3rd Party API for Sennheiser Products

PDF export of the original HTML instructions

SENNHEISER

v3.0 | 06/2024

Contents

1. Preface

PDF export of the original HTML instructions

This PDF document is an automated export of an interactive set of HTML instructions. It may be the case that not all contents and interactive elements are contained in the PDF as they cannot be presented in this format. Furthermore, automatically generated page breaks may cause coherent contents to be moved slightly. We can therefore only guarantee the completeness of the information in the HTML instructions, and recommend that you use these. You can find these in the download section of the website under www.sennheiser.com/download.

2. Release Notes

Latest release information on the firmware and OpenAPI versions.

New Release

- 24-07-03 | The first **TC Bar Open API 1.9** is available.

Features:

- The TeamConnect Bar S and M are the first audio and video devices which support the Sennheiser Sound Control Protocol v2. The protocol allows to configure and monitor the devices using REST API calls.

Previous Releases

- 23-07-01 | The first **TCC M Open API 1.6** is available.
- 23-07-01 | The TCC M firmware version 1.2.x is supporting the API 1.6.
- 22-05-05 | MobileConnect now provides plug-ins for **Crestron and Extron 3rd party integrations**, based on the MobileConnect API.

3. Sennheiser Sound Control Protocol (SSC)

All product specific HTTPs methods, parameters and responses at a glance.

i

Sennheiser offers two different protocols that can be used depending on the functional scope of the implemented device firmware and software supplied with the product:

- **SSCv2:** New protocol with a high security standard for Sennheiser devices that are delivered with a password.
- **SSCv1:** Old protocol with command and control functions without support for networked audio streaming.

SSCv2

The newest Sennheiser 3rd party API protocol allows to configure and monitor devices using REST API calls. The following Sennheiser devices are supported:

- TeamConnect Ceiling Medium

SSC

The older protocol allows to command and to control devices. This protocol does not support the network audio streaming. The following Sennheiser devices are supported:

- SL Rack Receiver
- CHG 4N - network-enabled charger
- CHG 2N - 2 bay network charger
- Multi-channel receiver (SL MCR2 & MCR4)

- EW-DX EM 2 rack receiver (EW-DX EM2)
- EW-DX EM 2 rack receiver Dante (EW-DX EM2 Dante)
- CHG70N - 2 bay network charger

- TeamConnect Ceiling 2 (TCC 2)

Related information

Sennheiser Sound Control Protocol v2 (SSCv2)

Sennheiser Sound Control Protocol v1

Sennheiser Sound Control Protocol v2 (SSCv2)

The newest Sennheiser 3rd party API protocol called SSCv2 allows to configure and monitor the Sennheiser devices using REST API calls. Here you can find the introduction to the REST protocol, how to enable 3rd party access and how to subscribe for device configuration change.

Media Control Protocol

Document version: 0.1 01.03.2023

Sennheiser electronic GmbH & Co. KG Am Labor 1, 30900 Wedemark, Germany, www.sennheiser.com

This document introduces the Sennheiser Sound Control Protocol v2 (SSCv2) for Sennheiser devices.

List of compatible devices:

- TeamConnect Ceiling Mic M

The Sennheiser Sound Control Protocol v2 is a secure RESTful API via HTTPS transport, suitable for command, control and monitoring of networked audio devices. It uses JavaScript Object Notation (JSON) for data serialization.

Terminology

SSCv2 server: Sennheiser device or application that receives and replies to SSCv2 messages.

SSCv2 client: third party device, application or person that sends SSCv2 messages to a Sennheiser device.

Protocol Basics

This section describes the key elements of the SSCv2 protocol.

HTTP(S)

The following optional parts of HTTP(S) have been made mandatory or are recommended for SSCv2:

- An SSCv2 server must implement HTTPS using HTTP/1.1
- An SSCv2 client should use persistent connections

OpenAPI

The SSCv2 definitions are provided as an OpenAPI specification for each Sennheiser device.

Messages

SSCv2 uses two modes of message exchange:

- Synchronous: The client sends a GET/PUT/POST/DELETE request and the server immediately sends a response
- Asynchronous: The client subscribes to parameter changes and the server notifies the client of parameter changes via a Server Sent Event (SSE). For more see SSCv2 Subscriptions.

Usage

This section describes how to use SSCv2.

Enabling Third Party Access

The Sennheiser device cannot be accessed via the API in factory default state. In order to enable it:

- Connect the Sennheiser device to Sennheiser Control Cockpit.
- Go to the device page.
- Enable third party access and configure a third party password.

Authentication

It is mandatory to authenticate on the device with each request:

- Using HTTP basic authentication
- Username: api
- Password: configured using Sennheiser Control Cockpit

Connecting to a Device

Use the device IP address and the HTTPS port 443 to form the base URL for the connection requests, e.g. "url: https://192.168.0.1:443".

SSCv2 Specifications

This section explains how HTTPS, JSON and SSE are used in the context of SSCv2.

Case sensitivity

- The path component of a URI must be case sensitive, as per RFC 3986, 6.2.2.1.
- The JSON payload must be interpreted by SSCv2 clients and SSCv2 servers as case sensitive.

Default Returns

For some generic actions, mandatory HTTP status codes have been defined:

- When a client sends a request without being authenticated, the SSCv2 server replies with 401 – "Unauthorized".
- When a client is authenticated but not authorized to access the resource, the SSCv2 server replies with 403 – "Forbidden".
- When a client is authorized and tries to access a resource that does not exist, the SSCv2 server replies with 404 – "Not Found".
- When a client is authorized to access a resource but the HTTP request method is not allowed by the SSCv2 server, the SSCv2 server replies with 405 – "Method not allowed".
- When a client is authorized to access a resource but has a format error in its request, the SSCv2 server replies with 400 – "Bad request".
- When a client is authorized to access a resource and the format is correct, but the device cannot honor the request because of the internal device status, the SSCv2 server replies with 409 – "Conflict". An example for this would be trying to configure an IP address while the device is using DHCP, without also switching to fixed addresses.
- When a client is authorized to access a resource and the format is correct, but there are semantic/logical errors, the SSCv2 server replies with 422 – "Unprocessable Entity". An example for a logical error would be using a subscription ID which does not exist.

JSON

The following optional parts of JSON have been made mandatory for SSCv2:

- An SSCv2 server returns all text-based entities as a JSON object

SSCv2 Subscriptions

The SSCv2 API allows third party clients to subscribe to parameter changes. In response, the server notifies the client of parameter changes via a Server Sent Event (SSE), formatting messages according to the EventSource specification.

Subscription Message Format

For the EventSource specification only the parameters `data` and `event` are used. For the parameter `event` only the following events are used:

- `open`, at the start of a subscription.
- `message`, when sending resource updates. As it is there by default, it MAY be omitted in the event stream.
- `close`, when the SSCv2 client actively closes the subscription by sending DELETE to `/api/ssc/state/subscriptions/{sessionUUID}` or when the subscription is closed because the device is initiating a reboot.

A full EventSource-compatible update looks as follows:

```
event: <eventtype>\ndata: <updatejson>\n\n
```

where `\n` denotes the ASCII character for a linefeed.

Starting a Subscription

To initiate a subscription:

- The subscription is initialized by the SSCv2 client by issuing a GET request to `/api/ssc/state/subscriptions`
- The SSCv2 server replies to the subscription request immediately either by acknowledging the request, or by sending an error reply.
    - The SSCv2 server sends an initial message to the client, containing:
        - The `Content-Type` set to `text/event-stream`
        - The `Content-Location` containing the path `/api/ssc/state/subscriptions/{sessionUUID}`
        - where `SessionUUID` is the ID generated for the subscription and associated with the HTTP session. It can be used later to modify the subscription.
        - The initial `open` event, with the following data:

```
{
  "path": "/api/ssc/state/subscriptions/{sessionUUID}",
  "sessionUUID": "{sessionUUID}"
}
```

- The SSCv2 subscription is initially empty without any subscribed resources, and the SSCv2 client must subscribe to resources using `/api/ssc/state/subscriptions/{sessionUUID}` and `/api/ssc/state/subscriptions/{sessionUUID}/add`, respectively.
- An SSCv2 client must not send further regular HTTP requests to the SSCv2 server via the connection used to request the subscription. After the initial subscription request the connection can only be used to receive events from the SSCv2 server.

The resource for starting subscription is provided in the OpenAPI snippet below.

```
/api/ssc/state/subscriptions:
  get:
    summary: Start a subscription
    description: An SSCv2 Command to start a subscription
    responses:
      '200':
        description: Successful request
        headers:
          Content-Location:
            description: Location of the resource to be used to
 change the subscribed resources
            schema:
              type: string
        content:
          text/event-stream:
```

```
            schema:
              type: string
              description: The initial subscription stream.
      '403':
        description: User is not authorized to subscribe to
resources
      '500':
        description: SSCv2 Server encountered internal error
```

Getting Subscription Status

The SSCv2 client can request the subscription status:

- Send a GET request to `/api/ssc/state/subscriptions/{sessionUUID}` using the previously received `sessionUUID`.
- If the `sessionUUID` is incorrect, an error is returned.

OpenAPI snippet:

```
/api/ssc/state/subscriptions/{sessionUUID}:
  get:
    summary: Get the subscription list
    description: An SSCv2 Command to retrieve the list of
 subscriptions associated with the sessionUUID
    parameters:
        - in: path
          name: sessionUUID
          schema:
            type: string
          required: true
    responses:
      '200':
        description: Successful request
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
                example: /api/device/site
      '403':
        description: User is not allowed to get subscription list
 for this sessionUUID
      '422':
        description: sessionUUID did not exist
      '500':
        description: SSCv2 Server encountered internal error
```

Changing Subscribed Resources

It is possible to either change the full set of resources of a subscription, or to add/remove resources from the currently subscribed set.

The SSCv2 client can only subscribe to resources specifying a GET request, otherwise an error is returned.

Changing the Full Set of Subscribed Resources

The SSCv2 client can change the full set of subscribed resources:

- Send a PUT request to `/api/ssc/state/subscriptions/{sessionUUID}` using the previously received `sessionUUID`.
    - The SSCv2 client must send a complete list of the resources to which it wants to subscribe.
- After accepting and setting up the subscription, the SSCv2 server sends the current state of each subscribed resource which is new or has been removed.

OpenAPI snippet

```
/api/ssc/state/subscriptions/{sessionUUID}:
  put:
    summary: Set/change the subscription list
    description: An SSCv2 Command to set/change the list of
 subscriptions associated with the sessionUUID
    parameters:
        - in: path
          name: sessionUUID
          schema:
            type: string
          required: true
    requestBody:
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
                example: /api/device/site
    responses:
      '200':
        description: Successful request
      '400':
        description: The request for subscription was invalid
        content:
            application/json:
```

```
              schema:
                type: object
                properties:
                  path:
                    type: string
                    example: /api/ssc/version
                  error:
                    type: integer
                    example: 403
      '403':
        description: User is not allowed to change subscription
list for this sessionUUID
      '422':
        description: sessionUUID did not exist
      '500':
        description: SSCv2 Server encountered internal error
```

Adding Resources to Subscriptions

The SSCv2 client can add one or more resources to an existing list of subscribed resources:

- Send a PUT request to `/api/ssc/state/subscriptions/{sessionUUID}/add` using the previously received `sessionUUID`.
- The SSCv2 server adds the resources to the list of subscribed resources for the existing subscription.
- If even one resource in the set of resources that the SSCv2 client wants to add is not allowed, the SSCv2 server refuses the entire request and the current set of subscribed resources for the subscription is not changed.
- The SSCv2 server treats an empty resource list as no action and replies with 200 – OK.
- The SSCv2 server reports the initial values of the newly added resources using the existing subscription.

OpenAPI snippet

```
/api/ssc/state/subscriptions/{sessionUUID}/add:
  put:
    summary: Add resource(s) the subscription list
    description: An SSCv2 Command to add a set of resources to the
 list of subscriptions associated with the sessionUUID
    parameters:
        - in: path
          name: sessionUUID
          schema:
            type: string
          required: true
    requestBody:
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
                example: /api/device/site
    responses:
      '200':
        description: Successful request
      '400':
        description: The request to add to the subscription was
 invalid
        content:
            application/json:
              schema:
                type: object
                properties:
```

```
                path:
                  type: string
                  example: /api/ssc/version
                error:
                  type: integer
                  example: 403
      '403':
        description: User is not allowed to add to the
subscription list for this sessionUUID
      '422':
        description: sessionUUID did not exist
      '500':
        description: SSCv2 Server encountered internal error
```

Removing Resources from Subscriptions

The SSCv2 client can remove one or more resources from an existing list of subscribed resources:

- Send a PUT request to `/api/ssc/state/subscriptions/{sessionUUID}/remove` using the previously received `sessionUUID`.
- The SSCv2 server removes the resources from the list of subscribed resources for the existing subscription.
- If even one resource in the set of resources that the SSCv2 client wants to remove is not allowed, the SSCv2 server refuses the entire request and the current set of subscribed resources for the subscription is not changed.
- The SSCv2 server treats an empty resource list as no action and replies with 200 – OK.
- The SSCv2 server does not terminate the subscription if the list of subscribed resources is empty after the removal of resources.
- For every removed resource the SSCv2 server sends an update, using the resource as the key and the JSON keyword "null" as the value.

OpenAPI snippet

```
/api/ssc/state/subscriptions/{sessionUUID}/remove:
  put:
    summary: Remove resource(s) from the subscription list
    description: An SSCv2 Command to remove a set of resources
 from the list of subscriptions associated with the sessionUUID
    parameters:
        - in: path
          name: sessionUUID
          schema:
            type: string
          required: true
    requestBody:
        content:
          application/json:
            schema:
              type: array
              items:
                type: string
                example: /api/device/site
    responses:
      '200':
        description: Successful request
      '400':
        description: The request to remove to the subscription was
 invalid
        content:
```

```
            application/json:
              schema:
                type: object
                properties:
                  path:
                    type: string
                    example: /api/ssc/version
                  error:
                    type: integer
                    example: 404
      '403':
        description: User is not allowed to remove from the
subscription list for this sessionUUID
      '422':
        description: sessionUUID did not exist
      '500':
        description: SSCv2 Server encountered internal error
```

Canceling a Subscription

An SSCv2 client can end a subscription either implicitly or explicitly.

- To end a subscription implicitly, the SSCv2 client simply closes the connection that receives the subscription data.
- To end a subscription explicitly, an SSCv2 client sends a DELETE request to the resource `/api/ssc/state/subscriptions/{sessionUUID}`.
    - The SSCv2 server sends a `close` event to the subscription before it closes the connection.
    - The data sent in the `close` event is the same as in the `open` event.

The SSCv2 server terminates a subscription in the following cases:

- The subscribed client cancels the subscription explicitly.
- The SSCv2 client closes the connection.
- The transport layer of the SSCv2 connection signals a fatal communication error.

OpenAPI snippet

```
/api/ssc/state/subscriptions/{sessionUUID}:
  delete:
    summary: End an existing subscription
    description: An SSCv2 Command to end the subscription
 associated with the sessionUUID
    parameters:
        - in: path
          name: sessionUUID
          schema:
            type: string
          required: true
    responses:
      '200':
        description: Successful request
      '403':
        description: User is not allowed to end subscription for
 this sessionUUID
      '422':
        description: sessionUUID did not exist
      '500':
        description: SSCv2 Server encountered internal error
```

Subscribing to Multiple Addresses

The SSCv2 client may request to subscribe to multiple resources in a single request.

- If the SSCv2 server is not able to successfully subscribe to even one of the resources, it sends an error and refuses all resources.
- If there are already subscribed resources present, the previous state of the subscription remains unchanged.

Error Handling

When the SSCv2 server refuses a request for a subscription or a set of subscriptions, it returns an object containing the first resource which was not subscribable and the reason for the error.

- 403 if subscribing to the resource is not allowed.
- 404 if the resource does not exist.

Subscription Notification Syntax

A subscription notification constitutes Server Sent Event (SSE) data in the form of a JSON object, using the resource that triggered the notification as the key and the entities of the resource wrapped in a JSON object as the value.

- An SSCv2 server may combine notifications for multiple resources in one JSON object, if the updates occur at the same time.
- Since a newly created subscription of multiple addresses must report the values of the resources, this may be combined into a larger JSON object.
- If this combination of the different resources is not supported, all notifications must be sent in the stream as individual JSON objects.

The following example shows a stream containing a notification after an initial subscription and a later notification of a name change:

```
{
 "/api/device/site":
  {
  "name": "MyDevice",
  "location": "Chemistry Building 5, Room 15",
  "site": "Left side near the big pillar"
 }
}
{
  "/api/device/site":
  {
    "name": "MyRenamedDevice",
    "location": "Chemistry Building 5, Room 15",
    "site": "Left side near the big pillar"
 }
}
```

The address `/api/device/site` was subscribed, the initial values were reported. After some time the `name` was changed and the resulting update was sent as a second JSON object.

Sennheiser Sound Control Protocol v1

This protocol allows to command and to control the Sennheiser devices supporting the older SSC v1 version.

Below you will find a list of Sennheiser products that support the SSCv1 protocol. Click on the provided link to display the appropriate protocol for the desired Sennheiser product.

> **i** Note that the protocol is intended for command and control. Network audio streaming is entirely out of its scope.

SpeechLine Digital Wireless

SSC Protocol for SpeechLine Digital Wireless devices:

- SL Rack Receiver
- CHG 4N - network-enabled charger
- CHG 2N - 2 bay network charger
- Multi-channel receiver (SL MCR2 & MCR4)

Evolution Wireless Digital

SSC Protocol for Evolution Wireless Digital devices:

- EW-DX EM 2 rack receiver (EW-DX EM2)
- EW-DX EM 2 rack receiver Dante (EW-DX EM2 Dante)
- CHG70N - 2 bay network charger

TeamConnect

SSC Protocol for TeamConnect devices:

- TeamConnect Ceiling 2 (TCC 2)

4. Open API

Here you can find all product specific HTTPs methods, parameters and responses at a glance.

Currently the REST API is supported by:

- TeamConnect Bar S and M
- Team Connect Ceiling Mic Medium
- MobileConnect

> **i** Any future supported Sennheiser products will be added to this page. Please note that Sennheiser products that are not found on this page, support the Sennheiser Sound Control Protocol v1, and you can find their specifications on their respective here.

TeamConnect Bar
Known Issues
TC Bar Open API 1.9

TeamConnect Ceiling Medium
TCC M Open API 1.6
Subscription for real-time updates

MobileConnect
MobileConnect

TeamConnect Bar (TC Bar S/M)

Known Issues

Here you can find all known issues regarding the methods, parameters and responses for TC Bar S/M.

General

Please note the following information:

- The request for PoE function `/api/device/power/poe/output` is not avaibale for TC Bar S.
- There is no **PUT** method for the request `/api/interfaces/network/dante`. You can find the **GET** method under **Device**.
- The **PUT** method for the request `/api/video/output/hdmi` (listed under **Video**) is out of function.
- The **PUT** request for activating the camera `/api/video/input/internalCamera/enable` is deprecated.
- There is no PUT method available for the request `/api/video/input/internalCamera/videoParameters`.

Activating Bluetooth®

To activate the Bluetooth® function via the **PUT** request `/api/interfaces/bluetooth`, please only send the pairing request after 10 seconds. Requesting both values at the same time does not work.

▷ Send the first request with:

```
{
  "enabled": true
  }
```

▷ After 10 sec send the second request:

```
{
  "pairing": false
  }
```

TC Bar Open API 1.9

All TeamConnect Bar HTTPs methods, parameters and responses at a glance.

TeamConnect Ceiling Medium (TCC M)

TCC M Open API 1.6

All TeamConnect Ceiling Medium HTTPs methods, parameters and responses at a glance.

Subscription for real-time updates

This script sets up a subscription to receive real-time updates from a Sennheiser device.

The
script uses HTTP Basic Authentication to authenticate with the device and requests an event stream fro

Once the stream is established, it adds a subscription for a specific request and listens for incoming d
The incoming data is printed to the console as a string representation.
If the stream fails to establish the reason for the failure is printed to the console.

To set up a subscription for real-time updates:

- ▷ Replace the `ip` variable with the IP address of your Sennheiser device.
- ▷ Replace the `request` variable with the API request you want to subscribe to.
- ▷ Replace the `password` variable with the password for your Sennheiser Smart Control device.

```
Sennheiser electronic GmbH & Co. KG
Version: 1.0.0
Date: 2023-09-18
"""

# Import necessary libraries
from requests.auth import HTTPBasicAuth
import requests

# Set up variables for authentication and API requests
ip = '192.168.42.100' # Replace with the IP address of your Sennheiser device
request = '/api/audio/inputs/microphone/beam/direction'

# Replace with the API request you want to subscribe to
password = 'the_future_of_audio' # Replace with the password for your Sennheiser Smart Control device

# Set up HTTP Basic Authentication and request an event stream from the device's API
auth = HTTPBasicAuth(
'api',
password
)
stream = requests.get(
f'https://{ip}:443/api/ssc/state/subscriptions',
auth=auth, stream=True,
headers={'Accept': 'text/event-stream'},
verify=False
)

# If the stream is established, add a subscription for the specified request and listen for incoming data
if stream.status_code == 200:
stream.uuid = stream.headers['Content-Location'].split('/')[-1]
requests.put(
f'https://{ip}:443/api/ssc/state/subscriptions/{stream.uuid}/add',
json=[request], auth=auth,
verify=False
```

```
)

while True:
data = stream.raw._fp.fp.read1(1024)
data = data.decode('utf-8')
print(repr(data))
# If the stream fails to establish, print the reason for the failure to the console

else:
print(stream.reason)
```

MobileConnect