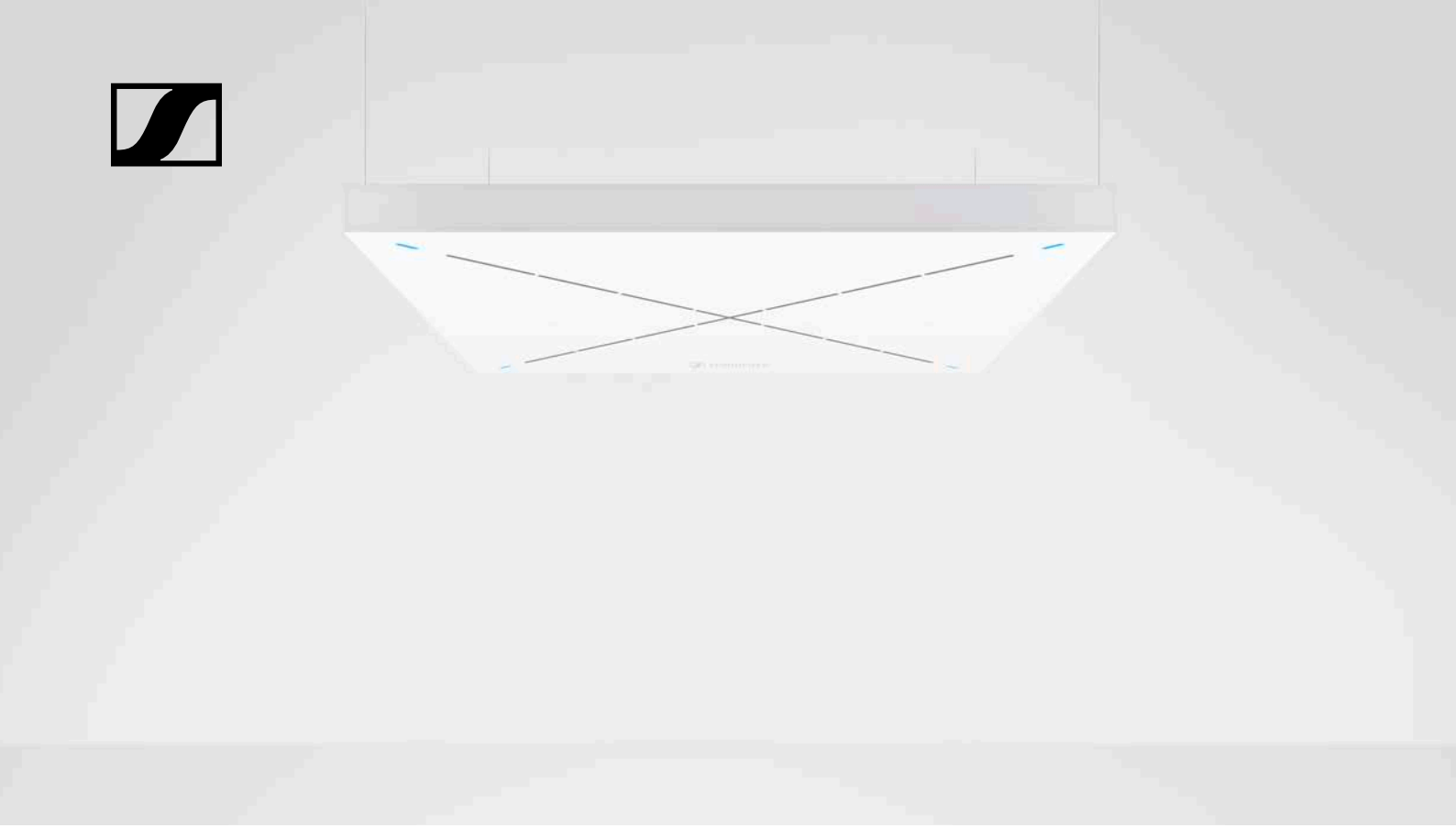

Sennheiser Sound Control Protocol (SSC)

Media control protocol

TeamConnect Ceiling 2

Firmware Version 1.8.0

Sennheiser electronic GmbH & Co. KG

Am Labor 1, 30900 Wedemark, Germany, www.sennheiser.com
TI 1245 v1.8.0

SENNHEISER

Table of Contents

1. Introduction

Modern professional audio devices are designed as building blocks for large, complex systems.

Whereas audio signal paths have converged to industry standards a long time ago, driven by practical necessities, and only recently challenged by new transport technologies like Ethernet, the professional audio markets have not evolved a similar technological convergence in the area of remote, centralised control of systems of audio equipment (the notable historical exception being MIDI, which but has a limited scope and extensibility).

In this heterogeneous environment of diverging standards proposed by individual vendors as well as open communities, there is no existing self-evident solution to be found for the needs raised by designing professional Sennheiser audio equipment.

As a consequence, communication protocols implemented in Sennheiser products have so far been designed on a single-product or product-family basis. This has worked sufficiently well, up to the point that separately developed protocols start to concur in nexus devices or applications, like:

- Wireless Systems Manager (PC-based control application for wireless transmission)
- remote channel for future Sennheiser PRO microphones
- Media Control Systems (third party products, e.g., Crestron)
- A/V studio integration (third party products, e.g., Lawo)
- smartphone or tablet apps
- future centralised Sennheiser services

It has become evident that product-specific protocols fail to scale well in nexus products because of the added complexity in re-implementing the same remote control functionality from a customer point of view in a multitude of different backwards-compatible ways. It is not feasible to add more ever different technical solutions to the existing variety - the aim must be to define a reasonably future-proof protocol suitable for existing as well as envisioned products, devices, and services.

A broad market evaluation of existing technical solutions was performed in a joint Sennheiser PRO/IS working group. As a result, it turns out that Open Sound Control comes closest to the specific needs for an extensible, future-proof command, control, metering, and configuration protocol for Sennheiser products.

This document describes the specific adaption of Open Sound Control to Sennheiser use, "Sennheiser Sound Control", SSC. The main other ingredient is JavaScript Object Notation (JSON), which enhances ease-of-use and the implementation complexity for small to smallest devices.

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

2. Open Sound Control Overview

Open Sound Control (OSC) is a protocol developed at The Center For New Music and Audio Technology (CNMAT) at University of California, Berkeley.

The OSC specification Version 1.1 is available from the Open Sound Control website at

http://www.opensoundcontrol.org/.

It is a very simple and very extensible protocol that can be implemented easily in embedded systems. It can be transported over IPv4 and IPv6 protocols using UDP packets and TCP streams.

Even very small PIC microcontrollers can handle OSC messages via projects such as MicroOSC from http://cnmat.berkeley.edu/research/uosc.

The OSC Schema defined by MicroOSC at:

http://cnmat.berkeley.edu/library/uosc_project_documentation/osc_address_schema was used as a starting point for some parts of the schema defined in this document.

OSC handles more advanced packet formats such as bundles of messages to be atomically executed at the same time with timestamps, as well as addresses with wildcards and array values.

2.1 JavaScript Object Notation Overview

JavaScript Object Notation (JSON) is a lightweight data-interchange format. It is easy for humans to read and write. It is easy for machines to parse and generate. It is based on a subset of the JavaScript Programming Language, Standard ECMA-262 3rd Edition - December 1999. JSON is a text format that is completely language independent but uses conventions that are familiar to programmers of the C-family of languages, including C, C++, C#, Java, JavaScript, Ruby, Python, and many others. These properties make JSON an ideal data-interchange language.

The central website for JSON information is http://json.org. JSON is formally specified in RFC 4627 (http://www.ietf.org/rfc/rfc4627.txt):

JavaScript Object Notation (JSON) is a text format for the serialization of structured data. It is derived from the object literals of JavaScript, as defined in the ECMAScript Programming Language Standard, Third Edition.

JSON can represent four primitive types (strings, numbers, booleans, and null) and two structured types (objects and arrays).

A string is a sequence of zero or more Unicode characters.

An object is an unordered collection of zero or more name/value pairs, where a name is a string and a value is a string, number, boolean, null, object, or array.

An array is an ordered sequence of zero or more values.

The terms "object" and "array" come from the conventions of JavaScript.

JSON's design goals were for it to be minimal, portable, textual, and a subset of JavaScript.

The MIME media type for JSON text is application/json.

3. Conventions

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP14/RFC 2119, "Key words for use in RFCs to Indicate Requirement Levels".

3.1 Terminology

SSC Message	protocol unit of transmission
SSC Server	device or application that receives SSC messages, and replies to them
SSC Client	device, application or person that sends SSC messages
SSC Container	named entity containing SSC Methods or other Containers
SSC Method	named attribute or action callable on a SSC Server
SSC Address	full name of a SSC Method, including names of all enclosing Containers may be represented by a JSON object hierarchy
SSC Address Tree	a JSON object hierarchy consisting of one or more SSC Addresses
SSC Address Space	hierarchical tree comprising all the SSC Addresses of a SSC Server
SSC Method Call	SSC Message requesting execution of a SSC Method
SSC Method Arguments	arguments included in a SSC Method Call
SSC Method Reply	SSC Message send by SSC Server as result of a Method Call
binary OSC	the binary OSC encoding as opposed to JSON-based SSC
restricted SSC Server	a SSC Server that doesn‘t implement some optional parts of this specification

4. SSC Data Structure Specification

4.1 Applying JSON to the OSC device model

OSC models the controlled device as a tree-shaped hierarchy of *methods*, with the method *addresses* constructed from the names of all the nodes in the hierarchy, written like a file path.

```
/                           container at address "/"
out1/                       container at address "/out1/"
        xlr1/               container at address "/out1/xlr1/"
                gain 5      address "/out1/xlr1/gain": method with numeric argu-
                            ment
                mute true   address "/out1/xlr1/mute": method with a boolean
                            argument
                ...         more methods of  "/out1/xlr1"
        xlr2/               container at address "/out1/xlr2/"
        ...                 methods of "/out1/xlr2"
out2/                       container at address "/out2/"
        ...                 methods of "/out2"
...                         more methods and containers of  "/"
```

JSON allows to model that structure as a hierarchy of *JSON objects*.

```
{                               root object
"out1": {                       object "out1"
        "xlr1": {               object "out1.xlr1"
                "gain": 5,      numerical property "out1.xlr1.gain"
                "mute": true,   boolean property  "out1.xlr1.mute"
                ...             more properties of  "out1.xlr1"
        },
                "xlr2": {       object "out1.xlr2"
                ...             properties of "out1.xlr2"
        },
"out2": {                       object "out2"
        ...                     properties of "out2"
},
...                             more properties and objects of the root object
}
```

The OSC Method Address (like "/out1/xlr2/gain") is interpreted as a property path navigating through the hierarchy of JSON objects. The value of each property MUST be either a primitive JSON data type, or a JSON array. Rationale: This allows to clearly separate SSC Method Addresses from SSC Method Arguments at JSON parser level without knowledge of the underlying method address tree.

The resulting JSON tree structure of hierarchical objects, the SSC *Address Space*, is tailored to describe the functionality of a specific SSC Server, in the same way as foreseen by OSC.

In JSON it is possible to serialise the complete state of all properties in the tree to a closed form, thus describing the complete state of the SSC Server. In this way, JSON can be used as an excellent extensible data format for configuration files, or for scripting applications, which drive a system of SSC Servers through a sequence of programmed configurations.

For command and control applications it is desirable to access single properties independently. This can be achieved in JSON syntax by the simple convention, that all the properties of an SSC Server that are not mentioned in a JSON message are left unchanged.

In this way, applied to the example above, the JSON form

```
{ "out1": { "xlr1": { "gain": 5 } } }
```

can be understood as an SSC Method Call of the SSC Method "/out1/xlr1/gain" with the argument 5, presumably to set the gain to that level, or as an SSC Method Reply message stating the current gain level.

4.2 JSON Message Transaction Syntax

The SSC Message exchange is described here as transaction using the following syntax:

- Prefix "TX:" indicates an SSC Message that an SSC Client is sending to an SSC Server.
- Prefix "RX:" indicates an SSC Message that the SSC Server will send back to the Client.
- An SSC-Message is written verbatim, enclosed by curly brackets { }.

A transaction to set the gain of "xlr2" of "out1" to -10 then looks like this:

```
TX: { "out1": { "xlr2": { "gain": -10 }}}
RX: { "out1": { "xlr2": { "gain": -10 }}}
```

Note that the execution of the method results in a method reply message, which for simple property setters states the actual value of the property resulting from executing the message.

The resulting value may be different from the supplied argument, e.g., for a read-only property, or if the argument is out of range, and the device may adapt it to the allowed range (this is not considered as an error):

```
TX: { "out1": { "xlr2": { "gain": -10000 }}}
RX: { "out1": { "xlr2": { "gain": -15 }}}
```

Getter-methods, which request the value of a property from the SSC Server, are realised by supplying the special JSON value null as argument to method sent to the address of the property:

```
TX: { "out1": { "xlr2": { "gain": null }}}
RX: { "out1": { "xlr2": { "gain": -15 }}}
```

Compared to binary OSC, the JSON syntax is slightly more verbose for single attribute settings, but this is compensated when multiple attributes are set in the same transaction:

```
TX: { "out1": { "xlr2": { "gain": -10, "mute": false }}}
RX: { "out1": { "xlr2": { "gain": -10, "mute": false }}}
```

SSC Address Patterns are an optional feature for an SSC Server. They allow transactions like the following, to presumably to mute all outputs at once:

```
TX: { "out1": { "*": { "mute": true }}}
RX: { "out1": { "xlr1": { "mute": true },
                "xlr2": { "mute": true }}}
```

To facilitate true interactive use, an extra-placable SSC Server is introduced as an implementation option.

4.3 SSC JSON Message Syntax

4.3.1 Elementary data types

All SSC data is composed of the primitive JSON data types:

- string: a sequence of zero or more Unicode characters in UTF-8 encoding, wrapped in double quotes, using backslash escapes. A character is represented as a single character string. Binary zero bytes can be included in a string using Unicode escape notation: "\u0000".
- number: a number in conventional "scientific" notation. `0, 42, -23, 3.141259, 1.0e+100` are all valid numbers. A Restricted SSC Server MAY reject non-integer numeric arguments, or it MAY adapt them by silently converting them to integer values.
- `true`: the boolean true value.
- `false`: the boolean false value.
- `null`: indicates a missing value; used as pseudo argument for getter-methods.

The SSC Server MAY auto-convert elementary data types without further indication to their specific purpose. The client is usually informed about the actual value used by the SSC Server in the response to the SSC method execution.

If the server auto-converts the data type, it MUST follow these conversion rules:

- string to number: String is parsed for leading whitespace, which is skipped, then for a numerical part. Any remaining non-numerical tailing characters are ignored. Completely non-numerical strings convert to zero. The exact behaviour MUST have the same result as calling the C standard function `strtod()`.
- string to boolean: Any non-empty string is true, an empty string is `false`.
- number to string: a string representation of the number suitable for interpretation by `strtod()` is used.
- number to boolean: Number zero is `false`, everything else is `true`.
- boolean to string: true results in "`true`", false in an empty string "".
- boolean to number: true is `1`, false is `0`.

4.3.2 SSC Messages

A Message is the protocol unit of transmission. Any application that sends SSC Messages is an SSC Client, any application that receives SSC Messages is an SSC Server.

An SSC Message MUST be sent as a single closed JSON form describing a JSON object. Extra whitespace between the elements of the message MUST be ignored by the receiver.

This means that every SSC Message is enclosed in a pair of curly brackets `{ }`.

The length of an SSC Message is variable. If the underlying transport protocol is packet-based, like UDP/IP or ZigBee, then exactly one SSC Message SHOULD be contained in one transport packet. If the underlying transport protocol is a byte-stream, like TCP/IP or a serial link, then SSC Messages MUST be terminated, additionally to the grouping provided by the JSON syntax, with Message Separator Characters specific to the transport. The message separator characters MUST NOT be able to occur unescaped in the non-ignored contents of the packet. Compare section [TCP/IP].

4.3.3 SSC Addresses

Every SSC Server implements a set of SSC Methods. SSC Methods are the potential destinations of SSC Messages received by the SSC Server, and correspond to each of the points of control that the application makes available. "Invoking" an SSC Method is analogous to a procedure call; it means supplying the method with arguments and causing the method's effect to take place. The SSC Server MUST respond to each received SSC Message by sending an SSC Method Reply Message to the originating SSC Client.

An SSC Server's SSC Methods are arranged in a tree structure called an SSC Address Space. The leaves of this tree are the SSC Methods and the branch nodes are called SSC Containers. An SSC Server's SSC Address Space MAY be dynamic; that is, its contents and shape MAY change over time.

Each SSC Method and each SSC Container other than the root of the tree MUST have a symbolic name which MUST be composed entirely of printable ASCII characters other than the following:

" "	space,	ASCII 32
"	double quote,	ASCII 34
#	number sign,	ASCII 35
*	asterisk,	ASCII 42
,	comma,	ASCII 44
/	slash,	ASCII 47
:	colon,	ASCII 58
?	question mark,	ASCII 63
[	open bracket,	ASCII 91
]	close bracket,	ASCII 93
{	open curly brace,	ASCII 123
}	close curly brace,	ASCII 125

The SSC Address of an SSC Method is a symbolic name giving the full path to the SSC Method in the SSC Address Space, starting from the root of the tree. An SSC Method's SSC Address begins with the character "/" (forward slash), followed by the names of all the containers, in order, along the path from the root of the tree to the SSC Method, separated by forward slash characters, followed by the name of the SSC Method. The syntax of SSC Addresses was chosen to match the syntax of URLs. The SSC address syntax SHOULD be used in documentation, but it SHOULD NOT be used as an argument to other SSC Methods; the JSON syntax of hierarchical objects SHOULD be used instead.

SSC Methods MAY be overloaded with respect to their arguments: the SSC Server may execute the method in different ways depending on the arguments given.

SSC Methods MAY also be overloaded with respect to their Address: the SSC Server may execute a different SSC Method instead, and reply with an SSC Method Reply to that different SSC Method Address ("aliased" SSC Methods). Example: a wireless receiver might report the battery charge level of the wireless transmitter either as a lifetime or as a percentage, and it might respond to a general "battery state" SSC Method Address either by executing the lifetime or the percentage Method, depending on the circumstances.

An SSC Method may be invoked with an empty argument list by supplying the JSON null value. This kind of SSC Method call SHOULD normally have the semantics of a query resulting in the current value of the property addressed by the method, without further side effects. SSC Methods that change the state of an SSC Server SHOULD normally have arguments.

Example:

- query current gain of XLR2 output of OUT1 module:

```
TX: { "out1": { "xlr2": { "gain": null }}}
RX: { "out1": { "xlr2": { "gain": -10 }}}
```

- change gain of XLR2 output of OUT1 module (note that the server adapts the value):

```
TX: { "out1": { "xlr2": { "gain": -10000 }}}
RX: { "out1": { "xlr2": { "gain": -15 }}}
```

4.3.4 SSC Message Dispatching and Pattern Matching

When an SSC Server receives an SSC Message, it must invoke the appropriate SSC Methods in its SSC Address Space based on the SSC Message's SSC Address Patterns. This process is called dispatching the SSC Message to the SSC Methods that match its SSC Address Patterns. All the matching SSC Methods are invoked with the same argument data, namely, the SSC Arguments in the SSC Message.

The parts of an SSC Address or an SSC Address Pattern are the successive names of the JSON object members in the SSC Method Call.

A received SSC Message must be dispatched to every SSC Method in the current SSC Address Space whose SSC Address matches the SSC Message's SSC Address Pattern. An SSC Address Pattern matches an SSC Address if: 1. The SSC Address and the SSC Address Pattern contain the same number of parts; and 2. Each part of the SSC Address Pattern matches the corresponding part of the SSC Address.

A part of an SSC Address Pattern matches a part of an SSC Address if every consecutive character in the SSC Address Pattern matches the next consecutive substring of the SSC Address and every character in the SSC Address is matched by something in the SSC Address Pattern. These are the matching rules for characters in the SSC Address Pattern:

1. ? in the SSC Address Pattern matches any single character
2. * in the SSC Address Pattern matches any sequence of zero or more characters
3. A string of characters in square brackets (e.g., [aeiou]) in the SSC Address Pattern matches any character in the string. Inside square brackets, the minus sign (-) and exclamation point (!) have special meanings:

- Two characters separated by a minus sign indicate the range of characters between the given two in ASCII collating sequence. A minus sign at the end of the string has no special meaning.
- An exclamation point at the beginning of a bracketed string negates the sense of the list, meaning that the list matches any character not in the list. (An exclamation point anywhere besides the first character after the open bracket has no special meaning.)

4. A comma-separated list of strings enclosed in curly braces (e.g., {foo,bar}) in the SSC Address Pattern matches any of the strings in the list.

- Any other character in an SSC Address Pattern can match only the same character.

When an SSC Address Pattern is dispatched to multiple SSC Methods, the order in which the matching SSC Methods are invoked is unspecified.

Support for address pattern matching is OPTIONAL for an SSC Server; it MAY be left out in a restricted implementation. If the SSC Server does not support address pattern matching, it MUST treat the special pattern characters like normal characters. An SSC Client can find out whether address patterns are supported by receiving error replies, or by calling the SSC Method `/osc/feature/pattern`.

4.3.1 SSC Methods addressing array values

An SSC Method MAY have an array of elementary data as value. The value array MAY be empty. The SSC method MUST NOT switch between returning arrays and elementary data as values.

All the elements of the array SHOULD have the same elementary data type. If the SSC Method is reflected in a corresponding /osc/limits Method, then the limits information SHOULD include the count property to describe the array size, and all elements MUST have the same elementary data type (see also section "SSC Method parameter range reflection - "/osc/limits").

An SSC Method that accepts an array as SSC Method Argument SHOULD also accept an elementary value, and handle it by silently converting it to an array containg that elementary value as single element. The SSC Method Reply MUST contain the actual array in this case.

Accessing complete arrays

It MUST be possible to invoke an array-valued SSC Method in the same way as an elementary-valued Method.

An SSC Method Call with a JSON null value as argument is used to query the current state of the

property addressed by the Method. The SSC Method Reply then contains the array value.

An SSC Method Call with a JSON array as Method Argument is used to change the property addressed by the SSC Method.

```
TX: { "presets": { "bank1": { "carriers": null }}}

RX: { "presets": { "bank1":
 { "carriers": [470000,470400,470800,471200,471600] }}}

TX: { "presets": { "bank1": { "carriers":
 [470000,470450,470800,471250,471600] }}}

RX: { "presets": { "bank1": { "carriers":
 [470000,470450,470800,471250,471600] }}}
```

An array-valued SSC Method MAY put additional requirements on the acceptable size of the array value argument. The SSC Method Reply MAY indicate an SSC Error in this case. SSC Error Code 416 ("requested range not satisfiable") MUST be used to indicate errors caused by the requested array value size.

Combined query-and-change of complete arrays

If the array-valued argument of an SSC Method Call contains a combination of null and not-null values, the SSC Server MUST interpret this as a query for the current value of the null-valued elements of the array property, combined with a change of the non-null valued elements. The SSC Method Reply MUST then state the current value of the complete array, after applying the requested changes. A restricted SSC Server MAY respond with an SSC Error 414 ("request too complex") if it does not support this kind of request.

This combined query-and-change request allows an SSC client to directly change specific elements of an array-valued SSC Method while saving an extra Method Call to first request the current values of array elements to keep unchanged. The first SSC Method Call of the following SSC Method transaction is only included to show all current values.

```
TX: { "presets": { "bank1": { "carriers": null }}}

RX: { "presets": { "bank1":
 { "carriers": [470000,470400,470800,471200,471600] }}}

TX: { "presets": { "bank1":
 { "carriers": [ null ,470450, null ,471250, null ] }}}

RX: { "presets": { "bank1":
 { "carriers": [470000,470450,470800,471250,471600] }}}
```

Accessing array ranges

An SSC Server MAY implement an additional SSC Method Argument syntax to facilitate partially querying and changing of array-valued SSC Methods. The OPTIONAL support of this feature is is indicated in the SSC feature method /osc/feature/array_ranges (See also section "Support for array range access").

Array ranges are specified by an JSON object as part of the SSC Method Arguments. This object MAY contain the keys

- index: specifies the zero-based index of the first element of the requested array range
- count: specifies the number of elements of the requested array range

The range specification object MUST be the first element of the JSON array forming the SSC Method Argument of the SSC Method Call.

In Method Calls to query array-valued SSC Methods, the null Method Argument is replaced by an JSON array consisting of the range specification JSON object only. The SSC Method Reply MUST also contain the range specification inside the value array, followed by the values of the array elements

in the specified range. The range specification count MUST be equal to the number of value array elements.

```
TX: { "presets": { "bank1": { "carriers": [
                   { "index":1, "count":3 }
                                          ] }}}
RX: { "presets": { "bank1": { "carriers": [
                   { "index":1, "count":3 }, 470400, 470800, 471200
                                          ] }}}
```

An SSC Method Call to change an range of an array value also provides a JSON array as the SSC Method Argument, which contains the range specification as the first element, followed by the values for the array elements inside the specified range. The value for the range count MUST be equal to the number of the value array elements.

```
TX: { "presets": { "bank1": { "carriers": [
                   { "index":1, "count":3 }, 488000, 488400, 488800
                                          ] }}}
RX: { "presets": { "bank1": { "carriers": [
                   { "index":1, "count":3 }, 488000, 488400, 488800
                                          ] }}}
```

Special array range specifications

If the properties index and/or count are missing from a range specification they default to the following values:

- index: zero
- count: current size of the array value addressed by the SSC Method

In SSC Method Replies a default array range specification MUST be suppressed to minimise the message size.

So the following transaction is equivalent to "Accessing complete arrays":

```
TX: { "presets": { "bank1": { "carriers": [ {} ]
                                          }}}
RX: { "presets": { "bank1":
 { "carriers": [470000,470400,470800,471200,471600]
                                          }}}
```

If the specified value for index is negative, then the array range shall start at an index of array size plus the specified value (e.g., -1 indicates the last array element).

If the specified value for count is negative, then the array range shall contain as many elements as the array size plus the specified value (e.g., -1 indicates all but one array element).

In SSC Method Replies, all values of a range specification MUST be given as positive numbers, even if they were specified with negative values in the SSC Method Call. Thus the last array element can be requested as follows:

```
TX: { "presets": { "bank1": { "carriers": [
       {"index":-1,"count":1}
                             ] }}}
RX: { "presets": { "bank1": { "carriers": [
       {"index":4,"count":1}, 471600
                             ] }}}
```

Example requesting a negative count. From the total array size of 5, 5 - 2 = 3 values are returned for count=-2.

```
TX: { "presets": { "bank1": { "carriers": [ {"index":1,"count":-2} ]
}}}
RX: { "presets": { "bank1": { "carriers": [
                   { "index":1, "count":3 }, 470400,470800,471200
                                          ] }}}
```

The current size of the array addressed by the SSC Method can be queried by specifying an empty array range starting at the last element. The index value given in the array range specification of the Method Reply is one less than the array size, e.g. if count is not constant (see also "count":-1 in "SSC Method parameter range reflection - "/osc/limits").

```
      TX: { "presets": { "bank1": { "carriers": [ {"index":-1,"count":0} ]
}}}
      RX: { "presets": { "bank1": { "carriers": [ {"index":4,"count":0} ]
}}}
```

Error handling

If an SSC Method Call queries an array value, and specifies an array range that is not a subrange of the actual array value, then the SSC Server MUST adapt the specified range to fit the actual array range, in the following order:

- adapt the requested array range index to the nearest valid actual array index, i.e., either the requested index, zero, or one less than the actual array size
- adapt the requested array range count to the nearest possible size, i.e., the requested size, or the number of elements starting at the adapted index.

If an SSC Method Call requests to change an array value, and specifies an array range that is not a subrange of the actual array value, then the SSC Method Reply MUST consist of a regular Reply indicating the actual array size combined with an SSC Error 416 ("requested range not satisfiable", see also section "SSC error state - /osc/error"). The actual array size is indicated by an array range specification with size zero and the index of the last array element, e.g.:

```
      TX: { "presets": { "bank1": { "carriers": [
                        { "index":3, "count":2 }, 488800, 488800
                                                ] }}}
      RX: { "osc":{"error":[{"presets":{"bank1":{"carriers":[
                           416,{"desc":"requested range not satisfiable"}]
                                                    }}}]},
            "presets":{"bank1":{"carriers":[{"index":4,"count":0}]}}}
```

4.3.2 Temporal Semantics and SSC Time Tags

Per default, the SSC Server shall invoke the SSC Methods addressed by an SSC Message immediately, i.e., as soon as possible after receipt of the message.

An SSC Server may have access to a representation of the correct current absolute time. The optional SSC Method /device/time can be used to query and optionally set the local SSC time used by a device. SSC does not provide a mechanism for clock synchronisation; if an SSC Server utilises a mechanism like NTP or PTP to sync to the absolute time it should handle request to set its SSC time by introducing a local offset from SSC time to the absolute time.

An SSC Message may contain the SSC method /osc/timetag in addition to other methods. In this case, the SSC Time Tag indicates the time when the SSC Server shall execute all of the methods contained in an SSC Message. If the time represented by the SSC Time Tag is before or equal to the current time, the SSC Server should invoke the methods immediately (unless the user has configured the SSC Server to discard messages that arrive too late). Otherwise the SSC Time Tag represents a time in the future, and the SSC Server must store the SSC Message until the specified time and then invoke the appropriate SSC Methods.

Time tags are represented by a JSON number. The integer part of the number specifies the number of seconds since January 1, 2000, and decimal part specifies subsecond precision. Time tags with a value less than 31622400 (corresponding to January 1, 2001) are interpreted as a time offset relative to the current SSC time of the SSC Server.

The actual precision that the SSC Server supports is implementation dependent; especially, it's allowed for an restricted SSC Server to ignore the fractional part of a time tag without raising an error. An SSC Client may enquire the supported time precision by invoking the method /device/timeprecision.

SSC Method Invocations in the same SSC Message are atomic; their corresponding SSC Methods should be invoked in immediate succession as if no other processing took place between the SSC Method invocations.

4.3.3 SSC Sessions

An SSC Session is defined by the association of a specific SSC Client with an SSC Server. The SSC Server MAY keep state information specific to each SSC Client (e.g., state relating to SSC Method

subscriptions, or authorisation). If the SSC Server keeps such state, it MUST be coupled to the SSC Session. When the SSC Session terminates, session state SHOULD be cleared (e.g., all SSC Method subscriptions of the client are cancelled).

An SSC Session begins implicitly with the first SSC Method Call that a specific SSC Client sends to an SSC Server.

The SSC Server SHOULD provide the SSC Method `/osc/state/close` to allow the SSC Client to actively terminate the SSC Session (see also section "SSC Session termination - /osc/state/close").

If the underlying transport protocol is based on connections (e.g., TCP/IP), then the SSC Session MUST last for the duration of the connection. The SSC Session MUST be terminated when the connection is closed. The SSC Server MAY send an SSC Method Reply `{"osc":{"state":{"close":true}}}` before it terminates the SSC Session. The SSC Client MUST also consider the SSC Session as terminated if the connection is closed.

If the underlying transport protocol is not based on connections (e.g., UDP/IP), then the SSC Session MUST last for at least 60 seconds. The SSC Server SHOULD terminate the session automatically 60 seconds after the last succesful SSC Method Call. Each succesful SSC Method Call sent by the SSC Client MUST reset the automatic SSC Session timeout interval. The SSC Client MAY call the SSC Method `/osc/ping` for this purpose. The SSC Server SHOULD send an SSC Method Reply `{"osc":{"state":{"close":true}}}` before it terminates the SSC Session. The SSC Client SHOULD also consider the SSC Session as terminated 60 seconds after receiving the last SSC Message from the SSC Server.

The SSC Server SHOULD keep state information that is explicitly specific to the SSC Session in SSC Methods that are based in the SSC Container `/osc/state`, e.g., information about SSC Method subscriptions (see also section "SSC subscriptions - /osc/state/subscribe"). Additionally, SSC Server state specific for an SSC Session MAY affect the results of calls to other SSC Methods (e.g., the SSC Server might only allow a single SSC Client at a time, or it might provide an SSC Method for Session authorisation, and reject other SSC Method Calls with SSC Error replies until an SSC Client has been authorised).

5. General SSC Address Schema

Some parts of the SSC address space are reserved by this specification for purposes of meta-protocol information, generic device-independent features, and device capability description. The reserved parts of the address space are rooted in the addresses: /osc /device /internal

The addresses and methods rooted in these reserved addresses are described in the following sections. This specification should be revised if additional addresses in the reserved address spaces, or additional reserved address spaces are introduced.

The /internal address space is reserved for internal usage in the SSC Server itself. SSC Clients MUST NOT send any requests addressing methods based in /internal, and SSC Servers MUST NOT implement any externally callable methods.

5.1 SSC Meta Information - /osc

5.1.1 SSC Protocol version - /osc/version

Read-only value. Reports the SSC version implemented in the server.

```
TX: { "osc": { "version": null } }
RX: { "osc": { "version": "1.2" } }
```

5.1.2 SSC error state - /osc/error

Read-only method. Typically this method is not requested actively by the client, but the server sends it as the SSC Method Reply to a faulty SSC Method Call.

Simple example:

```
TX: { "out1": { "xlr23": { "gain": 10 } } }
RX: { "osc": { "error": [
                  { "out1": { "xlr23": [ 404, { "desc": "not found" } ]
                      } }
                            ] } }
```

The error method result MUST contain an array of all the error messages resulting of all method executions in the client message.

The error method result array MUST contain as elements one or more SSC Address Trees corresponding to the method address of each faulty method execution (in the example: "/out1/xlr23", corresponding to the JSON object chain { "out1": { "xlr23": ... } }).

The value of each of the error object chains MUST be an array; this is the actual error message resulting from executing the specified method address. Typeset in bold in the example.

The error message MUST contain an integer numeric value, the error code. The error code SHOULD be chosen from the list of error codes detailed below. The SSC Server MAY send different error codes, which then SHOULD be chosen in the same spirit as the canonical error codes.

The error message MAY contain additional information about the error. This will be contained in a JSON object, given as the second item of the error array. Exceptionally this array object has arbitrary properties. The additional information MAY contain a human-readable description of the error; this MUST be sent as a string value for the property named desc (for "description"). The additional error information MAY contain other properties intended for debug or service purposes, or future protocol extensions. The SSC Client MUST ignore any properties that it does not know.

If the SSC Server sends a non-canonical error message, it SHOULD supply human-readable "desc" information as well, because the SSC Client can‘t be expected to react in any specific way to an unknown error, other than to relay the description to the user.

The language of any error description MAY depend on the optional /device/language setting.

The following complex example shows how an SSC Message containing three SSC Method calls is answered, where two methods fail and one succeeds:

```
TX: { "out1": { "xlr1": { "mute": true },
                "xlr23": { "gain": 3 }},
      "out2": { "xlr1": { "gain": 42 }}}
```

```
RX: { "osc": { "error": [ {
      "out1": { "xlr23": [ 404, { "desc": "not found" }]},
      "out2": { "xlr1": [ 307, { "desc": "not just now" }]}
                        } ] },
      "out1": { "xlr1": { "mute": true }} }
```

If the request message violates the JSON syntax, the complete message cannot reliably be parsed and MUST NOT be partially parsed or executed, so that the SSC Server MUST send an error response (400, "not understood") relating to the complete message, not to any method address. This error result message would look like this:

```
TX: { "out1": { "xlr23": { "ga schnr blabl

RX: { "osc": { "error": [ 400, { "desc": "not understood" }]}}
```

Error method results for successful method executions MUST NOT be sent without being explicitly requested by the client, by querying "/osc/error". Example:

```
TX: { "out1": { "xlr1": { "gain": 17 }},
      "osc": { "error": null }}

RX: { "osc": { "error": [ { "out1": { "xlr1": { "gain":
                              [ 202, { "desc": "adapted" }]}}} ] },
      "out1": { "xlr1": { "gain": 15 }}}
```

The error code is a three digit integer, defined in the style of SMTP, FTP or HTTP error codes.

The first digit of the error code defines the class of response. The last two digits do not have any categorization role.

There are 5 values for the first digit:

- 1xx - Informational response - Request received, continuing process.
- 2xx - Success - The action was successfully received, understood, and accepted.
- 3xx - Incomplete - Further action must be taken in order to complete the request.
- 4xx - Client Error - The request contained bad syntax or cannot be fulfilled.
- 5xx - Server Error - The server failed to fulfill an apparently valid request.

A simple SSC Client would only have to look at the first digit of the error code in order to determine what how to deal with the Method Reply.

1xx - Informational response

interim status for time-consuming methods.

- 100 continue
- 102 processing

2xx – Success

- 200 OK
- 201 Created
- 202 Adapted
- 210 Partial Success

3xx – Incomplete

- 310 subscription terminates

4xx - Client Error

- 400 message not understood
- 401 authorisation needed
- 403 forbidden
- 404 address not found
- 406 not acceptable (e.g., wrong type for parameter)
- 408 request time out

- 409 conflict
- 410 gone
- 413 request too long
- 414 request too complex
- 416 requested range not satisfiable
- 422 unprocessable entity (error in a complex method parameter)
- 423 locked
- 424 failed dependency
- 450 answer too long
- 454 parameter address not found (e.g., address in a subscription request)

5xx - Server Error

- 500 internal server error
- 501 not implemented
- 503 service unavailable

5.1.3 SSC transaction ID - /osc/xid

When an SSC Clients calls the Method /osc/xid, the parameters supplied for the method will be reflected back in the Method Reply of the SSC Server. This can be used by the client to keep track of client-side per-server state.

```
TX: { "osc": { "xid": 1234567, "version": null }}
RX: { "osc": { "xid": 1234567, "version": "1.2" }}
```

See also section "SSC subscriptions - /osc/state/subscribe" for a special application of this Method to subscriptions.

5.1.4 SSC Ping - /osc/ping

When a client invokes the /osc/ping method the server will immediately respond with an `/osc/ping` response with identical result as invoked. The invoked parameters in an array COULD exceptionally have arbitrary properties.

With no parameters:

```
TX: { "osc": { "ping": null }}
RX: { "osc": { "ping": null }}
```

With some parameters:

```
TX: { "osc": { "ping": [ "abcdefghijklm", 3.14159 ] }}
RX: { "osc": { "ping": [ "abcdefghijklm", 3.14159 ] }}
```

5.1.5 SSC Schema reflection - /osc/schema

The `/osc/schema` method exists to allow clients to query servers about what address schemes are available on a specific server.

For instance, in our standard example the following Method Call on the top level `/out1` address

```
TX: { "osc": { "schema": [ { "out1": null } ] }}
```

would return the following SSC Reply Message which describes the addresses that are contained in the `/out1` address one level deep:

```
RX: { "osc": { "schema": [{ "out1": { "xlr1": {}, "xlr2": {} }}] }}
```

An alternative representation of the same SSC Reply in a format that unbundles the SSC Address Tree into an array of SSC Addresses is:

```
RX: { "osc": { "schema": [
      { "out1": { "xlr1": {} }},
      { "out1": { "xlr2": {} }}] }}
```

SSC Clients MUST be able to understand both bundled and unbundled Replies.

Note that the responses are empty JSON objects if the address is an SSC Container for more addresses, JSON null if the address is an SSC Method Address.

The method `/osc/schema` may be called with a null parameter. This is equivalent to querying for the root address schema.

The SSC Client is able to enumerate the complete SSC Address Space of the SSC Server by starting with a query for the address root scheme `{ "osc": { "schema": null }}`, and recursively querying all the SSC Addresses where the replies point to SSC containers.

5.1.6 SSC Method parameter range reflection - /osc/limits

The `/osc/limits` method allows clients to query what kind of values and what range are accepted by the server in an SSC Method call as parameter values. The response of the request is always a JSON array containing a JSON object describing properties of the addressed SSC Method. These properties MUST be constant during runtime of the device.

The property list is extensible for application-specific features as well as for revised versions of this specification.

Currently defined optional properties are:

"type": `string`	"Number", "String", "Boolean", or "Container"
"min": `number`	minimum valid value
"max": `number`	maximum valid value
"inc": `number`	recommended user interface increment value
"length": `number`	maximum length of a string
"count": `number`	count of array elements that can be expected in responses and that has to be used for settings. -1 represents a not constant or indistinct size (see also section "Special array range specifications").
"subscr": `boolean`	if false then the value can not be subscribed, if true then the value can be subscribed.
"const": `boolean`	if false then the value can change during runtime, if true then the value is constant. "const":`true` implies "subscr":`false` and "writeable":`false`.
"writeable": `boolean`	if false then the value can not be set, if true then the value can be changed by command. "writeable":`true` implies "const":`false`
"units": `string`	String describing value units (preferably SI)
"desc": `string`	descriptive text, meant for display to the user
"desc_ref": `string`	reference to a not constant description node
"option": `number`, `string`	array of all allowed options for the value
"option_desc": `string`	array with description text relating to the option values

The language for "units", "description", and "option_desc" MAY depend on /device/language, see also section "/device/language".

Examples:

```
TX: { "osc": { "limits": [
                { "out1": { "xlr1" : { "level" : null }}} ] }}

RX: { "osc": { "limits": [
                { "out1": { "xlr1" : { "level" : [{ "type": "Number",
                    "min": -10, "max": 18, "inc": 3, "units": "dB",
                          "desc": "output level"
                             }] }}}] }}

TX: { "osc": { "limits": [ { "main_format": null } ] }}

RX: { "osc": { "limits": [ { "main_format" : [
       { "type": "String",
          "desc": "main output mode", "option": [ "analogue",
             "digital" ], "option_descr": [ "analogue",
                    "digital AES3" ]}] } ] }}
```

Similar as described for /osc/schema, the SSC Server may format the Method Replies in bundled or unbundled representation of the SSC Addresses, and the SSC Client MUST be able to understand either.

5.1.7 Session-specific SSC Address Space - /osc/state

SSC Methods under the /osc/state Address have results which are specific to the SSC Session between SSC Client and SSC Server. This means that it is possible that different SSC Clients invoke the same SSC Method with different arguments, and the immediate reply as well as the resulting state of the SSC Server will differ for each SSC Client.

This behaviour differs from the normal behaviour of an SSC Server, where the server state is shared between all SSC Clients and connections.

5.1.8 SSC Session termination - /osc/state/close

When an SSC Client calls this SSC Method with a true argument, the SSC Server MUST terminate the SSC Session immediately after the reply has been sent.

In case of an underlying connection-oriented transport like TCP, the SSC Server MUST close the transport-layer connection after the SSC Method Reply has been sent.

When the SSC Session is terminated, the SSC Server clears any state specific to the session, see also section "SSC Sessions".

Example:

```
TX: { "osc": { "state": { "close": true }}}
RX: { "osc": { "state": { "close": true }}}
< Server closes connection >
```

When applications of SSC Servers need to regulate SSC Clients, for example to enforce exclusive access for a single Client at a time, this Method SHOULD be used to detect disconnecting Clients even when the transport layer works connection-less. The SSC Server would see the first SSC Method Call of each unknown SSC Client as an implicit request to establish a logical connection, which the Server may accept or refuse by sending an error response, and then close the connection when this Method is called (example application: Infra-red sync of a wireless microphone).

5.1.9 SSC subscriptions - /osc/state/subscribe

A subscription request is sent by a client to a server for an address pattern to subscribe to. The SSC Server normally accepts the subscription request, and remembers that the requesting client wishes to be notified about value changes of the subscribed addresses.

The SSC Server MAY refuse subscription requests, subject to device-specific policy or implementation specific limitations. The SSC Server MUST reply on the subscription request immediately either by acknowledging the request, or by sending an error reply.

The SSC Server MUST send an initial subscription notification to the client, which contains the result of calling the subscribed SSC Methods immediately with null-argument when the subscription request is handled. This initial notification MAY be bundled with the reply to the subscription request itself.

Each subscription notification MUST have identical contents to the reply to an imagined SSC Method invocation with null-argument to the subscribed SSC Method Address at the time that the notification is sent.

The SSC Client MAY bundle a call to `/osc/xid` with the subscription request. If an xid is supplied, a reply to `/osc/xid` MAY be bundled with each subscription notification, with the xid of the reply identical to that supplied by the client.

The SSC Server MUST send value changes of the subscribed addresses to the SSC Client. By default, the SSC Server will send subscription notifications if and only if the subscribed addresses change in value. The SSC Client can modify this behaviour by supplying optional parameters with the subscription request, allowing to either throttle the rate of notifications, or stimulate additional periodic notifications even if the subscribed addresses do not change in value.

Every subscription is specific to the connection between SSC Client and SSC Server. Also each SSC Method can only be subscribed once per connection. This means, that if an SSC Client requests

a subscription which is already subscribed by that client on that connection, then the SSC Server MUST treat this as if the existing subscription was silently terminated and immediately requested anew.

Subscription notification rate parameters

Optional subscription request parameters related to notification rate:

- "min" - minimum notification period (ms), 0=none, default 0
- "max" - maximum notification period (ms), 0=none, default 0
- "bw" - maximum bandwidth for replies (byte/s), 0=unlimited, default 0

If "min" is 0, then notifications are not sent when a subscribed address changes in value, they are only sent based on the "max" period. If "min" is greater than 0, notifications are sent after the specified time duration has elapsed, even if the value of the subscribed address is unchanged.

If "max" is 0, then notifications are only sent when a value changes, or based on the "min" period. If "max" is greater than 0, then notifications are sent not earlier before the specified time duration has elapsed, even if the subscribed address changes value in the meantime.

If "bw" is greater than 0, then the bandwidth consumed for notification replies is tracked by the SSC Server, and notifications are suppressed when the specified bandwidth would be exceeded. If any notification has been suppressed due to bandwidth limitation, the SSC Server SHOULD send a notification about the actual value of the subscribed address as soon as the bandwidth limitation can be met again. The bandwidth calculation MAY be approximate, and the SSC Client MUST NOT rely on a byte-exact bandwidth limitation.

If multiple subscriptions are requested with a common set of optional parameters, then the optional parameters MUST be interpreted as if all of the requests had been issued separately, each request with the identical set of parameters. Especially, the "bw" parameter imposes the specified bandwidth limit for each subscribed SSC address separately; it is not a summary limit for all of the requested subscriptions.

The SSC Server SHOULD ignore any unknown subscription parameters. Parameter name "internal" is reserved and MUST NOT be used in SSC Client requests.

Subscription cancelling and expiration

The SSC Server MUST terminate a subscription in these cases:

- the subscribed client cancels the subscription explicitly
- a maximum number of notifications has been sent
- a maximum lifetime relating to the begin of the subscription expires
- the SSC Client closes the connection
- the transport layer of the SSC connection signals a communication error

If the SSC Server decides to terminate the connection because the lifetime or notification count expires, then it MUST inform the SSC Client by sending an error reply "310 – subscription terminated" to the SSC address that terminates subscription together with or immediately after the last subscription notification.

Optional subscription request parameters related to termination:

- "cancel" "true" cancels the subscription (default false).
- "count": maximum number of notifications to send, 0=unlimited, default 0
- "lifetime": maximum lifetime (s) of the subscription, 0=until SSC session termination, default 0

The SSC Client may renew a subscription at any time, thereby resetting all of the lifetime limitations. To renew a subscription, the SSC Client re-requests it; there's no difference between an initial subscription request and a renewal request.

Subscribing to multiple addresses

The SSC Client MAY request multiple subscriptions in a single request; either by providing them explicitly as SSC Address Tree, or by specifying address patterns as subscription addresses, or even both in the same request.

The SSC Server MAY either treat all those subscription requests separately, as if the addresses had

all been requested for subscription individually. In this case all the subscription notifications would each contain the SSC Method Reply to a single subscribed address.

Alternatively, the SSC Server MAY bundle subscription notifications which happen to be sent at the same time into a single notification. The SSC Client MUST be able to handle a bundled notification if it requests multiple subscriptions in a single request, but it MUST NOT rely on the SSC Server bundling the notifications.

In any case the SSC Server SHOULD NOT bundle notification causes, meaning that the SSC Server SHOULD NOT send any subscription notifications for addresses in a bundle with notifications to other addresses, if they would not be sent if all subscriptions had been requested individually.

If some of the SSC addresses in a subscription request must be rejected with errors, whereas other subscriptions succeed, then the SSC Server MAY reject the request completely with an error reply detailing all the failed addresses. If possible, the SSC Server SHOULD instead execute the successful subscriptions and only reject the erroneous ones. This MUST result in a successful reply message to the subscription request, with the reply value including only the successful addresses. In this case the SSC Error state MUST be set to "210 – Partial Success", and MAY be accompanied by a parameter named "failed_addresses" with an Array of Address trees composed of all the failed Method Addresses (erroneous Addresses replaced by {}), in bundled or unbundled representation. The value of the Address in the Address Tree SHOULD be set to the SSC Error Code relating to the failure of the specific Address. See also section "Subscription example transactions".

The SSC Server MAY also send an SSC Error "210 – Partial Success" when in fact all of the subscriptions have failed, because the SSC Client receives sufficient information in this Error Reply to work out this fact.

Supscription request and reply syntax

The SSC Address for subscriptions is `/osc/state/subscribe`.

This SSC Method may be called with a null parameter, which results in an SSC Address tree of all addresses currently subscribed by the SSC Client on the current connection.

The SSC Method also takes a structured parameter, specified as a JSON array.

Each element of the array is an SSC Address Tree specifying the SSC addresses that the SSC Client requests to subscribe. The SSC Address Tree MAY contain Address patterns.

Subscription parameters are specified by embedding them into the Address Tree object as the first JSON object name/value pair with the special name "#" (which can not appear as an Address). The value MUST be a JSON object containing one or more optional subscription parameters by name and value. The subscription parameters are applied for subscribing all SSC Method Addresses in the Address Tree that contains the parameter object. The "#" name/value pair SHOULD be the first item, otherwise the behaviour of the SSC Server MAY depend on the implementation.

An SSC Server that supports subscription MUST be able to interpret a single Address Tree element in the Method Argument array. Multiple Address Trees MAY be supported, or the SSC Server MAY reject them with an SSC Error 414 (request too complex).

The Response to the subscription Request will normally echo the Request, if all subscriptions can be handled successfully. If subscription parameters were requested, then the SSC Server MAY adapt the requested parameters, and MUST send back the adapted parameter values in the Reply. If multiple subscriptions are requested in a single Request, then the SSC Server might find it necessary to adapt subscription parameters differently for different Addresses. In that case, the array in the Reply MAY contain additonal Address trees containing additional adapted parameter objects. The SSC Server MAY also reject the subscription request completely (with SSC Error code 406), or partially (with SSC Error code 210) in such a case.

Subscription example transactions

Standard case: subscription request, reply and notifications, automatically terminated:

```
TX: { "osc": { "state": { "subscribe": [
                    { "out1": { "xlr2": { "level": null }}} ] }}}
RX: { "osc": { "state": { "subscribe": [
                    { "out1": { "xlr2": { "level": null }}} ] }}}
```

```
        RX: { "out1": { "xlr2": { "level": 15 }}}
...

        RX: { "out1": { "xlr2": { "level": 3 }}}
          ...
        RX: { "out1": { "xlr2": { "level": 9 }}}
          ...
        RX: { "osc": { "error": [
                 { "out1": { "xlr2": { "level": [310,
                        { "desc": "subscription terminates" } ] }}}] }}
```

Subscribing to multiple addresses in a single request:

```
        TX: { "osc": { "state": { "subscribe": [
                 { "out1": { "xlr\*": { "level": null }}} ] }}}
        RX: { "osc": { "state": { "subscribe": [
                 { "out1": { "xlr1": { "level": null },
                        "xlr2": { "level": null }}} ] }}}
        RX: { "out1": { "xlr1": { "level": 15 },"xlr2": { "level": 7} }}
          ...
        RX: { "out1": { "xlr1": { "level": 3 }}}
        RX: { "out1": { "xlr1": { "level": 5 }}}
          ...
        RX: { "out1": { "xlr2": { "level": 9 }}}
          ...
```

Subscribing with non-default parameters, partially adapted by the Server:

```
    TX: { "osc": { "state": { "subscribe": [ {
                    "#": { "min": 96, "max": 50, "lifetime": 3600 },
                        "out1": { "xlr2": { "level": null }}} ] }}}
    RX: { "osc": { "state": { "subscribe": [ {
            "#": { "min": 100, "max": 50, "lifetime": 600 },
            "out1": { "xlr2": { "level": null }}} ] }}}
    RX: { "out1": { "xlr2": { "level": 15 }}}
```

Client cancelling the subscription of the previous example:

```
    TX: { "osc": { "state": { "subscribe": [ {
                    "#": { "cancel": true },
            "out1": { "xlr2": { "level": null }}} ] }}}
    RX: { "osc": { "state": { "subscribe": [ {
                    "#": { "cancel": true },
            "out1": { "xlr2": { "level": null }}} ] }}}
```

Subscribing to multiple addresses, requesting error details, only partially successful:

```
    TX: { "osc": { "state": { "subscribe": [
                    { "out1": { "xlr1": { "level": null,
                                    "nope": null },
                        "xlr2": [ "invalid address" ]} } ] },
            "error": null }}
    RX: { "osc": { "state": { "subscribe": [
                    { "out1": { "xlr1": { "level": null }}} ] },
            "error": [ "osc": { "state": { "subscribe": [210, {
                 "desc": "Partial Success","failed_addresses": [
                        { "osc": { "xlr1": { "nope": 404 }}},
```

```
                                                    { }]}] } } ]} }
```

Subscribing to array elements if array_ranges are supported (see also section "Accessing array ranges"):

```
TX: { "osc": { "state": { "subscribe": [ { "device": { "log":
      [ {"index":-1, "count":1} ] } } ] } } }

RX: { "osc": { "state": { "subscribe": [ { "device": { "log": [
      {"index":-1, "count":1} ] } } ] } } }
    ...
    {"device":{"log": [{ "index": 1016, "count": 1 },
      "turbogenerator trip signal blocked"] }}
    ...
    {"device":{"log": [{ "index": 1017, "count": 1 },
      "required operating reactivity margin violated"] }}
    ...
    {"osc":{"error": [{ "device": { "log": [ 310, {"desc":
      "subscription terminates"} ] } }] }}
```

5.1.10 SSC reply output style - /osc/state/prettyprint

An SSC Server MAY support this Method to allow the SSC Client to select a preferred formatting style for all SSC reply messages to be sent on the connection to the SSC Client by the SSC Server. Two styles are defined:

prettyprint = `false`: compact representation, no whitespace

prettyprint = `true`: formatted by adding whitespace

The exact format of prettyprinted messages depends on the implementation. The SSC Client MUST NOT depend on any specific whitespace decoration. The default style is not specified; the SSC Server MAY choose the default style for each connection independently, e.g., by analysing the first SSC Message sent by the SSC Client; or an UDP SSC Server may use different default output styles on different UDP ports.

Example transaction:

```
TX: { "osc": { "state": { "prettyprint": false }}}
RX: { "osc":{"state":{"prettyprint":false}}}
TX: { "device": { "name": null }}
RX: { "device":{"name":"example device"}}
TX: { "osc": { "state": { "prettyprint": true }}}
RX: { "osc": { "state": { "prettyprint": true }}}
TX: { "device": { "name": null }}
RX: { "device": { "name": "example device" }}
```

5.1.11 SSC interactive method address base - /osc/state/baseaddr

This is an OPTIONAL Method. It helps to explore a device in a truely interactive manner, and may additionally be used to reduce message lengths by shortening addresses in SSC Messages for applications, where an application monitors only a tiny subset of the address space of a complex device with a deeply-nested address space.

The "baseaddr" must specify an existing, valid address on the SSC Server. It is automatically added by the SSC Server to the beginning of any SSC Address in SSC Method calls by the client. It is automatically stripped from replies by the SSC Server to the Client. The base address has effect on messages send over the specific transport-layer client connection.

If the SSC Server can't match an address in an SSC Method call to an address by prepending the base address, it additionally tries to match it as an absolute address. In this way, especially the / osc-address space can be accessed again to reset the base address.

Setting the base address to non-empty fails if the targeted address contains a method or container with the partial address "osc".

Example:

```
TX: { "osc": { "state": {"baseaddr":[{ "out1": {"xlr2": null}}] }}
RX: { "osc": { "state": {"baseaddr":[{ "out1": {"xlr2": null}}] }}
TX: { "gain": -10 }
RX: { "gain": -10 }
```

The SSC Client may query the server for the support of this feature by invoking `/osc/feature/baseaddr`.

5.1.12 SSC timed method execution - /osc/timetag

When this method is called as part of an SSC Message, all the SSC Method Calls contained in the same Message are to be executed by the SSC Server at a time determined by the timetag parameter. Compare section "Temporal Semantics and SSC Time Tags".

```
TX: { "osc": { "ping": null, "timetag": 3.0 }}
```

... 3 seconds pass ..

```
RX: { "osc": { "ping": null }}
```

5.1.13 SSC Method time stamps - /osc/timestamp

This method may be called as part of an SSC Message to estimate the communication delay and clock offset between SSC Client and server. Like in NTP or PTP, four timestamps are used, forming an array. The SSC Client sends the method call and provides the array containing the first, client-side timestamp. The SSC Server will add two server-side timestamps in the SSC Method Reply, and finally the client can insert the fourth timestamp upon reception of the Reply Message.

The four timestamps are:

- sending time of the Message from the SSC Client as value of the client `/device/time`
- reception time at the SSC Server as value of the server `/device/time`
- sending time of the Reply at the SSC Server, value of server `/device/time`
- reception time of the Reply at the client, value of client `/device/time`

Example:

```
TX: { "osc": { "ping": null, "timestamp": [ 396711511.044569 ] }}
RX: { "osc": { "ping": null,
               "timestamp": [ 396711511.044569, 396711500.05,
                              396711500.08, 396711511.644569 ] }}
```

Support for timestamps is optional. If the SSC Server doesn't implement it, it shall omit the timestamp completely from the Reply Message.

5.1.14 SSC Method Authorisation - /osc/tan

In some applications (like conference systems), remote configurability of SSC devices might pose a security risk, because some attacker with access to the LAN might remotely gain access to sensitive audio streams by reconfiguring devices.

Transaction Authorisation Numbers (TANs) are a simple but efficient means to control configuration access without relying on complex cryptographic protocols, which might be too complex for environments like media control systems.

A list of TAN numbers is distributed between the SSC devices in a system. The SSC Client must provide a valid TAN with each SSC Method Call. The SSC Server refuses to execute the method if it finds the TAN to be invalid. Each TAN may only be used once.

How the TAN list is distributed between the SSC devices in a system is out of the scope of this specification.

```
TX: { "osc": { "ping": null, "tan": "1124frvso!" }}
RX: { "osc": { "ping": null }}
TX: { "osc": { "ping": null, "tan": "1124frvso!" }}
RX: { "osc": { "error": [{
        "osc": { "tan": [{"code": 403, "desc": "TAN invalid"}],
                 "ping": [{"code": 403, "desc": "forbidden"}] }}] }}
```

5.1.15 SSC protocol feature reflection - /osc/feature

This SSC Address Space is provided to enable the SSC Client to query the SSC Server whether optional protocol features are supported. In the spirit of extensibility, the SSC Server MUST send a reply with a value of false for each feature that is doesn't know about, even if the feature didn´t exist at all when the server was implemented.

Support for pattern matching on SSC addresses

A client may query `/osc/feature/pattern` to inquire whether the SSC Server supports pattern-matching meta-characters in SSC addresses. The support is described by a string containing one character for each supported matching pattern:

- '*' matches on complete address parts are supported
- '?' matches on partial addresses are supported
- '[' matches on character ranges are supported

Example:

```
TX: { "osc": { "feature": { "pattern": null }}}
RX: { "osc": { "feature": { "pattern": "*" }}}
```

If the SSC Server does not support address pattern matching at all, it MAY also reply with false.

Support for time tags

A client may query `/osc/feature/timetag` to inquire whether the SSC Server supports timed method execution.

```
TX: { "osc": { "feature": { "timetag": null }}}
RX: { "osc": { "feature": { "timetag": false }}}
```

Support for subscription

A client may query `/osc/feature/subscription` to inquire whether the SSC Server supports SSC subscription.

```
TX: { "osc": { "feature": { "subscription": null }}}
RX: { "osc": { "feature": { "subscription": true }}}
```

Support for method base address

A client may query `/osc/feature/baseaddr` to inquire whether the SSC Server supports to set a method base address.

```
TX: { "osc": { "feature": { "baseaddr": null }}}
RX: { "osc": { "feature": { "baseaddr": false }}}
```

Support for array range access

A client may query `/osc/feature/array_ranges` to inquire whether the SSC Server supports advanced access to array elements.

```
TX: { "osc": { "feature": { "array_ranges": null }}}
RX: { "osc": { "feature": { "array_ranges": true }}}
```

5.2 Generic Device Information and Settings: Address Space - /device

The device settings are parameters that are common to any compliant device on the network that are relevant to the device as a whole.

5.2.1 /device/identity/product

Read-only string. Product identification, should be identical to the designation on the label on the product itself.

5.2.2 /device/identity/version

Read-only string. Product version, e.g., firmware revision.

5.2.3 /device/identity/serial

Read-only string. Unique product number, preferrably identical to the number on a product label.

5.2.4 /device/identity/vendor

Read-only string: often "Sennheiser".

5.2.5 /device/name

User-settable persistent device name. This name should be the preferred, and most convenient way for the customer to identify devices. If the device has a display, this name should be displayed there, preferably in the network context; if it has a menu, this name should be configurable or it COULD be derived from /device/system or/and e.g. /device/location. This name MUST not be misunderstood as channel name that identifies for example an audio link like rx1/name but it is to understand as device identification in a network environment.

If the device is networked, this name shall be used as the name for device discovery (see also section "SSC Server Discovery"). If device discovery automatically renames the device to resolve naming conflicts, this should be reflected in this property as well as in the display of the device.

Note that the maximum lenght of /device/name which may be represented in the limits (see also section "SSC Method parameter range reflection - /osc/limits") is dependent on the device discovery concerning unicode length and the name collision resolution.

5.2.6 /device/system

User-settable persistent system name.

This determines the logical system that this device is a part of. It's intended to help the customer to logically separate devices which form different functional systems, but are accessed by means of the same communication medium.

5.2.7 /device/time

Current absolute clock value of the device. Units are seconds, potentially fractional, counting the seconds from 2000-01-01T00:00:00+0000 UTC. The time can be changed by using this address, unless the server is synchronised to absolute time externally, e.g., by means of NTP. If the device doesn't support absolute clock value, the clock value shall always indicate a time before 2001-01-01T00:00:00+0000, that means the value is greater than 0 and less than 31622400. If the device doesn't support a clock at all, it shall always return the value 0.

5.2.8 /device/timeprecision

Read-only value specifying the precision of the clock of the device as used for /device/time, SSC time tags, and SSC time stamps.

If the device doesn't support a clock at all, it shall always return the value 0.

5.2.9 /device/language

User-settable value determining the language to be used for values returned by the server which are meant to be displayed to the user. Examples are `/osc/error/desc`, `/osc/limits/.../desc`, `/osc/limits/.../option_desc`.

An SSC Client can determine the possible language options by querying `/osc/limits/device/language`.

Languages are encoded with 2-3-letter-codes as per the locale convention, e.g. "de", "de_DE". Default language is British English, "en_GB".

Support for languages is optional. Restricted SSC Servers may omit description texts completely; they should return an empty string for /device/language. Servers offering only one fixed language should return that for /device/language, and refuse attempts to change it.

5.2.10 /device/network

This address space allows remote configuration of network settings. Devices without network connectivity don't need to implement the address space. Devices that don't support IPv4 should not implement the IPv4 address space.

/device/network/ether/interfaces

Read-only array containing a list of all the user-relevant ethernet interface of the device. The names SHALL match the user-readable labeling of the connectors of the device. If the physical port on the device do not carry a textual label, then the textual designation in the user manual of the product SHALL be used. Internal interfaces which are not accessible to the user MUST NOT be listed here. All accessible physical ports MUST be listed here, even if they all can only be used in a shared configuration, e.g., if they are connected to an internal switch.

/device/network/ether/macs

Read-only array containing a list of the MAC addresses of all the user-relevant ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The MAC addresses are specified as strings in standard hex-colon-notation.

/device/network/ipv4

Address Space for IPv4-specific settings. These generic methods only relate to the most common case of a device with only a single network interface, or when all of the physical interfaces operate in a bridged configuration, so that only one set of IPv4 network settings is necessary. More complex devices using different IPv4 addresses on different sets of physical interfaces SHOULD utilise this address space for the main remote control connection, and provide additional address spaces for the remaining interface sets; e.g., /device/network/ipv4-dante.

/device/network/ipv4/interfaces

Read-only array relating to `/device/network/ether/interfaces`. The array contains numbers indexing into the array of physical interface names, and thus also into the array of interface MAC addresses at /device/network/ether/macs. The interface index array MUST contain the indices of all physical interfaces which share the IPv4 configuration detailed by the following methods.

/device/network/ipv4/auto

Boolean value that indicates whether IPv4 shall be configured automatically by means of DHCP and ZeroConf (Auto-IP). If this value is set to true, all the following properties are read-only. true is the default.

/device/network/ipv4/ipaddr

Current IPv4 address of the device as a string in standard dot-decimal notation.

/device/network/ipv4/netmask

Current IPv4 netmask of the device as a string in standard dot-decimal notation.

/device/network/ipv4/gateway

Current IPv4 gateway of the device as a string in standard dot-decimal notation, "0.0.0.0" if no gateway is available.

/device/network/ipv6

Address Space for IPv6-specific settings. Like /device/network/ipv4, only the most common case of a single physical interface is specified here. IPv6 SHALL be applied in full autoconfiguration mode only, so that all IPv6 specific method are read-only.

/device/network/ipv6/interfaces

Read-only array relating to /device/network/ether/interfaces. The array contains numbers indexing into the array of physical interface names, and thus also into the array of interface MAC addresses

at /device/network/ether/macs. The interface index array MUST contain the indices of all physical interfaces which share the IPv6 configuration accessible by the following methods.

/device/network/ipv6/ipaddr

Read-only array of all the IPv6 addresses used on the specified physical interfaces in standard string notation. The interface specifiers for link-local IPv6 addresses MUST be stripped from the address because they have only internal relevance. An SSC Client application MUST add the interface specifier of the interface local to the client to link-local IPv6 addresses, so that they may be copied and pasted on the client side.

6. SSC Transport Layer Adaptations

The SSC data format as defined in the previous sections can be transported by different transport protocols, or stored in persistent files. This section specifies what transports are supported, and how the specific features of transport layers shall be applied to transporting SSC Messages.

If an SSC Server supports more than a single transport for SSC, it SHALL behave consistently regardless of the transport used.

6.1 UDP/IP

UDP/IP is the standard transport for all devices with an Ethernet interface or another interface typically used for internet connectivity. All those device MUST implement the UDP/IP transport for SSC.

All devices SHALL implement UDP over IPv6. Support for UDP over IPv4 is OPTIONAL.

One UDP datagram is used to transport one SSC Message. If the SSC Message is really large (e.g., a complete device configuration), IP fragmentation might fail, if a restricted device does not implement IP re-assembly properly. In that case, the SSC Server should break up the message into multiple SSC Method Calls instead. If atomic execution is relevant, SSC time tags may be used.

The UDP port number to used by the SSC Server should normally be discovered by the SSC Client by means of the server discovery protocol. The default port number is 45.

Rationale: No other standard UDP service is expected to use 45. The IANA reservation for a "Message Passing Service" is historic, and SSC is actually passing messages itself. Sennheiser was founded in 1945.

6.2 TCP/IP

Support for transporting SSC Messages over TCP/IP is OPTIONAL. An SSC device choosing to support TCP/IP SHOULD support the same set of IP versions for TCP as well as for UDP.

One important application for TCP/IP is to integrate SSC devices into media control systems. In these systems ease of use of the protocol is of special relevance.

TCP/IP is a byte-stream based transport. Therefore it is necessary to define a way of fragmenting the stream into SSC Messages.

The following two-character sequences are recognised as an SSC Message separator:

- ASCII carriage return (13) – newline (10)
- ASCII newline (10) – newline (10)

Rationale: An unescaped newline cannot appear in the data of a legal SSC Message. The newline character supports interactive use of SSC. Newline alone as single separator character would prevent formatting a large SSC Message over multiple lines (important to store SSC Messages in readable or editable text files).

To support interactive use, SSC Servers providing TCP transport MAY implement the SSC base address feature. They MAY also support the relaxed SSC parser as specified in the appendix.

SSC Messages containing time-critical commands should be pushed (TCP flag PSH) through the TCP stack for minimum latency, after writing the Message separator sequence.

The TCP port number to be used by the SSC Server is expected to be discovered by the client by means of the server discovery protocol. The default port number is 45.

Rationale: Same as UDP.

6.3 HTTP(S)/TCP/IP

Support for transporting SSC Messages over HTTP over TCP is OPTIONAL. If an SSC Server provides the HTTP transport, it MUST also provide TCP transport. Consequentially, HTTP MUST be provided over the same set of IP versions as UDP. An SSC Server supporting HTTP SHOULD support HTTP version 1.1.

HTTP should be provided by those SSC Server devices that should be easily accessible for control apps based on smart phone, tablets, or web-apps (might be served by the device itself).

SSC Messages are transported as HTTP request or reply contents. The MIME-Type of the contents MUST be specified as "application/json" in the HTTP header field "Content-Type". Using this MIME-Type eases integration of SSC-over-HTTP into Web-Apps using JSON libraries.

The SSC Client MUST send SSC Messages to the SSC Server as the contents of HTTP POST requests. The Reply of the server is transported back as the contents of the HTTP response.

There are two different models of SSC communication over HTTP:

- One SSC Message at a time:

  The SSC Client formats the HTTP request including SSC Message as contents, and specifies the length of the SSC Message in the HTTP "Content-Length"-header. The SSC Server sends the SSC Reply Message in the HTTP response, also specifying the "Content-Length" header. The HTTP connection may persist, depending on the HTTP "Connection"-header. Every SSC Client and SSC Server supporting the HTTP transport MUST implement this.

- SSC Message stream over HTTP:

  Here the client doesn't specify the "Content-Length", but instead uses HTTP "chunked" Transport Encoding. In this case, one HTTP chunk MUST contain one SSC Message, and the client can send multiple successive SSC Messages as the contents of a single HTTP request. The SSC Server SHALL use chunked transport encoding for the HTTP response as well, also using one HTTP transport chunk for one SSC Message. The SSC Client can terminate the SSC Message stream by closing the HTTP request with a zero-size chunk. At this moment, the Server SHALL also close the ongoing HTTP response. The HTTP connection MAY still persist, depending on the HTTP "Connection" header. SSC Clients and SSC Servers SHOULD implement this transport mechanism, if a high rate of SSC Message transactions is expected, because it offers significantly less protocol overhead. The SSC Server MUST refuse the HTTP request for chunked transport encoding if it does not support the SSC Message transport described here. The SSC Client MUST support chunked HTTP responses if it requests chunked transport from the SSC Server.

The HTTP request URL to be used by the SSC Client is expected to be discovered by the SSC Client by means of the server discovery protocol. The default URL is "/ssc".

The address spaces of SSC and HTTP are treated as interlinked: leading parts of SSC addresses may be appended to the HTTP request URL, meaning that this part of the SSC addresses is left out of the SSC Messages transported over this specific HTTP request. This means that this HTTP transaction:

```
TX: POST /ssc/out1/xlr2 HTTP/1.1
    Host: ssc-server.local.
    Content-Type: application/json
    { "gain": -10 }

RX: HTTP/1.1 OK
    Content-Type: application/json
    { "gain": -10 }
```

is equivalent to this:

```
TX: POST /ssc HTTP/1.1
    Host: ssc-server.local.
    Content-Type: application/json

    { "out1": { "xlr2": { "gain": -10 }}}

RX: HTTP/1.1 OK
    Content-Type: application/json

    { "out1": { "xlr2": { "gain": -10 }}}
```

SSC errors and HTTP errors are strictly separate.

The TCP port number to be used by the SSC Client is expected to be discovered by the client by

means of the server discovery protocol. The default port number is 80, same as for standard HTTP, or 443 for HTTPS.

Rationale: The standard HTTP ports are least likely to be blocked by firewall setups. The SSC service can easily coexist with other HTTP services on the same device by utilising a separated HTTP URI base, so a separate port should not be needed.

HTTP may be provided by the SSC Server as HTTP-over-TLS, optionally secured with server or even client side certificates, if absolute security is required for the SSC system.

6.4 Secure Shell Transport/TCP/IP

This is handled exactly like TCP/IP transport.

6.5 SSC Server Discovery

If IP network devices implement a discovery protocol then it MUST be DNS-SD (Apple Bonjour). The discovery service CAN be enabled or disabled by the application. The DNS Service-Type is specified as "_ssc".

Because all networked SSC Servers must implement SSC-over-UDP, they MUST all publish a DNS-SD service under "_ssc._udp”. Those servers that additionally support TCP MUST publish another DNS-SD service under "_ssc._tcp”.

The DNS-SD service instance name SHOULD/MUST be identical to the device name accessible as /device/name. DNS-SD automatic name collision resolution MUST be performed, and the resulting name changes MUST be reflected back into /device/name and the persistent device configuration due to avoid recurrent renaming. The renaming rules MAY be tailored to suit product specific requirements. Here it will be easy to replace a device by simply assign the name of the replaced device to the new device. If in a specific application an interaction with the network is not wanted and the service instance name can not be identical to /device/name then the DNS-SD service instance name MUST follow the pattern </device/name>-<id> where <id> is the unique ID which is represented in the TXT record id=. Here the name conflict resolution is improbable and the /device/name can be extracted from service instance name. In the here unexpected case of name collision the id MUST be adapted to the resulting suffix.

The DNS-SD service registration includes the port numbers used. SSC Clients SHOULD NOT rely on default ports.

The DNS-SD hostname SHOULD NOT be presented to the user. But anyway it SHOULD have a model specific prefix. It also MUST contain a unique identification suffix (e.g., derived from the device MAC or serial), to avoid name collisions and automatic renaming. Furthermore it COULD contain module identifier. The name parts MUST be separated by '-' and the model and module name SHOULD be upper case.

Additional information about the SSC Server may be provided with an DNS-SD TXT-record. The following properties are currently defined for the TXT record:

- txtvers: Version of the TXT record format. Currently "2”.
- sscvers: SSC-Version provided by the SSC Server.
- model: Model name
- id: Unique identification of the device, derived from the device MACs or serial.
- links: URI(s) of other related services.
- links.sub: URI(s) of other related services in the same device.
- links.http: If this SSC Server offers HTTP(S) transport, the base SSC request URI, including TCP port number.
- info: Optional Information, e.g. "info=simulation". which is not to misunderstand as indicator for device or communication properties but for debugging.

An SSC Server providing SSC-over-HTTP transport MUST only publish a DNS-SD record as a web server "_http._tcp”, if it provides a web app or configuration page of interest for the human user. The URI published in the HTTP DNS-SD-TXT-record is expected to differ from that for SSC-over-HTTP. SSC Method List

6.6 IEEE 802.15.4 / ZigBee / DECT

One SSC Message per transport packet.

6.7 Low-bandwidth serial infrared link

Tbd. Framing defined separately.

6.8 Byte-stream connections (serial interface etc.)

SSC Messages are separated from the byte stream in the same way as used for TCP/IP connections. Physical parameters of the serial link are out of the scope of this specification.

6.9 Unidirectional low-bandwidth monitoring

Tbd. Device has defined set of addresses which are permanently monitored.

6.10 Configuration files

Configuration files should be kept readable and editable for humans.

The following two-character sequences are recognised as a Message separator:

- ASCII carriage return (13) – newline (10)
- ASCII newline (10) – newline (10)

Normally, a configuration file would contain only one SSC Message, encompassing all configurable settings. Multiple SSC Messages have the same semantics as if they were sent to an SSC Server in the sequence as they appear in the file. Allowing multiple Messages in configurations is useful for log-structured config files, where each setting may be appended, one at a time, as they are configured by the user handling the device.

Further syntactic elements are allowed in SSC files: "#" as the first character on a line introduces a comments line, which should be ignored by the SSC parser. As a special comment, "#!" on the first line of a file is interpreted in the UNIX way to indicate the application that should handle the file, so that the files may be used as executables.

6.11 Scripting files

See configuration files. Normally, multiple SSC Messages wil be contained in a script, one after the other. Allow meta-commands for script player:

```
#! osc: { delay: 3 }
```

Time stamps may be inserted by SSC logger:

```
#! osc: { timestamp: "2012-07-26T12:51:22+0200" }
```

To play back recorded log with logged pauses, starttime-meta-command allows SSC script player to re-interpret logged time stamps as relative delays:

```
#! osc: { starttime: "2012-07-26T12:35:46+02:00" }
```

6.12 Apendix

Relaxed SSC Parser for Interactive Use

An SSC Server may implement the following relaxed parsing rules to interpret interactive input. SSC Messages generated by the SSC Server or SSC Messages generated by programs must be strictly conformant to this specification.

- the top-level braces of an SSC message may be left out.
- the double quotes surrounding SSC address parts may be omitted.
- the JSON value null that indicates SSC queries may be left out.
- SSC address parts may be concatenated with "/", and inner JSON objects may be left out, when only a single SSC Method is addressed in an SSC Message.

The relaxed rules allow these SSC transactions:

```
TX: { "osc": { "state": { "baseaddr": "/out1/xlr2" }}}
RX: { "osc": { "state": { "baseaddr": "/out1/xlr2" }}}
TX: { "gain": -10 }
RX: { "gain": -10 }
TX: { "mute": null }
RX: { "mute": false }
```

to be entered interactively in this even more intuitive way:

```
TX: osc/state/baseaddr: "/out1/xlr2"
RX: { "osc": { "state": { "baseaddr": "/out1/xlr2" }}}
TX: gain: -10
RX: { "gain": -10 }
TX: mute:
RX: { "mute": false }
```

6.13 References

Normative References

tbd.

Additional References

tbd.

7. SSC Method List

7.1 "/m/beam/elevation"

The metering method returns the elevation of the beam in degree.

- type: Number
- const: false
- writeable: false
- max: 90
- min: 0
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"m":{"beam":{"elevation":null}}}]}}}
```

Example SSC get:

```
{"m":{"beam":{"elevation":null}}}
```

7.2 "/m/beam/azimuth"

The metering method returns the azimuth of the beam in degree.

- type: Number
- const: false
- writeable: false
- max: 359
- min: 0
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"m":{"beam":{"azimuth":null}}}]}}}
```

Example SSC get:

```
{"m":{"beam":{"azimuth":null}}}
```

7.3 "/m/in1/peak"

The metering method returns the current detected input peak level.

- type: Number
- const: false
- writeable: false
- units: dB
- max: 0
- min: -90
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"m":{"in1":{"peak":null}}}]}}}
```

Example SSC get:

```
{"m":{"in1":{"peak":null}}}
```

7.4 "/m/ref1/rms"

RMS level of AEC reference signal via DANTE.

- type: Number
- const: false
- writeable: false

- units: dBfs
- max: 0
- min: -120
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"m":{"ref1":{"rms":null}}}]}}}
```

Example SSC get:

```
{"m":{"ref1":{"rms":null}}}
```

7.5 "/device/identification/visual"

If set to true the device will go to identification state. The LEDs will blink for 10 seconds. Parameter resets to false after the timeout of 10 seconds.

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"identification":{"visual":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"identification":
    {"visual":null}}}]}}}
```

Example SSC request:

```
{"device":{"identification":{"visual":null}}}
```

Example SSC response:

```
{"device":{"identification":{"visual":true}}}
```

7.6 "/device/button/state"

Returns the state of the reset button.

- type: Boolean
- const: false
- writeable: false
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"button":
    {"state":null}}}]}}}
```

Example SSC request:

```
{"device":{"button":{"state":null}}}
```

7.7 "/device/button/label"

Returns the label of the reset button.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"button":{"label":null}}}
```

7.8 "/device/button/description"

Description for the container "device/button".

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"button":{"description":null}}}
```

7.9 "/device/identity/version"

Returns the firmware version of the device.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"identity":{"version":null}}}
```

7.10 "/device/identity/vendor"

Vendor name of the product.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"identity":{"vendor":null}}}
```

7.11 "/device/identity/serial"

Returns the serial number of the device.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"identity":{"serial":null}}}
```

7.12 "/device/identity/product"

Returns the identity of the product.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"identity":{"product":null}}}
```

7.13 "/device/identity/hw_revision"

Returns device hardware revision.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"identity":{"hw_revision":null}}}
```

7.14 "/device/update/progress"

Returns the progress state of the device firmware update.

- type: Number
- const: false
- writeable: false
- max: 100
- min: 0
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"update":
    {"progress":null}}}]}}}
```

Example SSC request:

```
{"device":{"update":{"progress":null}}}
```

7.15 "/device/update/error"

Returns the result of the device firmware update.

- type: String
- const: false
- writeable: false
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"update":
    {"error":null}}}]}}}
```

Example SSC request:

```
{"device":{"update":{"error":null}}}
```

7.16 "/device/update/enable"

Set to true, to start the firmware update. The device sets to false in case the firmware update ended (after reboot or error).

Abortion by client is not possible.

- type: Boolean
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"update":{"enable":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"update":
    {"enable":null}}}]}}}
```

Example SSC request:

```
{"device":{"update":{"enable":null}}}
```

Example SSC response:

```
{"device":{"update":{"enable":true}}}
```

7.17 "/device/led/custom/color"

Custom LED color during power cycle. The selected color will be shown for 5 seconds.

Activate with `{"device":{"led":{"custom":{"active":true}}}}`

- type: String

- default: BLUE
- const: false
- writeable: true
- options: 1. LIGHT GREEN 2. GREEN 3. BLUE 4. RED 5. YELLOW 6. ORANGE 7. CYAN 8. PINK
- subscr: true

Example SSC set:

```
{"device":{"led":{"custom":{"color":"CYAN"}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":{"custom":
    {"color":null}}}}]}}}
```

Example SSC request:

```
{"device":{"led":{"custom":{"color":null}}}}
```

Example SSC response:

```
{"device":{"led":{"custom":{"color":"CYAN"}}}}
```

7.18 "/device/led/custom/active"

Set to "true" to activate custom LED color. Hint: After reboot this will be set back to false.

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"led":{"custom":{"active":true}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":{"custom":
    {"active":null}}}}]}}}
```

Example SSC reuest:

```
{"device":{"led":{"custom":{"active":null}}}}
```

Example SSC response:

```
{"device":{"led":{"custom":{"active":true}}}}
```

7.19 "/device/led/mic_mute/color"

User defined mic mute color. The selected color will be shown for 5 seconds.

- type: String
- default: RED
- const: false
- writeable: true
- options: 1. LIGHT GREEN 2. GREEN 3. BLUE 4. RED 5. YELLOW 6. ORANGE 7. CYAN 8. PINK
- subscr: true

Example SSC set:

```
{"device":{"led":{"mic_mute":{"color":"ORANGE"}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":{"mic_mute":
    {"color":null}}}}]}}}
```

Example SSC request:

```
{"device":{"led":{"mic_mute":{"color":null}}}}
```

Example SSC response:

```
{"device":{"led":{"mic_mute":{"color":"ORANGE"}}}}
```

7.20 "/device/led/mic_on/color"

User defined mic on color. The selected color will be shown for 5 seconds.

- type: String
- default: GREEN
- const: false
- writeable: true
- options: 1. LIGHT GREEN 2. GREEN 3. BLUE 4. RED 5. YELLOW 6. ORANGE 7. CYAN 8. PINK
- subscr: true

Example SSC set:

```
{"device":{"led":{"mic_on":{"color":"BLUE"}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":{"mic_on":
    {"color":null}}}}]}}}
```

Example SSC request:

```
{"device":{"led":{"mic_on":{"color":null}}}}
```

Example SSC response:

```
{"device":{"led":{"mic_on":{"color":"BLUE"}}}}
```

7.21 "/device/led/show_farend_activity"

Enable far-end activity LED mode.

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"led":{"show_farend_activity":false}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":
{"show_farend_activity":null}}}]}}}
```

Example SSC request:

```
{"device":{"led":{"show_farend_activity":null}}}
```

Example SSC response:

```
{"device":{"led":{"show_farend_activity":false}}}
```

7.22 "/device/led/brightness"

LED brightness in 6 steps. Set to '0' to turn off LEDs.

- type: Number
- default: 5
- const: false
- writeable: true
- max: 5
- min: 0
- subscr: true

Example SSC set:

```
{"device":{"led":{"brightness":5}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"led":
    {"brightness":null}}}]}}}
```

Example SSC request:

```
{"device":{"led":{"brightness":null}}}
```

Example SSC response:

```
{"device":{"led":{"brightness":5}}}
```

7.23 "/device/network/ether/macs"

List of all MAC addresses.

- type: String
- const: true
- writeable: false
- count: 1

Example SSC request:

```
{"device":{"network":{"ether":{"macs":null}}}}
```

7.24 "/device/network/ether/interfaces"

List of IPv4 interface names.

- type: String
- const: true
- writeable: false
- count: 1

Example SSC request:

```
{"device":{"network":{"ether":{"interfaces":null}}}}
```

7.25 "/device/network/ipv4/netmask"

List of current IPv4 netmasks.

- type: String
- const: false
- writeable: false
- count: 1
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
      {"netmask":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"netmask":null}}}}
```

7.26 "/device/network/ipv4/manual_netmask"

List of IPv4 netmasks which will be applied after setting "device/network/ipv4/auto" to "false".

- type: String
- default: ["255.255.255.0"]
- const: false
- writeable: true
- count: 1
- subscr: true

Example SSC set:

```
{"device":{"network":{"ipv4":{"manual_netmask":["255.255.254.0"]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
      {"manual_netmask":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"manual_netmask":null}}}}
```

Example SSC response:

```
{"device":{"network":{"ipv4":{"manual_netmask":["255.255.254.0"]}}}}
```

7.27 "/device/network/ipv4/manual_ipaddr"

List of IPv4 addresses which will be applied after setting "device/network/ipv4/auto" to "false".

- type: String
- default: ["192.168.178.23"]
- const: false
- writeable: true
- count: 1
- subscr: true

Example SSC set:

```
{"device":{"network":{"ipv4":{"manual_ipaddr":["192.168.178.23"]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
      {"manual_ipaddr":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"manual_ipaddr":null}}}}
```

Example SSC response:

```
{"device":{"network":{"ipv4":{"manual_ipaddr":["192.168.178.23"]}}}}
```

7.28 "/device/network/ipv4/manual_gateway"

List of IPv4 gateways which will be applied after setting "device/network/ipv4/auto" to "false".

- type: String
- default: ["192.168.178.1"]
- const: false
- writeable: true
- count: 1
- subscr: true

Example SSC set:

```
{"device":{"network":{"ipv4":{"manual_gateway":["192.168.178.1"]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
      {"manual_gateway":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"manual_gateway":null}}}}
```

Example SSC response:

```
{"device":{"network":{"ipv4":{"manual_gateway":["192.168.178.1"]}}}}
```

7.29 "/device/network/ipv4/ipaddr"

List of current IPv4 addresses.

- type: String
- default: ["192.168.178.23"]
- const: false
- writeable: false
- count: 1
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
    {"ipaddr":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"ipaddr":null}}}}
```

7.30 "/device/network/ipv4/interfaces"

List of current IPv4 interfaces corresponding to the list of interface names in "device/network/ether/interfaces".

- type: Number
- const: true
- writeable: false
- count: 1
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
    {"interfaces":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"interfaces":null}}}}
```

7.31 "/device/network/ipv4/gateway"

List of current IPv4 default gateways.

- type: String
- default: ["192.168.178.1"]
- const: false
- writeable: false
- count: 1
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
    {"gateway":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"gateway":null}}}}
```

7.32 "/device/network/ipv4/auto"

If set to "true" the corresponding interface in the IPv4 interface list will activate it's dhcp client. If no dhcp server is present, link-local addressing will be applied within the link-local range. If set to "false" the "manual_" prefixed settings are taken over to the current settings as long as the "manual_" settings are valid, otherwise the device returns SSC ERROR "406".

- type: Boolean
- default: true
- const: false
- writeable: true
- count: 1
- subscr: true

Example SSC set:

```
{"device":{"network":{"ipv4":{"auto":[true]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":{"ipv4":
    {"auto":null}}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"ipv4":{"auto":null}}}}
```

Example SSC response:

```
{"device":{"network":{"ipv4":{"auto":[true]}}}}
```

7.33 "/device/network/mdns"

If set to true the SSC service and hostname of the device will be published via the MDNS protocol.

- type: Boolean
- default: true
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"network":{"mdns":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"network":
      {"mdns":null}}}]}}}
```

Example SSC request:

```
{"device":{"network":{"mdns":null}}}
```

Example SSC response:

```
{"device":{"network":{"mdns":true}}}
```

7.34 "/device/timeprecision"

The time precision is constantly 1 second.

- type: Number
- const: true
- writeable: false
- max: 1
- min: 1

Example SSC request:

```
{"device":{"timeprecision":null}}
```

7.35 "/device/time"

The system time can be set/read out.

The integer value represents the seconds since January 1, 2000.

Example: `{"device":{"time":582874956}}` will set the date/time to "Thu Jun 21 05:42:36 UTC 2018" - request with `{"device":{"date":null}}`

- type: Number
- const: false
- writeable: true
- min: 0

Example SSC set:

```
{"device":{"time":582874956}}
```

Example SSC request:

```
{"device":{"time":null}}
```

Example SSC response:

```
{"device":{"time":582874956}}
```

7.36 "/device/system"

User settable parameter which can be used freely to store device information.

- type: String
- default: System
- const: false
- writeable: true
- length: 30
- subscr: true

Example SSC set:

```
{"device":{"system":"Device"}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"system":null}}]}}}
```

Example SSC request:

```
{"device":{"system":null}}
```

Example SSC response:

```
{"device":{"system":"Device"}}
```

7.37 "/device/position"

This parameter can be used to define the position of the device in the room. Useful if more than one device is in the same room ("device/location").

The lenght is limited to 30.

The position information must consist of characters (a-z or A-Z) or digits (0-9) or blanks.

- type: String
- default: over central table
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"device":{"position":"middle of meeting room"}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"position":null}}]}}}
```

Example SSC request:

```
{"device":{"position":null}}
```

Example SSC response:

```
{"device":{"position":"middle of meeting room"}}
```

7.38 "/device/name"

The user settable name of the device.

The following rules must be applied:

- The lenght is limited to 8 characters.
- The name must start with a letter (a-z or A-Z).
- The name must end with a letter (a-z or A-Z) or a digit (0-9).
- All other symbols may consist of letters, digits or '-' or '_'.
    - type: String
    - default: SLCM2
    - const: false
    - writeable: true
    - length: 8

- subscr: true

Example SSC set:

```
{"device":{"name":"MIC2_A-1"}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"name":null}}]}}}
```

Example SSC request:

```
{"device":{"name":null}}
```

Example SSC response:

```
{"device":{"name":"MIC2_A-1"}}
```

7.39 "/device/location"

This parameter can be used to define in which room and building the device is located. An UTF-8 string up to 100 characters is supported.

- type: String
- default: Room
- const: false
- writeable: true
- length: 100
- subscr: true

Example SSC set:

```
{"device":{"location":"ROOM_C31"}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"device":{"location":null}}]}}}
```

Example SSC request:

```
{"device":{"location":null}}
```

Example SSC response:

```
{"device":{"location":"ROOM_C31"}}
```

7.40 "/device/language"

The supported language is English (Great Britain).

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"language":null}}
```

7.41 "/device/date"

Date/time stamp of the device - for more details see explanation of "/device/time".

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"device":{"date":null}}
```

7.42 "/device/restore"

FACTORY_DEFAULTS: Reset device to factory defaults - this will also reboot the device!

AUDIO_DEFAULTS: Reset all audio settings to factory defaults - the device will NOT reboot.

DANTE_FACTORY_DEFAULTS: Reset all Dante settings to factory defaults - this will also reboot the device!

- type: String
- const: false
- writeable: true
- options: 1. FACTORY_DEFAULTS 2. AUDIO_DEFAULTS 3. DANTE_FACTORY_DEFAULTS

Example SSC set:

```
{"device":{"restore":"FACTORY_DEFAULTS"}}
```

Example SSC request:

```
{"device":{"restore":null}}
```

Example SSC response:

```
{"device":{"restore":"FACTORY_DEFAULTS"}}
```

7.43 "/device/restart"

Reboots SLCM2

- type: Boolean
- const: false
- writeable: true

Example SSC set:

```
{"device":{"restart":true}}
```

Example SSC request:

```
{"device":{"restart":null}}
```

Example SSC response:

```
{"device":{"restart":true}}
```

7.44 "/interface/version"

Version of the SSC tree.

This includes versioning of the available parameters and their limits.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"interface":{"version":null}}
```

7.45 "/audio/equalizer/preset"

Activate custom settings for the 7-band graphical equalizer.

OFF: Bypass EQ and ignore "/audio/equalizer/custom".

CUSTOM: Activate EQ and "audio/equalizer/custom" will be applied.

- type: String
- default: OFF
- const: false
- writeable: true
- options: 1. OFF 2. CUSTOM
- subscr: true

Example SSC set:

```
{"audio":{"equalizer":{"preset":"CUSTOM"}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"equalizer":
      {"preset":null}}}]}}}
```

Example SSC get:

```
{"audio":{"equalizer":{"preset":null}}}
```

Example SSC response:

```
{"audio":{"equalizer":{"preset":"CUSTOM"}}}
```

7.46 "/audio/equalizer/custom"

Gain settings for the 7-band graphical equalizer.

The cut-off frequencies are: 125, 250, 500, 1000, 2000, 4000, 8000

Filter quality is Q=1.4142

- type: Number
- default: [0,0,0,0,0,0,0]
- const: false
- writeable: true
- count: 7
- units: dB
- max: 8
- min: -8
- inc: 0.5
- subscr: true

Example SSC set:

```
{"audio":{"equalizer":{"custom":[3,6,-3,2,0,-3,-5]}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"equalizer":
      {"custom":null}}}]}}}
```

Example SSC get:

```
{"audio":{"equalizer":{"custom":null}}}
```

Example SSC response:

```
{"audio":{"equalizer":{"custom":[3,6,-3,2,0,-3,-5]}}}
```

7.47 "/audio/exclusion/zones"

Up to 5 audio exclusion zones may be defined. They are mutually independend. Each zone has two min/max angles for elevation in the range from 0 to 90 and two min/max angles for azimuth in the range from 0 to 360 with a step size (increment) of 5. Furthermore the minimal aperture of min/max (elevation as well as azimuth) must be 10. For azimuth any same min/max values like [null, null, 0, 0] or e.g. [null, null, 120, 120] and [null, null, 0, 360] mean a full circle.

To leave an exclusion zone unchanged you may also use ‚null'. The following example only changes zone 1 and 2.

```
TX:
{"audio":{"exclusion":
      {"zones":[[0, 10, 0, 360], [10, 50, 20, 70],
            null, null, null]}}}
RX:
{"audio":{"exclusion":
      {"zones":[[0, 10, 0, 360], [10, 50, 20, 70],
            [10, 50, 110, 160], [10, 50,
                  200, 250], [10, 50, 290, 340]]}}}
```

You may also use null for angles within a zone:

```
TX:
{"audio":{"exclusion":{"zones":
      [null, 15, null, null], [20, 30, 240, 265], null, null, null]
      }}}
```

```
RX:
{"audio":{"exclusion":{"zones":
       [[0, 15, 0, 360], [20, 30, 240, 265],
              [10, 50, 110, 160], [10, 50,
                     200, 250], [10, 50, 290, 340]]}}
```

If limits are set to values which do not meet constraint of minimal aperture the upper value will be adjusted automatically. If the available range is not sufficient the min value will also be adjusted.

```
TX:
{"audio":{"exclusion":
       {"zones":[[80, 85, null, null], null, null, null, null]}}}
RX:
{"audio":{"exclusion":
       {"zones":[[80, 90, 100, 160],
              [20, 30, 240, 265], [10, 50, 110, 160],
                     [10, 50, 200, 250], [10, 50, 290, 340]]
                            }}}
TX:
{"audio":{"exclusion":
       {"zones":[[85, 90, null, null], null, null, null, null]}}}
RX:
{"audio":{"exclusion":
       {"zones":[[80, 90, 100, 160], [20, 30, 240, 265],
              [10, 50, 110, 160], [10, 50, 200, 250],
                     [10, 50, 290, 340]]}}}
```

- type: Number
- default: [[0, 10, 0, 360], [10, 50, 20, 70], [10, 50, 110, 160], [10, 50, 200, 250], [10, 50, 290, 340]]
- const: false
- writeable: true
- count: 5
- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion":
       {"zones":[[0, 10, 0, 360], [10, 50, 20, 70],
              [10, 50, 110, 160], [10, 50, 200, 250],
                     [10, 50, 290, 340]]}}}
```

Example SSC subscr:

```
{"osc":{"state":
       {"subscribe":[ {"audio":{"exclusion":{"zones":null}}}]}}}
```

Example SSC get:

```
{"audio":{"exclusion":{"zones":null}}}
```

Example SSC response:

```
{"audio":{"exclusion":
       {"zones":[[0, 10, 0, 360], [10, 50, 20, 70],
              [10, 50, 110, 160], [10, 50, 200, 250],
                     [10, 50, 290, 340]]}}}
```

7.48 "/audio/exclusion/active"

With these 5 flags each of the according exclusion zone (see "/audio/exclusion/zones") can be set to on/off individually.

Set to ‚true' to activate exclusion zone, set to ‚false' to deactivate exclusion zone. If ‚null' is used the setting for the according exclusion zone remains unchanged.

- type: Boolean
- default: [true, false, false, false, false]

- const: false
- writeable: true
- count: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion":
        {"active":[true, false, false, false, false]}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[
        {"audio":{"exclusion":{"active":null}}}]}}}
```

Example SSC get:

```
{"audio":{"exclusion":{"active":null}}}
```

Example SSC response:

```
{"audio":{"exclusion":
        {"active":[true, false, false, false, false]}}}
```

7.49 "/audio/exclusion_zone/azimuth/3"

HINT: This method is deprecated! Please use "/audio/exclusion/zones" instead.

Provide two angles to set a horizontal exclusion zone. The beam will not point to directions within [azimuth min, azimuth max]. The beam will exclude the union of all configured zones.

- type: Number
- default: [0,0]
- const: false
- writeable: true
- count: 2
- max: 360
- min: 0
- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion_zone":{"azimuth":{"3":[0,120]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"exclusion_zone":
        {"azimuth":{"3":null}}}}]}}}
```

Example SSC request:

```
{"audio":{"exclusion_zone":{"azimuth":{"3":null}}}}
```

Example SSC response:

```
{"audio":{"exclusion_zone":{"azimuth":{"3":[0,120]}}}}
```

"/audio/exclusion_zone/azimuth/2"

HINT: This method is deprecated! Please use "/audio/exclusion/zones" instead.

Provide two angles to set a horizontal exclusion zone. The beam will not point to directions within [azimuth min, azimuth max]. The beam will exclude the union of all configured zones.

- type: Number
- default: [0,0]
- const: false
- writeable: true
- count: 2
- max: 360
- min: 0

- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion_zone":{"azimuth":{"2":[0,120]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"exclusion_zone":
    {"azimuth":{"2":null}}}}]}}}
```

Example SSC request:

```
{"audio":{"exclusion_zone":{"azimuth":{"2":null}}}}
```

Example SSC response:

```
{"audio":{"exclusion_zone":{"azimuth":{"2":[0,120]}}}}
```

7.50 "/audio/exclusion_zone/azimuth/1"

HINT: This method is deprecated! Please use "/audio/exclusion/zones" instead.

Provide two angles to set a horizontal exclusion zone. The beam will not point to directions within [azimuth min, azimuth max]. The beam will exclude the union of all configured zones.

- type: Number
- default: [0,0]
- const: false
- writeable: true
- count: 2
- max: 360
- min: 0
- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion_zone":{"azimuth":{"1":[0,120]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"exclusion_zone":
    {"azimuth":{"1":null}}}}]}}}
```

Example SSC request:

```
{"audio":{"exclusion_zone":{"azimuth":{"1":null}}}}
```

Example SSC response:

```
{"audio":{"exclusion_zone":{"azimuth":{"1":[0,120]}}}}
```

7.51 "/audio/exclusion_zone/elevation/1"

HINT: This method is deprecated! Please use "/audio/exclusion/zones" instead.

Provide two angles to set a vertical exclusion zone. The beam will not point to directions within [elevation min, elevation max]. The beam will exclude the union of all configured zones.

- type: Number
- default: [0,10]
- const: false
- writeable: true
- max: 75
- min: 0
- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"exclusion_zone":{"elevation":{"1":[0,60]}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"exclusion_zone":
      {"elevation":{"1":null}}}}]}}}
```

Example SSC request:

```
{"audio":{"exclusion_zone":{"elevation":{"1":null}}}}
```

Example SSC response:

```
{"audio":{"exclusion_zone":{"elevation":{"1":[0,60]}}}}
```

7.52 "/audio/noise_gate/threshold"

Audio level threshold which triggers the Noise-Gate to open the audio output of the microphone channel. An implicit hysteresis is applied to this threshold for closing the channel.

- type: Number
- units: dB
- max: -40
- min: -90
- subscr: true

Example SSC set:

```
{"audio":{"noise_gate":{"threshold":-50}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"noise_gate":{"thresh
      old":null}}}]}}}
```

Example SSC request:

```
{"audio":{"noise_gate":{"threshold":null}}}
```

Example SSC response:

```
{"audio":{"noise_gate":{"threshold":-50}}}
```

7.53 "/audio/noise_gate/hold_time"

The Hold-Time of the Noise-Gate sets the minimum time the ouput will be open once the threshold has been triggered.

- type: Number
- units: ms
- max: 1000
- min: 50
- subscr: true

Example SSC set:

```
{"audio":{"noise_gate":{"hold_time":450}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"noise_gate":
{"hold_time":null}}}]}}}
```

Example SSC request:

```
{"audio":{"noise_gate":{"hold_time":null}}}
```

Example SSC response:

```
{"audio":{"noise_gate":{"hold_time":450}}}
```

7.54 "/audio/noise_gate/active"

The Noise Gate can be (de-) activated

- type: Boolean

- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"audio":{"noise_gate":{"active":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"noise_gate":
{"active":null}}}]}}}
```

Example SSC request:

```
{"audio":{"noise_gate":{"active":null}}}
```

Example SSC response:

```
{"audio":{"noise_gate":{"active":true}}}
```

7.55 "/audio/out1/label"

Label that can be found on the housing for connector "Analog Out".

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"audio":{"out1":{"label":null}}}
```

7.56 "/audio/out1/desc"

Description for container audio/out1.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"audio":{"out1":{"desc":null}}}
```

7.57 "/audio/out1/attenuation"

Analogue output attenuation for audio/out1.

- type: Number
- default: 0
- const: false
- writeable: true
- units: dB
- max: 0
- min: -18
- subscr: true

Example SSC set:

```
{"audio":{"out1":{"attenuation":-10}}}
```

Example SSC subscribe:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out1":
      {"attenuation":null}}}]}}}
```

Example SSC request:

```
{"audio":{"out1":{"attenuation":null}}}
```

Example SSC response:

```
{"audio":{"out1":{"attenuation":-10}}}
```

7.58 "/audio/out2/identity/version"

Software version of the Dante interface.

- type: String
- const: true
- writeable: false

Example:

```
SSC get: {"audio":{"out2":{"identity":{"version":null}}}}
```

7.59 "/audio/out2/network/ether/macs"

List of Dante MAC addresses.

- type: String
- const: true
- writeable: false
- count: 2

Example: SSC get:

```
{"audio":{"out2":{"network":{"ether":{"macs":null}}}}}
```

7.60 "/audio/out2/network/ether/interfaces"

List of Dante IPv4 interface names.

- type: String
- const: true
- writeable: false
- count: 2

Example SSC get:

```
{"audio":{"out2":{"network":{"ether":{"interfaces":null}}}}}
```

7.61 "/audio/out2/network/ether/interface_mapping"

Dante network port configuration. After changing this setting, the device will automatically reboot for the changes to take effect.

- type: String
- const: false
- writeable: true
- options: 1. AUDIO_SWITCHED 2. AUDIO_REDUNDANT
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"net
       work":{"ether":{"interface_mapping":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ether":{"interface_mapping":null}}}}}
```

7.62 "/audio/out2/network/ipv4/netmask"

List of current IPv4 netmasks of the Dante primary and secondary interfaces.

- type: String
- const: false
- writeable: false

- count: 2
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"net
work":{"ipv4":{"netmask":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"netmask":null}}}}}
```

7.63 "/audio/out2/network/ipv4/manual_ipaddr"

List of IPv4 addresses which will be applied after setting audio/out2/network/ipv4/auto to false

- type: String
- default: ["169.254.10.10","172.31.10.10"]
- const: false
- writeable: true
- count: 2
- subscr: true

Example SSC set:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_ipaddr":
        ["169.254.10.20","172.31.10.20"]}}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"network":
        {"ipv4":{"manual_ipaddr":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_ipaddr":null}}}}}
```

Example SSC response:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_ipaddr":
        ["169.254.10.20","172.31.10.20"]}}}}}
```

7.64 "/audio/out2/network/ipv4/manual_gateway"

List of IPv4 gateways which will be applied after setting audio/out2/network/ipv4/auto to false

- type: String
- default: ["169.254.2.1","172.31.2.2"]
- const: false
- writeable: true
- count: 2
- subscr: true

Example SSC set:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_gateway":
        ["169.254.1.1","172.31.1.2"]}}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"network":
        {"ipv4":{"manual_gateway":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_gateway":null}}}}}
```

Example SSC response:

```
{"audio":{"out2":{"network":{"ipv4":{"manual_gateway":
        ["169.254.1.1","172.31.1.2"]}}}}}
```

7.65 "/audio/out2/network/ipv4/ipaddr"

List of current IPv4 addresses of the Dante primary and secondary interfaces.

- type: String
- const: false
- writeable: false
- count: 2
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[
        {"audio":{"out2":{"network":{"ipv4":{"ipaddr":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":
        {"ipv4":{"ipaddr":null}}}}}
```

7.66 "/audio/out2/network/ipv4/interfaces"

List of current IPv4 interfaces corresponding to the list of interface names in `audio/out2/network/ether/interfaces`.

- type: `Number`
- const: `true`
- writeable: `false`
- count: 2
- subscr: `true`

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"network":
        {"ipv4":{"interfaces":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"interfaces":null}}}}}
```

7.67 "/audio/out2/network/ipv4/gateway"

List of current IPv4 default gateways of the Dante primary and secondary interfaces.

- type: String
- const: false
- writeable: false
- count: 2
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":
        {"network":{"ipv4":{"gateway":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"gateway":null}}}}}
```

7.68 "/audio/out2/network/ipv4/auto"

If set to true the corresponding interface in the IPv4 interface list will activate it's dhcp client. If no dhcp server is present, link-local addressing will be applied within the link-local range. If set to false the "manual_" prefixed settings are taken over to the current settings as long as the "manual_" settings are valid, otherwise the device returns SSC ERROR 406. After changing the ip configuration, the device will automatically reboot for the changes to take effect. Please refer to the Dante Controller Manual for more information regarding specific configuration edge cases.

- type: Boolean
- default: [true,true]
- const: false
- writeable: true
- count: 2

- subscr: true

Example SSC set:

```
{"audio":{"out2":{"network":{"ipv4":{"auto":[true,false]}}}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"network":
      {"ipv4":{"auto":null}}}}}]}}}
```

Example SSC get:

```
{"audio":{"out2":{"network":{"ipv4":{"auto":null}}}}}
```

Example SSC response:

```
{"audio":{"out2":{"network":{"ipv4":{"auto":[true,false]}}}}}
```

7.69 "/audio/out2/label"

Label that can be found on the housing for connector audio/out2.

- type: String
- const: true
- writeable: false

Example SSC get:

```
{"audio":{"out2":{"label":null}}}
```

7.70 "/audio/out2/gain"

Dante output gain for audio/out2.

- type: Number
- default: 12
- const: false
- units: dB
- max: 24
- min: 0
- subscr: true

Example SSC set:

```
{"audio":{"out2":{"gain":-10}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"out2":{"gain":null}}}]}}}
```

Example SSC request:

```
{"audio":{"out2":{"gain":null}}}
```

Example SSC response:

```
{"audio":{"out2":{"gain":-10}}}
```

7.71 "/audio/out2/desc"

Description for container audio/out2.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"audio":{"out2":{"desc":null}}}
```

7.72 "/audio/priority/zones"

Zone with prioritized sources for beam-steering: [elevMin,elevMax,aziMin,aziMax]

- type: Number

- default: [60, 80, 160, 200]
- const: false
- writeable: true
- count: 1
- inc: 5
- subscr: true

Example SSC set:

```
{"audio":{"priority":{"zones":[60, 80, 160, 200]}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"priority":
    {"zones":null}}}]}}}
```

Example SSC request:

```
{"audio":{"priority":{"zones":null}}}
```

Example SSC response:

```
{"audio":{"priority":{"zones":[60, 80, 160, 200]}}}
```

7.73 "/audio/priority/weights"

Priority zone weights (1.0 = neutral)

- type: Number
- default: [1.5]
- const: false
- writeable: true
- count: 1
- max: 4
- min: 1
- inc: 0.1
- subscr: true

Example SSC set:

```
{"audio":{"priority":{"weights":[1.3]}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"priority":
    {"weights":null}}}]}}}
```

Example SSC request:

```
{"audio":{"priority":{"weights":null}}}
```

Example SSC response:

```
{"audio":{"priority":{"weights":[1.3]}}}
```

7.74 "/audio/priority/active"

Activate priority zone. Hint: The boolean value can be set in array format, e.g. "[false]" or directly, e.g. "false". The response will allways be in array format.

- type: Boolean
- default: false
- const: false
- writeable: true
- count: 1
- subscr: true

Example SSC set:

```
{"audio":{"priority":{"active":[false]}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"priority":
      {"active":null}}}]}}}
```

Example SSC request:

```
{"audio":{"priority":{"active":null}}}
```

Example SSC response:

```
{"audio":{"priority":{"active":[false]}}}
```

7.75 "/audio/ref1/label"

Label that can be found on the housing for connector "Dante Out".

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"audio":{"ref1":{"label":null}}}
```

7.76 "/audio/ref1/gain"

AEC Reference gain.

- type: Number
- default: 0
- units: dB
- max: 10
- min: -60
- subscr: true

Example SSC set:

```
{"audio":{"ref1":{"gain":0}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"ref1":{"gain":null}}}]}}}
```

Example SSC request:

```
{"audio":{"ref1":{"gain":null}}}
```

Example SSC response:

```
{"audio":{"ref1":{"gain":0}}}
```

7.77 "/audio/ref1/farend_auto_adjust_enable"

Enable automatic threshold adjustment based on noise floor estimation (nfe).

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"audio":{"ref1":{"farend_auto_adjust_enable":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"ref1":
{"farend_auto_adjust_enable":null}}}]}}}
```

Example SSC request:

```
{"audio":{"ref1":{"farend_auto_adjust_enable":null}}}
```

Example SSC response:

```
{"audio":{"ref1":{"farend_auto_adjust_enable":true}}}
```

7.78 "/audio/ref1/desc"

Description for container audio/ref1.

- type: String
- const: true
- writeable: false

Example SSC request:

```
{"audio":{"ref1":{"desc":null}}}
```

7.79 "/audio/source_detection/threshold"

Threshold for detecting the speaker depending on the room noise level.

- type: String
- default: normal_room
- const: false
- writeable: true
- options: 1. quiet_room 2. normal_room 3. loud_room
- subscr: true

Example SSC set:

```
{"audio":{"source_detection":{"threshold":"quiet_room"}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"source_detection":
      {"threshold":null}}}]}}}
```

Example SSC request:

```
{"audio":{"source_detection":{"threshold":null}}}
```

Example SSC response:

```
{"audio":{"source_detection":{"threshold":"quiet_room"}}}
```

7.80 "/audio/voice_lift/emergency_mute_time"

Time for how long the mic should be muted after the Emergency Mute Threshold has been exceeded.

- type: Number
- units: s
- max: 30
- min: 1
- inc: 1
- subscr: true

Example SSC set:

```
{"audio":{"voice_lift":{"emergency_mute_time":4.0}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"voice_lift":{"emergency_
mute_time":null}}}]}}}
```

Example SSC request:

```
{"audio":{"voice_lift":{"emergency_mute_time":null}}}
```

Example SSC response:

```
{"audio":{"voice_lift":{"emergency_mute_time":4.0}}}
```

7.81 "/audio/voice_lift/emergency_mute_threshold"

The Voice-Lift mode has an inbuilt automatic mute function which will temporarily shut down the output in case the microphone level exceeds the set threshold value.

- type: Number

- default: -20
- const: false
- writeable: true
- units: dB
- max: -3
- min: -50
- subscr: true

Example SSC set:

```
{"audio":{"voice_lift":{"emergency_mute_threshold":-30}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"voice_lift":{"emergency_
mute_threshold":null}}}]}}}
```

Example SSC request:

```
{"audio":{"voice_lift":{"emergency_mute_threshold":null}}}
```

Example SSC response:

```
{"audio":{"voice_lift":{"emergency_mute_threshold":-30}}}
```

7.82 "/audio/voice_lift/active"

Activates certain algorithms to the microphone output signal in order to mitigate the risk of feedback from the loudspeakers.

Function

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"audio":{"voice_lift":{"active":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"voice_lift":{"ac-
tive":null}}}]}}}
```

Example SSC request:

```
{"audio":{"voice_lift":{"active":null}}}
```

Example SSC response:

```
{"audio":{"voice_lift":{"active":true}}}
```

7.83 "/audio/room_in_use"

Coarse Near-End-Activity-Flag based on microphone level.

- type: Boolean
- const: false
- writeable: false
- subscr: true

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"room_in_use":null}}]}}}
```

Example SSC request:

```
{"audio":{"room_in_use":null}}
```

7.84 "/audio/mute"

Mute state of the microphone. Set to true to mute the audio outputs.

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"audio":{"mute":true}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":{"mute":null}}]}}}
```

Example SSC request:

```
{"audio":{"mute":null}}
```

Example SSC response:

```
{"audio":{"mute":true}}
```

7.85 "/audio/installation_type"

Type of installation of the microphone. The audio signal will be optimized for the choosen installation type.

- flush_mount: The SLCM2 is mounted directly on the ceiling.
- suspended: The SLCM2 is mounted suspended from the ceiling (at least 30 cm).
  - type: String
  - default: flush_mount
  - options:
    1. flush_mount
    2. suspended
  - subscr: true

Example SSC set:

```
{"audio":{"installation_type":"flush_mount"}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"audio":
      {"installation_type":null}}]}}}
```

Example SSC request:

```
{"audio":{"installation_type":null}}
```

Example SSC response:

```
{"audio":{"installation_type":"flush_mount"}}
```

7.86 "/beam/orientation/visual"

Two LEDs will indicate the orientation of the device (green: right side, red: left side). The indicated direction shows 0 degrees. Adjust the 0 degree reference by changing beam/orientation/offset.

- type: Boolean
- default: false
- const: false
- writeable: true
- subscr: true

Example SSC set:

```
{"beam":{"orientation":{"visual":true}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"beam":{"orientation":
      {"visual":null}}}]}}}
```

Example SSC request:

```
{"beam":{"orientation":{"visual":null}}}
```

Example SSC response:

```
{"beam":{"orientation":{"visual":true}}}
```

7.87 "/beam/orientation/offset"

All beam azimuth values can be adjusted by this offset to match the installation of the device to the room.

- type: Number
- default: 0
- const: false
- writeable: true
- options 1. 0 2. 90 3. 180 4. 270
- subscr: true

Example SSC set:

```
{"beam":{"orientation":{"offset":180}}}
```

Example SSC subscr:

```
{"osc":{"state":{"subscribe":[ {"beam":{"orientation":
      {"offset":null}}}]}}}
```

Example SSC request:

```
{"beam":{"orientation":{"offset":null}}}
```

Example SSC response:

```
{"beam":{"orientation":{"offset":180}}}
```

7.88 "/osc/state/auth/access"

Reflects access rights to SSC methods. "default:/" means default rights from root and lower.

- type: String

- const: false

- writeable: false

Example SSC get:

```
{"osc":{"state":{"auth":{"access":null}}}}
```

7.89 "/osc/state/prettyprint"

SSC reply output style is not supported. Returns false.

Example SSC request:

```
{"osc":{"state":{"prettyprint":null}}}
```

7.90 "/osc/state/close"

SSC connection close.

Example SSC request:

```
{"osc":{"state":{"close":null}}}
```

7.91 "/osc/state/subscribe"

See chapter "SSC subscriptions".

Example SSC request:

```
{"osc":{"state":{"subscribe":null}}}
```

7.92 "/osc/feature/timetag"

SSC timed method execution is not supported. Returns false.

Example SSC request:

```
{"osc":{"feature":{"timetag":null}}}
```

7.93 "/osc/feature/baseaddr"

SSC interactive method address base is not supported. Returns false.

Example SSC request:

```
{"osc":{"feature":{"baseaddr":null}}}
```

7.94 "/osc/feature/subscription"

SSC subscriptions are supported. Returns true.

Example SSC request:

```
{"osc":{"feature":{"subscription":null}}}
```

7.95 "/osc/feature/pattern"

SSC message dispatching and pattern matching are supported. Returns "*?".

Example SSC request:

```
{"osc":{"feature":{"pattern":null}}}
```

7.96 "/osc/limits"

SSC method parameter range reflection.

Example SSC request:

```
{"osc":{"limits":null}}
```

7.97 "/osc/schema"

SSC schema reflection.

Example SSC request:

```
{"osc":{"schema":null}}
```

7.98 "/osc/version"

SSC protocol version.

Example SSC request:

```
{"osc":{"version":null}}
```

7.99 "/osc/xid"

SSC transaction ID.

Example SSC request:

```
{"osc":{"xid":null}}
```

7.100 "/osc/ping"

SSC Ping.

Example SSC request:

```
{"osc":{"ping":null}}
```

7.101 "/osc/error"

SSC error state.

Example SSC request:

```
{"osc":{"error":null}}
```

8. SSC String characters

- ASCII 32...126 excluding escaped characters ASCII 34(‘”’), ASCII 47(‘/’) and ASCII 92(‘\’).
- No escaped primitives ‘\b’, ‘\f’, ‘\r’ ,’\n’, ‘\t’
- No unicode patterns ‘\uxxxx’

9. SSC Error List

- 100 : continue
- 102 : processing
- 200 : OK
- 201 : created
- 202 : adapted
- 210 : partial success
- 310 : subscription terminates
- 400 : message not understood
- 401 : authorisation needed
- 403 : forbidden
- 404 : address not found
- 406 : not acceptable
- 408 : request time out
- 409 : conflict
- 410 : gone
- 413 : request too long
- 414 : request too complex
- 416 : requested range not satisfiable
- 422 : unprocessable entity
- 423 : locked
- 424 : failed dependency
- 450 : answer too long
- 454 : parameter address not found
- 500 : internal server error
- 501 : not implemented
- 503 : service unavailable