forwarding_number	string	Number or URI forwarded to.
missed_calls	integer	Number of missed calls (cleared when recent calls are opened).
message_count	integer	Number of messages that have not been listened to.
standby	boolean	Whether the endpoint is currently in standby. This field is only present where a touchscreen controller is connected to a PT Monitor, PT Mini codec, or a room system codec such as Group Telepresence 3330 or 3351.
serial	string	The serial number of the touchscreen controller.

Forwarding availability value is a bitmask of the following (for example, 3 means that calls could be forwarded to either a mobile number or to another number):

	Forwarding number	Description
0	NONE	Calls cannot be forwarded.
1	MOBILE	Calls can be forwarded to a mobile number.
2	NUMBER	Calls can be forwarded to another number.

Forwarding state values are as follows:

	Forwarding state	Description
0	OFF	Calls are not forwarded.
1	VOICEMAIL	Calls are forwarded to voicemail.
2	MOBILE	Calls are forwarded to a mobile number.
3	NUMBER	Calls are forwarded to another number.

7.1.7. Audio

The **audio** section details mute state and volume levels.

Parameter	Data type	Description
mute	boolean	Whether audio mute is enabled.
ringer_volume	integer	Volume of the ringer.
adjunct_volume	integer	Volume of the adjunct processing audio.
incall_volume	integer	Volume of the HDMI or line-out audio. This is the volume of whichever audio device is in-use for the call.

7.1.8. Video

The **video** section details mute state, the state of PC sharing, and video settings.

Parameter	Data type	Description
mute	boolean	Whether video mute is enabled.
power_line_frequency	integer	Configured power line frequency, in Hz (50 or 60).
screens	integer	Number of screens attached.
pc_status	integer	PC input status (see below).
active_camera	integer	Index of the currently-active camera.
cameras	array	Array of objects each describing the state of each camera input.
status	integer	Camera input status (see below).
pc_share_status	integer	PC sharing status (see below).
self_view	integer	Self_view display mode status (see below).

PC or camera input status is as follows:

	Input status	Description
0	DISCONNECTED	Nothing is connected to the input
1	ACTIVE	Input is connected, active and working.
2	ASLEEP	Input is connected, but the source device is asleep or otherwise inactive.
3	INVALID	Input is connected and active, but the video mode being used could not be determined or is not supported.

Self-view display mode status is as follows:

	Input status	Description
0	AUTO	Self view is set to automatic mode.
1	ALWAYS OFF	Self view is set to always off.

	Input status	Description
2	ALWAYS ON	Self view is set to always on.

PC sharing status is as follows:

	Sharing status	Description
0	PC_SHARE_STATUS_NONE	There is no shared PC.
1	PC_SHARE_STATUS_LOCAL	A PC is shared at the local end of the call.
2	PC_SHARE_STATUS_REMOTE	A PC is shared from the remote end of the call

7.1.9. Favorites

The **favorites** section details calling and presence details of the first few favorite contacts.

Parameter	Data type	Description
id	integer	Favorite ID (pass as favorite to action dial).
display_name	string	Display name.
type	integer	Number type (see Contacts, under contact number types).
number	string	Number (or URI).
presence	integer	API presence state.

The presence states for favorites is as follows:

	Presence state	Description
0	UNKNOWN	Presence state is unknown.
1	AVAILABLE	The favorite is available.
2	IN CALL	The favorite is currently in a call.
3	DO NOT DISTURB	The favorite has set presence to do not disturb.
4	CALLS FORWARDED	The favorite has forwarded calls.
5	NOT AVAILABLE	The favorite is offline, or the endpoint is not connected.
6	MOBILE	The favorite is a signed in iPad user who is not currently looking at Breeze. This user can receive notifications of incoming calls and conferences.

7.1.10. Locale

The **locale** section contains the current localization settings.

Parameter	Data type	Description
lang	string	Language setting, as IETF language tag.

7.1.11. Network

The **network** section contains the current IP address and routing settings.

Parameter	Data type	Description
ip	string	IPv4 address of the endpoint.
hostname	string	Hostname of the endpoint.
netmask	string	Netmask of the local IPv4 subnet.
gateway	string	Default gateway for IPv4.
nameserver	string	IPv4 address of the DNS server.

7.2. Immediate and delayed state requests

There are two types of state request:
1. State requests that always return immediately with the current state.
2. State requests that delay feedback until the state changes from a particular point in time.

In the response to any state request, within each state section, a **`counter`** field can be found. The **`counter`** is an integer value, which identifies that particular state snapshot. The **`counter`** is incremented whenever any change to the state occurs (Note. there is no maximum value for the **`counter`** and it does not return to zero). If you include this **`counter`** in a subsequent state request, the codec only returns the state response when the **counter** has changed. (HTTP: In this case, the HTTP connection is held open until a relevant change occurs, and then the whole state as requested is returned. SERIAL: No response is given until this counter changes, at which point the whole state as requested is returned.)

Note that state can be returned as if changed without any differences being visible in the response (for example the state may change from A to B and back to A), and that some state differences do not result in a response being pushed (for example, the call timer). Additionally, every section has a local counter for that section (the field counter inside that section object), which is incremented when something in that section changes. There is no way for it to be waited-for directly, but it can be used to determine whether any detail in that section has changed between consecutive requests.

Examples:

1. Without the **`counter`** field, you immediately receive the response.

For example, request:

```
{"id":1,"request":{"action":"state","filter":"endpoint"}}
```

The response is:

```
{"id":1,"response":{"counter":33170,"endpoint":{"counter":54,"id":406,
"serial":"SLP1409040","dnd":false,"forwarding_availability"
:2,"forwarding_state":0,"missed_calls":0,"message_count"
:0,"standby":true}}}
```

2. You can use this feature to wait for an incoming call without repeatedly polling the server.
For example, request:

```
{"id":2,"request":{"action":"state","filter":"calls","counter":33170}}
```

There may be no immediate response at this point, but when an incoming call arrives, the response is:

```
{"id":2,"response":{"counter":33173,"calls":{"counter":95,"list":
[{"id":644,"state":3,"call_time":0,"start_time":0,"participants"
:[{"id":643,"state":2,"name":"John Smith","number":"john.smith
@starleaf.com"}]}]}}}
```

7.3. Requesters

Any state request can specify a requester name with the additional string argument **`requester`**. When this is made, all previous outstanding requests from the same requester are canceled and return immediately (with an error message). The intent of this feature is to allow purging of old requests from devices that know they will only ever have one outstanding state request but could be restarted for whatever reason.

If requester is not specified, then multiple outstanding requests are allowed. There is a global limit on the total number of state requests (both anonymous and named) against any API instance (currently 32) – if this is exceeded, the oldest request (the one made earliest in time) is canceled and returns an error message to the user.

8. Contacts

8.1. Directory types

Contacts are fetched by requesting the path **`/contacts`**.

There are four different contact directory types:

- **`global`** (or **`directory`**) – the organization global contact directory.
- **`personal`** – the personal contact directory of the current user.
- **`recent`** (or **`history`**) – the set of recent calls.
- **`favorite`** (or **`favourite`**)– the favorites of the current user.

The initial members of the **`favorite`** directory type are also accessible via the state section **`favorites`**, which has the additional ability to push presence updates for those members.

Parameter	Data type	Description
`type`	string	Directory type (see above).
`search`	string	Search string.
`sort`	string	Sort order (either **`first,last`** or **`last,first`**).
`start`	integer	Index in directory of first contact to fetch.
`count`	integer	Number of contacts to fetch (1 to 20).

The returned object contains the details of the view created (determined by the **`type`**, **`search`** and **`sort`** arguments to the request), along with the actual results of the search within that view (determined by the **`start`** and **`count`** arguments).

Parameter	Data type	Description
`total_entries`	integer	The total number of contacts in this view (whole directory or searched subset).
`view_sequence`	integer	Sequence number of the current transient view.
`ribbon`	array	Ribbon values (**`global`** and **`personal`** directories only)
`results`	array	Array of contact detail objects (see below).

The contact details available depend on which directory is being fetched.

The **`global`** and **`personal`** directories contain:

Parameter	Data type	Description
`index`	integer	Index in this directory. The index is 0-indexed.
`id`	integer	Contact ID (for fetching detail and to use with the **`dial`** action).
`first_name`	string	First name.
`last_name`	string	Last name.
`numbers`	array	Array of number objects.

The recent calls list entries contain:

Parameter	Data type	Description
index	integer	Index in this directory. The index is 0-indexed.
id	integer	Contact ID (for fetching detail and to use with the **dial** action).
display_name	string	Display name.
count	integer	Number of calls made e (repeated calls to the same contact are combined).
time	integer	Time of last call (UNIX time).
outcome	integer	Call outcome (see below).
duration	integer	Call duration (seconds).
number	object	A single number object.

The favorite list entries contain:

Parameter	Data type	Description
index	integer	Index in this directory. The index is 0-indexed.
id	integer	Contact ID (for fetching detail and to use with the **dial** action).
display_name	string	Display name.
presence	integer	Presence state.
evsip_presence	integer	EVSIP presence state.
number	object	A single number object.

8.2. Detailed contacts

To fetch a detailed contact, first search for the normal contact by whatever means. When the normal contact is available, fetch the detailed contact using the **id** in the short entry for that contact.

Note: The contact **id** is transient and may change and therefore you must search for the contact **id** before requesting any details.

Parameter	Data type	Description
type	string	Directory type (see above).
detail	boolean	Set to true to fetch contact detail.
contactid	integer	The ID of the contact to fetch.

Examples:

- To fetch a list of up to 20 contacts from the global directory:
  request: **contacts?type=global&pretty**
- To see more contacts, pass in the **start** argument:
  request: **contacts?type=global&start=20&pretty**

  This returns a list of up to 20 contacts from the global directory, starting at the 21st contact in the global directory as ordered by the default alphabetic ordering of last name, first name.

- To search for a particular contact:
  request: **contacts?type=global&search=John%20Doe&pretty**
  Example result:

```
{
 "view_sequence" : 3,
 "total_entries" : 1,
 "results" : [{
   "index" : 0,
   "id" : 165,
   "display_name" : "John Doe",
   "first_name" : "John",
   "last_name" : "Doe",
   "numbers" : [
   ]
  }
 ]
}
```

- An empty search string matches everything, so
  contacts?type=global&search=&pretty
  is equivalent to
  contacts?type=global&pretty

- When a search has been made, more information can be viewed on a particular contact by passing in the integer **contactid**.
  request: **contacts?type=global&detail&contactid=165&pretty**
  Example result:

```
{
 "view_sequence" : 3,
 "total_entries" : 1,
 "results" : [{
   "index" : 0,
   "id" : 165,
   "display_name" : "John Doe",
   "first_name" : "John",
   "last_name" : "Doe",
   "numbers" : [{
     "type" : 1,
     "number" : "9001"
    }
   ]
  }
 ]
}
```

8.3. Personal contacts

8.3.1. Contact number types

Contact number types that can be set are as follows:

	Contact type	Description
1	**INTERNAL**	Organization-internal cloud number.
2	**CLOUD**	Global cloud number.
3	**MOBILE**	Mobile number (PSTN).
4	**WORK**	Work number.
5	**HOME**	Home number.

8.4. Recent calls

8.4.1. Clearing recent calls

The **clear_recent** action clears the recent calls list. It takes no arguments.

9. Example Python key derivation script

This is an example of how to derive the authentication key. This example uses Python. Authentication is required where you connect to the codec over IP. This is not required for serial connections.

```
#!/usr/bin/python
# Copyright (c) StarLeaf Ltd. 2015

import hashlib
import hmac
import pbkdf2
import binascii

def endpoint_api_key(password, salt_hex, iterations, verbose=False):
  if verbose:
    print "Key is PBKDF2(HMAC-SHA256, '%s', '%s', %d, 32)." % (password,
salt_hex, iterations)

  salt = binascii.unhexlify(salt_hex)
  key = pbkdf2.PBKDF2(passphrase=password, salt=salt,
iterations=iterations,
                      digestmodule=hashlib.sha256, macmodule=hmac)
  key_hex = key.hexread(32)
  if verbose:
    print "Key is '%s'." % key_hex

  return key_hex

def endpoint_api_response(key, challenge, verbose=False):
  if verbose:
    print "Response is HMAC-SHA256('%s', '%s')." % (key, challenge)

  key_bytes = binascii.unhexlify(key)
  hash = hmac.new(key_bytes, challenge, hashlib.sha256)
  response = hash.hexdigest()
  if verbose:
    print "Response is '%s'." % response

  return response

if __name__ == '__main__':
  from optparse import OptionParser

  parser = OptionParser()

  parser.add_option("--password", dest='password', help="API password, as
set in portal.")
  parser.add_option("--salt", dest='salt', help="Salt to apply to the
password during key derivation.")
  parser.add_option("--iterations", dest='iterations', type='int',
help="Number of iterations to hash during key derivation.")
  parser.add_option("--key", dest='key', help="Key derived from
password.")
```

```
  parser.add_option("--challenge", dest='challenge', help="Challenge
returned by server.")

  parser.add_option("--mode", type='choice', choices=['key', 'respond'],
dest='mode', default='respond',
                    help="Mode: key (derive key from password) or respond
(build response to challenge).")
  parser.add_option("--format", type='choice', choices=['text', 'query',
'json'], dest='format', default='text',
                    help="Output format for response: (text (plain text),
query (HTTP URI query format) or json (JSON object)).")

  parser.add_option("--verbose", action='store_true', dest='verbose',
help="Print details of intermediate steps.")

  (opt, args) = parser.parse_args()

  if opt.key:
    key = opt.key
  else:
    key = endpoint_api_key(opt.password, opt.salt, opt.iterations,
opt.verbose)

  if opt.mode == 'key':
    print key

  elif opt.mode == 'respond':

    response = endpoint_api_response(key, opt.challenge, opt.verbose)

    if opt.format == 'query':
      print "challenge=%s&response=%s" % (opt.challenge, response)
    elif opt.format == 'json':
      print "{\n  \"challenge\": \"%s\",\n  \"response\": \"%s\"\n}" %
(opt.challenge, response)
    else:
      print opt.response

  else:
    print "Invalid mode '%s'." % opt.mode
```

10. Legal information

10.1. Third party software acknowledgments

Acknowledgments of third-party software are available at:

www.starleaf.com/support/legal

10.2. Disclaimers and notices

This guide may not be copied, photocopied, translated, reproduced, or converted into any electronic or machine-readable form in whole or in part without prior written approval of StarLeaf Limited.

StarLeaf Limited reserves the right to revise this documentation and to make changes in content from time to time without obligation on the part of StarLeaf Limited to provide notification of such revision or change.

StarLeaf Limited provides this documentation without warranty, term, or condition of any kind, either implied or expressed, including, but not limited to, the implied warranties, terms or conditions of merchantability, satisfactory quality, and fitness for a particular purpose. StarLeaf Limited may make improvements or changes to the product(s) and/or the program(s) described in this documentation at any time. All other product and company names herein may be trademarks of their respective owners.