Sennheiser Sound Control Protocol (SSC)

versatile command, control, and configuration for networked audio systems

Developer‘s guide for SpeechLine Digital Wireless

SL Rack Receiver DW
CHG 4N

Sennheiser electronic GmbH & Co. KG

Am Labor 1, 30900 Wedemark, Germany, www.sennheiser.com
TI 1094 v2.2

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)
- Sennheiser Control Cockpit
- remote channel for Sennheiser 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 division 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/ .

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 (MIME-type *application/json*).

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.

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, 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 "/"
        audio/                      container at address "/audio/"
                out1/               container at address "/audio/out1/"
                        label name  address "/audio/out1/label": method with string argument
                        gain_db 5   address "/audio/out1/gain_db": method with numeric argument
                        ...         more methods of  "/audio/out1"
                out2/               container at address "/audio/out2 /"
                ...                 methods of "/audio/out2"
        device/                     container at address "/device/"
                ...                 methods of "/device"
        ...                         more methods and containers of  "/"
```

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

```
{                                       root object
        "audio": {                      object "audio"
                "out1": {               object "audio.out1"
                        "label": name,  numerical property "audio.out1.label"
                        "gain_db": 5,   boolean property  "audio.out1.gain_db"
                        ...             more properties of  "audio.out1"
                },
                "out2": {               object "audio.out2"
                        ...             properties of "audio.out2"
                },
        "device": {                     object "device"
                ...                     properties of "device"
        },
        ...                             more properties and objects of  the root object
}
```

The OSC Method Address (like "/audio/out1/gain_db") 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 a SSC Server that are not mentioned in a JSON message are left unchanged.

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

```
{ "audio": { "out1": { "gain_db": 5 } } }
```

can be understood as a SSC Method Call of the SSC Method "/audio/out1/gain_db" 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 a SSC Message that a SSC Client is sending to a SSC Server.

Prefix "`RX:`" indicates a SSC Message that the SSC Server will send back to the Client.

A SSC-Message is written verbatim, enclosed by curly brackets `{ }`.

A transaction to set the gain of "audio" of "out1" to 1 then looks like this:

```
TX: { "audio": { "out1": { "gain_db": 1 }}}
RX: { "audio": { "out1": { "gain_db": 1 }}}
```

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: { "audio": { "out1": { "gain_db": 20000 }}}
RX: { "audio": { "out1": { "gain_db": 6 }}}
```

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: { "audio": { "out1": { "gain_db": null }}}
RX: { "audio": { "out1": { "gain_db": 6 }}}
```

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: { "audio": { "out1": { "gain_db": 3, "label": null }}}
RX: { "audio": { "out1": { "gain_db": 3, "label": "AF_LABEL" }}}
```

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.

4.3.2 SSC Messages

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

A 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 `{ }`.

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" a 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 a SSC Method Reply Message to the originating SSC Client.

A SSC Server's SSC Methods are arranged in a tree structure called a SSC Address Space. The leaves of this tree are the SSC Methods and the branch nodes are called SSC Containers. A 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 a 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. A 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.

A 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 a SSC Server SHOULD normally have arguments.

Example:

- query current gain of OUT1 output of AUDIO module:

```
TX: { "audio": { "out1": { "gain_db": null }}}
RX: { "audio": { "out1": { "gain_db": 5 }}}
```

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

```
TX: { "audio": { "out1": { "gain_db": 999 }}}
RX: { "audio": { "out1": { "gain_db": 6 }}}
```

5. 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 a 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 a 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.

5.1 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, default 1000
- "lifetime" maximum lifetime (s) of the subscription, default 10s

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.

5.2 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 the transaction example.

The SSC Server MAY also send a 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.

5.3 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 a 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 a SSC Address Tree specifying the SSC addresses that the SSC Client requests to subscribe. The SSC Address Tree MAY contain Address patterns.

A 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 a 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.

5.4 Subscription example transactions

5.4.1 Subscription request, reply and notifications, automatically terminated:

```
TX: { "osc": { "state": { "subscribe": [
       { "audio": { "out1": { "level_db": null }}} ] }}}
RX: { "osc": { "state": { "subscribe": [
        { "audio": { "out1": { "level_db": null }}} ] }}}
RX: { "audio": { "out1": { "level_db": -10 }}}
...
RX: { "audio": { "out1": { "level_db": -13 }}}
...
RX: { "audio": { "out1": { "level_db": -23 }}}
...
RX: { "osc": { "error": [
          { "audio": { "out1": { "level_db": [
              310, { "desc": "subscription terminates" }] }}}
```

5.4.2 Subscription request with chosen timeout (2 minutes):

```
TX: { "osc": { "state": { "subscribe": [
              "#": { "lifetime": 120 },
       "audio": { "out1": { "level_db": null }}
   } ] }}}
RX: { "osc": { "state": { "subscribe": [
              "#": { "lifetime": 120 },
              "audio": { "out1": { "level_db": null }}
   } ] }}}
RX: { "audio": { "out1": { "level_db": -9 }}}
...
RX: { "audio": { "out1": { "level_db": -14 }}}
...
RX: { "audio": { "out1": { "level_db": -26 }}}
...
RX: { "osc": { "error": [
          { "audio": { "out1": { "level_db": [
              310, { "desc": "subscription terminates" }] }}}
       ] }}
```

5.4.3 Client cancelling the subscription of the previous example:

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

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.

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 SSC Server Discovery

Networked devices implement DNS-SD (Apple Bonjour) as discovery protocol.

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 must be identical to the device name accessible as /device/name. DNS-SD automatic name collision resolution SHOULD be performed, and the resulting name changes MUST be reflected back into /device/name and the persistent device configuration. The renaming rules MAY be tailored to suit product specific requirements.

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. It may contain a unique identification part (e.g., derived from the device MAC or serial), to avoid name collisions and automatic renaming.

Additional information about the SSC Server may be provided with a DNS-SD TXT-record.

The following properties are currently defined for the TXT record:

- txtvers Version of the TXT record format. Currently "1".
- version SSC-Version provided by the SSC Server.

A 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.

7. Developer’s Guide for SL Rack Receiver DW

This chapter describes in detail how a developer should use the SSC interface as implemented for the SL Rack Receiver DW.

7.1 Limitations

7.1.1 SSC Transport Layer

The SSC Server implemented for SpeechLine DW devices supports only UDP/IP as transport protocol. All the devices support both IPv4 and IPv6.

7.1.2 Subscriptions

SpeechLine DW receivers support SSC Method subscriptions from up to eight different SSC clients simultaneously.

8. SSC Method List (SL Rack Receiver DW)

8.1 /interface/version

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: SSC interface version.

Example:

```
TX: {"interface":{"version":null}}
RX: {"interface":{"version":"1.0"}}
```

8.2 /osc/xid

Parameter type: N.A.

Permission: N.A.

Pre-condition: N.A.

Post-condition: N.A.

Description: When an SSC Client 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.

Examples:

```
TX: {"osc":{"xid":1234567890,"ping":["AbCdEfGhIjKlMnOpQrStUvWxYz",3,
    14159265358979323846264338327950]}}
RX: {"osc":{"ping":["AbCdEfGhIjKlMnOpQrStUvWxYz",3,
    14159265358979323846264338327950],"xid":1234567890}}
TX: {"osc":{"xid":1234567890},"brightness":null}
RX: {"brightness":75,"osc":{"xid":1234567890}}
```

8.3 /osc/version

Parameter type: String.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: Reports the SSC version implemented in the server.

Example:

```
TX: {"osc":{"version":null}}
RX: {"osc":{"version":"1.0"}}
```

8.4 /osc/error

Parameter type: Number (limit range)

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

The error message MUST contain an integer numeric value, the error code. The error code SHOULD be chosen from the SSC Error List detailed in chapter ""9. SSC Error List (SL Rack Receiver DW)"".

Examples:

```
TX: {"out1":{"gain":10}}
RX: {"osc":{"error":[{{"out1":[404]}}]}}     <-- not found
TX: {"write_protection":true}
RX: {"osc":{"error":[{"write_protection":[406]}]}}    <-- read only
TX: {"osc":{"limits":[{"internal":{"debug_screen":null}}]}}
RX: {"osc":{"error":[{"osc":{"limits":[454]}}]}}    <-- hidden
```

8.5 /osc/schema

Parameter type: N.A.

Permission: N.A..

Pre-condition: N.A.

Post-condition: N.A.

Description: The /osc/schema method exists to allow clients to query servers about what address schemes are available on a specific server. SSC clients MUST be able to understand both bundled and unbundled replies. 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.

Examples:

```
TX: {"osc":{"schema":null}}
RX: {"osc":{"schema":[{"audio":{},"device":{},"mates":{},"rx1":{},
    "osc":{},"brightness":null}]}}
TX: {"osc":{"schema":[{"rx1":null}]}}
RX: {"osc":{"schema":[{"rx1":{"warnings":null,"walktest":null,"rf_quality"
    :null,"pair":null,"mute_switch_active":null,"identify":null,"autolock"
    :null}}]}}
```

8.6 /osc/limits

Parameter type: N.A.

Permission: N.A..

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

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

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
- units string String describing value units (preferably SI)
- desc string descriptive text, meant for display to the user
- option 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.

Examples:

```
TX: {"osc":{"limits":[{"brightness":null}]}}
RX: {"osc":{"limits":[{"brightness":[{"type":"Number","max":100,"min":0,
    "inc":1,"units":"%"}]}]}}
TX: {"osc":{"limits":[{"audio":{"equalizer":{"preset":null}}}]}}
RX: {"osc":{"limits":[{"audio":{"equalizer":{"preset":[{"type":"Number",
    "desc":"EQ presets","option":[0,1,2,3,4],"option_desc":["Off",
    "Female Speech","Male Speech","Instr. / Media","Custom"]}]}}}]}}
```

8.7 /osc/feature/pattern

Parameter type: String.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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

Example:

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

8.8 /osc/feature/baseaddr

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: Using a baseaddr helps to explore a device in a truly interactive manner, and may additionally be used to reduce message lengths by shortening addresses in SSC requests.

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

Example:

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

8.9 /osc/feature/subscription

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

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

Example:

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

8.10 /osc/feature/timetag

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

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

Example:

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

8.11 /osc/state/subscribe

Parameter type: N.A.

Permission: N.A.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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. Without a lifetime value (unit is seconds), the default lifetime for a subscription to be active is 10 seconds.

Examples 1 – with default parameter:

```
TX: {"osc":{"xid":1234567890,"state":{"subscribe":[{"brightness":null}]}}}
RX: {"osc":{"xid":1234567890},"osc":{"state":{"subscribe":
    [{"brightness":null}]}}}
RX: {"brightness":75}
```

subscription terminates after 10 seconds with:

```
RX: {"osc":{"error":[{"brightness":[310]}]}}
```

Example 2 – with non-default parameter:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"brightness"
    :null}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"brightness"
    :null}]}}}
RX: {"brightness":75}
```

Parameter value changes:

```
RX: {"brightness":70}
RX: {"brightness":75}
RX: {"brightness":70}
RX: {"brightness":65}
```

subscription terminates at end of subscription lifetime with:

```
RX: {"osc":{"error":[{"brightness":[310]}]}}
```

Example 3 – multi-parameter with non-default parameter:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"rx1":
    {"mute_switch_active":null},"mates":{"tx1":{"bat_lifetime":null,
    "bat_gauge":null,"switch1":{"state":null}}}}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"rx1":
    {"mute_switch_active":null},"mates":{"tx1":{"bat_gauge":null}},
    "mates":{"tx1":{"bat_lifetime":null}},"mates":{"tx1":{"switch1":
    {"state":null}}}}]}}}
RX: {"rx1":{"mute_switch_active":false},"mates":{"tx1":{"bat_gauge":83}},
    "mates":{"tx1":{"bat_lifetime":38460}},"mates":{"tx1":{"switch1":
    {"state":true}}}}
```

subscription terminates at end of subscription lifetime with:

```
RX: {"osc":{"error":[{"rx1":{"mute_switch_active":[310]},"mates":{"tx1":
    {"bat_gauge":[310]}},"mates":{tx1":{bat_lifetime":[310]}},"mates":
    {"switch1":{"state":[310]}}}}]}}
```

8.12 /osc/state/close

Parameter type: Boolean.

Permission: Write only.

Pre-condition: N.A.

Post-condition: N.A.

Description: When an SSC Client calls this SSC Method with a true argument, the SSC Server MUST close the connection immediately after the reply has been sent.

Example:

```
TX: {"osc":{"state":{"close":true}}}
RX: {"osc":{"state":{"close":true}}}
```

<Server closes connection>

8.13 /osc/state/prettyprint

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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

Examples:

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

8.14 /device/name

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: 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; if it has a menu, this name should be configurable.

If the device is networked, this name shall be used as the name for device 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.

Example:

```
TX: {"device":{"name":null}}
RX: {"device":{"name":"SLDW"}}
```

8.15 /device/group

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: A reduced charset is supported: ‘0’-‘9’, ‘-‘, ‘_’, ‘A’-‘Z’ and ‘a’-‘z’, and following rules should be followed:

1. A group name SHOULD start only with a letter.
2. A group name MAY end with a letter or a number.
3. A group name SHOULD NOT start or end with a ‘-’ (dash) or ‘_’ (underscore).
4. A group name MAY be up to 8 characters.

Post-condition: The change has immediately effect.

Description: Method to retrieve/modify the group name. This parameter is related to the sRX1 UI menu item "Location".

Example:

```
TX: {"device":{"group":null}}
RX: {"device":{"group":"room"}}
```

8.16 /device/language

Parameter type: String

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: 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. 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.

Example:

```
TX: {"device":{"language":null}}
RX: {"device":{"language":["en_GB"]}}
```

8.17 /device/identity/product

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Product identification, should be identical to the designation on the label on the product itself or to the included electronic/component.

Example:

```
TX: {"device":{"identity":{"product":null}}}
RX: {"device":{"identity":{"product":"SLDW"}}}
```

8.18 /device/identity/version

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Device firmware version.

Example:

```
TX: {"device":{"identity":{"version":null}}}
RX: {"device":{"identity":{"version":"2.1.0"}}}
```

8.19 /device/identity/serial

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Unique product number, identical to the number on a product label. The value is given by production.

Example:

```
TX: {"device":{"identity":{"serial":null}}}
RX: {"device":{"identity":{"serial":"1454100930"}}}
```

8.20 /device/identity/vendor

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Device vendor name.

Example:

```
TX: {"device":{"identity":{"vendor":null}}}
RX: {"device":{"identity":{"vendor":"Sennheiser electronic GmbH & Co. KG"
    }}}
```

8.21 /device/network/ether/interfaces

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of all the user-relevant Ethernet interfaces of the device.

Example:

```
TX: {"device":{"network":{"ether":{"interfaces":null}}}}
RX: {"device":{"network":{"ether":{"interfaces":["LAN"]}}}}
```

8.22 /device/network/ether/macs

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

Example:

```
TX: {"device":{"network":{"ether":{"macs":null}}}}
RX: {"device":{"network":{"ether":{"macs":["00:1B:66:7D:F8:99"]}}}}
```

8.23 /device/network/ipv4/interfaces

Parameter type: Number

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array relating to /device/network/ether/interfaces. The array contains numbers indexing into the array of physical interface names. The interface index array contains the indices of all physical interfaces which share the IPv4 configuration.

Example:

```
TX: {"device":{"network":{"ipv4":{"interfaces":null}}}}
RX: {"device":{"network":{"ipv4":{"interfaces":[1]}}}}
```

8.24 /device/network/ipv4/auto

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: Array containing a list of boolean values that indicates whether IPv4 shall be configured automatically by means of DHCP and ZeroConf (Auto-IP) of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. If the value is set to false the values of /device/network/ipv4/fixed_ipaddr, /device/network/ipv4/fixed_ipaddr and /device/network/ipv4/fixed_ipaddr will be used during target initialization at start-up. A value change takes effect after device reset!

Example:

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

8.25 /device/network/ipv4/ipaddr

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the current IPv4 addresses of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"ipaddr":null}}}}
RX: {"device":{"network":{"ipv4":{"ipaddr":["192.168.1.203"]}}}}
```

8.26 /device/network/ipv4/netmask

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the current IPv4 netmasks of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 netmasks are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"netmask":null}}}}
RX: {"device":{"network":{"ipv4":{"netmask":["255.255.255.0"]}}}}
```

8.27 /device/network/ipv4/gateway

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the IPv4 gateways of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 gateways are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"gateway":null}}}}
RX: {"device":{"network":{"ipv4":{"gateway":["192.168.1.1"]}}}}
```

8.28 /device/network/ipv4/fixed_ipaddr

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 addresses in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4

addresses in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_ipaddr":["192.168.1.203"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_ipaddr":["192.168.1.203"]}}}}
```

8.29 /device/network/ipv4/fixed_netmask

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 netmasks in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4 netmasks in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_netmask":["255.255.255.0"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_netmask":["255.255.255.0"]}}}}
```

8.30 /device/network/ipv4/fixed_gateway

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 gateways in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4 gateways in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_gateway":["192.168.1.1"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_gateway":["192.168.1.1"]}}}}
```

8.31 /device/network/ipv6/interfaces

Parameter type: Number

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array relating to /device/network/ether/interfaces. The array contains numbers indexing into the array of physical interface names. The interface index array contains the indices of all physical interfaces which share the IPv6 configuration.

Example:

```
TX: {"device":{"network":{"ipv6":{"interfaces":null}}}}
RX: {"device":{"network":{"ipv6":{"interfaces":[1]}}}}
```

8.32 /device/network/ipv6/ipaddr

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing an arrary of all the IPv6 addresses of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv6 addresses are specified as strings in standard notation. Each IPv6 network interface can contain maximal 3 IPv6 addresses.

Example:

```
TX: {"device":{"network":{"ipv6":{"ipaddr":null}}}}
RX: {"device":{"network":{"ipv6":{"ipaddr":[["fe80::21b:66ff:fe7d
    :f899","2001:db8::b2f4:4bff:fa71:1a56"]]}}}}
```

8.33 /device/network/mdns_responder

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has effect after restart of the device.

Description: Method to retrieve/modify the setting to enable/disable mDNS. This parameter is related to the sRX1 UI menu item "mDNS" under "Network Settings".

Example:

```
TX: {"device":{"network":{"mdns_responder":null}}}
RX: {"device":{"network":{"mdns_responder":true}}}
```

8.34 /device/state

Parameter type: Number (limit range)

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the state of the sRX1.

Index	Description
0	Normal
1	Pairing
2	Receiver Update
3	Transmitter Update
4	Transmitter Update Confirmation
5	Transmitter Update Failed

Examples:

```
TX: {"device":{"state":null}}
RX: {"device":{"state":4}}
```

8.35 /device/progress

Parameter type: Number (limit range)

Permission: Read only, Subscribe-able

Pre-condition: The sRX1 must be in pairing or (receiver/transmitter) update state.

Post-condition: N.A.

Description: Method to retrieve the displayed progress bar in the sRX1 UI. The parameter value is a defined range [0..100].

Example:

```
TX: {"device":{"progress":null}}
RX: {"device":{"progress":4}}
```

8.36 /device/update/confirmation

Parameter type: Boolean

Permission: Read/Write

Pre-condition: The sRX1 must be in transmitter update confirmation state.

Post-condition: The change has immediately effect.

Description: Method to approve or reject a transmitter update.

Example:

```
TX: {"device":{"update":{"confirmation":true}}}
RX: {"device":{"update":{"confirmation":true}}}
```

8.37 /device/reset

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: A soft reset may be necessary for some configuration settings to take effect.

Example:

```
TX: {"device":{"reset":true}}
RX: {"device":{"reset":true}}
```

8.38 /device/factory_reset

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: The device will restart itself so that after restart the new settings are used.

Description: Boolean value to re-configure the device with factory defaults.

Example:

```
TX: {"device":{"factory_reset":true}}
RX: {"device":{"factory_reset":true}}
```

8.39 /rx1/identify

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: The sRX1 should not be in pairing or walk-test mode.

Post-condition: The change has immediately effect.

Description: Method to start/stop the identify mode on the sRX1.

Example:

```
TX: {"rx1":{"identify":true}}
RX: {"rx1":{"identify":true}}
```

8.40 /rx1/pair

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: The sRX1 should not be in walk-test mode.

Post-condition: The change has immediately effect.

Description: Method to start/stop the pair mode on the sRX1.

Example:

```
TX: {"rx1":{"pair":true}}
RX: {"rx1":{"pair":true}}
```

8.41 /rx1/rf_quality

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to the RF quality in percentage.

Example:

```
TX: {"rx1":{"rf_quality":null}}
RX: {"rx1":{"rf_quality":50}}
```

8.42 /rx1/rf_stack_active

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has immediately effect.

Description: Method to enable/disable the RF stack on the receiver.

Example:

```
TX: {"rx1":{"rf_stack_active":null}}
RX: {"rx1":{"rf_stack_active ":true}}
```

8.43 /rx1/walktest

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: The sRX1 should not be in pairing or identify mode.

Post-condition: The change has immediately effect.

Description: Method to start/stop the walk-test mode on the sRX1.

Example:

```
TX: {"rx1":{"walktest":true}}
RX: {"rx1":{"walktest":true}}
```

8.44 /rx1/mute_switch_active

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has immediately effect.

Description: Method to enable/disable the mute switch functionality on the transmitter, if a mute switch is available. This parameter is related to the sRX1 UI menu item "Mute Switch" under "System Settings".

Example:

```
TX: {"rx1":{"mute_switch_active":false}}
RX: {"rx1":{"mute_switch_active":false}}
```

8.45 /rx1/sync_info

Parameter type: Number (limit range)

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the Sync information of the sRX1

Index	Description
0	Unknown
1	Master
2	Follower
3	Unsync'd Follower

Examples:

```
TX: {"rx1":{"sync_info":null}}
RX: {"rx1":{" sync_info ":1}}
```

8.46 /rx1/rfpi

Parameter type: String

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Returns the RFPI of the SLDW.

Examples:

```
TX: {"rx1":{"rfpi":null}}
RX: {"rx1":{"rfpi":"028112cf28"}}
```

8.47 /rx1/last_paired_ipei

Parameter type: String

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Returns the IPEI of the (last) connected portable device.

Examples:

```
TX: {"rx1":{"last_paired_ipei":null}}
RX: {"rx1":{"last_paired_ipei":"028110a938"}}
```

8.48 /rx1/autolock

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has immediately effect.

Description: Method to modify/get the auto-lock state of the sRX1. This parameter is related to the sRX1 UI menu item "Auto Lock" under "System Settings".

Example:

```
TX: {"rx1":{"autolock":null}}
RX: {"rx1":{"autolock":true}}
```

8.49 /rx1/warnings

Parameter type: String

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to get sRX1 warnings. Supported warnings are "" if there is no warning, "HW Failure" in case during device start-up a HW failure is detected, "Bad Link" if the connected between the transmitter and sRX1 is bad, "No Link" if there is no transmitter connected with the sRX1 or "No FW Image" if the sRX1 cannot find a firmware image to update the connected transmitter via FWU_OTA. In case the RF Stack is disabled (f.i. by using /rx1/rf_stack_disable) the "Link Disabled" will be given.

Example:

```
TX: {"rx1":{"warnings":null}}
RX: {"rx1":{"warnings":["Bad Link"]}}
```

8.50 /mates/active

Parameter type: Boolean

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to get active mates. Returns ["tx1"] if the sRX1 has an established link with a transmitter, else it returns [].

Example:

```
TX: {"mates":{"active":null}}
RX: {"mates":{"active":["tx1"]}}
```

8.51 /mates/tx1/device_type

Parameter type: Number (limit range)

Permission: Read, Subscribe-able

Pre-condition: /mates/tx1 must be active.

Post-condition: NA.

Description: Method to retrieve the transmitter type.

Index	Description
0	Handheld
1	Bodypack
2	Tablestand
3	Boundary

Example:

```
TX: {"mates":{"tx1":{"device_type":null}}}
RX: {"mates":{"tx1":{"device_type":2}}}
```

8.52 /mates/tx1/bat_type

Parameter type: Number (limit range)

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to retrieve type of the battery.

Index	Description
0	Battery
1	Rechargeable

Examples:

```
TX: {"mates":{"tx1":{"bat_type":null}}}
RX: {"mates":{"tx1":{"bat_ type":0}}}
```

8.53 /mates/tx1/bat_state

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to retrieve status of the battery. If the connected transmitter uses a rechargeable battery and the lifetime is available the SSC Server will return with /mates/tx1/bat_lifetime, in all other cases it will return with /mates/tx1/bat_gauge. Note that it takes some time before it is possible to retrieve the predicted lifetime of a rechargeable battery, before the lifetime is available the actual capacity of the rechargeable battery is used.

Examples:

```
TX: {"mates":{"tx1":{"bat_state":null}}}
RX: {"mates":{"tx1":{"bat_gauge":16}}}

TX: {"mates":{"tx1":{"bat_state":null}}}
RX: {"mates":{"tx1":{"bat_lifetime":13560}}}
```

8.54 /mates/tx1/bat_charging

Parameter type: Boolean

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between a Table-stand or Boundary, powered by a rechargeable battery, and sRX1.

Post-condition: N.A.

Description: Method to retrieve the charge status.

Examples:

```
TX: {"mates":{"tx1":{"bat_charging":null}}}
RX: {"mates":{"tx1":{"bat_charging":false}}}
```

8.55 /mates/tx1/bat_gauge

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to retrieve the capacity of the rechargeable battery, or in case a battery is used the battery voltage level in percentage.

Examples:

```
TX: {"mates":{"tx1":{"bat_gauge":null}}}
RX: {"mates":{"tx1":{"bat_gauge":100}}}
```

8.56 /mates/tx1/bat_lifetime

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter, powered by a rechargeable battery, and sRX1.

Post-condition: N.A.

Description: Method to retrieve the lifetime of the rechargeable battery in seconds.

Examples:

```
TX: {"mates":{"tx1":{"bat_lifetime":null}}}
RX: {"mates":{"tx1":{"bat_lifetime":12900}}}
```

8.57 /mates/tx1/bat_bars

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter, powered by a rechargeable battery, and sRX1.

Post-condition: N.A.

Description: Method to retrieve the number of bars shown in the sRX1s' homescreen.

Examples:

```
TX: {"mates":{"tx1":{"bat_bars":null}}}
RX: {"mates":{"tx1":{"bat_bars":4}}}
```

8.58 /mates/tx1/bat_health

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter, powered by a rechargeable battery, and sRX1.

Post-condition: N.A.

Description: Method to retrieve the battery health status of the connected transmitter.

Examples:

```
TX: {"mates":{"tx1":{"bat_health":null}}}
RX: {"mates":{"tx1":{"bat_bars":100}}}
```

8.59 /mates/tx1/bat_cycles

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter, powered by a rechargeable battery, and sRX1.

Post-condition: N.A.

Description: Method to retrieve the battery cycle count (charging cycles).

Examples:

```
TX: {"mates":{"tx1":{"bat_cycles":null}}}
RX: {"mates":{"tx1":{"bat_bars":6}}}
```

8.60 /mates/tx1/switch1/label

Parameter type: String

Permission: Read only

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to get the switch1 label on the transmitter.

Example:

```
TX: {"mates":{"tx1":{"switch1":{"label":null}}}}
RX: {"mates":{"tx1":{"switch1":{"label":"Mute"}}}}
```

8.61 /mates/tx1/switch1/state

Parameter type: Boolean

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to get the switch1 state (Mute switch) on the transmitter. Note that the sRX1 can have disabled switch1 with the /rx1/mute_switch_active method.

Example:

```
TX: {"mates":{"tx1":{"switch1":{"state":null}}}}
RX: {"mates":{"tx1":{"switch1":{"state":true}}}}
```

8.62 /mates/tx1/acoustic

Parameter type: String

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to get the acoustic input type of mate "tx1". Support transmitters are: "Mic", "Line", "MME865", "MD42", "MMD945", "MMD935", "MMD845", "MMD835", "MMD815_1", "MMK965s", "MMK965c", "BOUNDARY" and "GOOSENECK". When a not supported capsule is connected "INCOMPATIBLE" will be returned, and if there is no capsule connected "" will be returned.

Examples:

Not connected with a transmitter:

```
TX: {"mates":{"tx1":{"acoustic":null}}}
RX: {"mates":{"tx1":{"acoustic":""}}}
```

Connected with a MD42:

```
TX: {"mates":{"tx1":{"acoustic":null}}}
RX: {"mates":{"tx1":{"acoustic":"MD42"}}}
```

8.63 /mates/tx1/warnings

Parameter type: String

Permission: Read only, Subscribe-able

Pre-condition: Link must be established between transmitter and sRX1.

Post-condition: N.A.

Description: Method to get transmitter warnings connected to the sRX1. Supported warnings are "" if there is no warning or "Low Bat" if the capacity of a rechargeable battery is below a certain level or if the voltage of a battery is below a certain level, the levels are depending on the battery manufacturer and type of transmitter.

Example:

```
TX: {"mates":{"tx1":{"warnings":null}}}
RX: {"mates":{"tx1":{"warnings":["Low Bat"]}}}
```

8.64 /mates/tx1/gooseneck_state

Parameter type: Boolean

Permission: Read, Subscribe-able

Pre-condition: The transmitter type must be "Tablestand"

Post-condition: NA.

Description: Method to retrieve information if the Gooseneck is attached to or detached from the Tablestand.

Example:

```
TX: {"mates":{"tx1":{"gooseneck_state":null}}}
RX: {"mates":{"tx1":{"gooseneck_state":true}}}
```

8.65 /audio/out1/label

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the output label.

Example:

```
TX: {"audio":{"out1":{"label":null}}}
RX: {"audio":{"out1":{"label": ["AF OUT BAL +18 dBu max","AF OUT UNBAL"]}}}
```

8.66 /audio/out1/level_db

Parameter type: Number (limit range)

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the actual output level on the sRX1. The parameter value is inside a defined range [-60..0].

Example:

```
TX: {"audio":{"out1":{"level_db":null}}}
RX: {"audio":{"out1":{"level_db":-56}}}
```

8.67 /audio/out1/gain_db

Parameter type: Number (limit range)

Permission: Read/Write, Subscribe-able

Pre-condition: The parameter value must be inside the index range [0-6].

Index	Description
0	-24 dB
1	-18 dB
2	-12 dB
3	-6 dB
4	0 dB
5	6 dB
6	12 dB

Post-condition: The change has immediately effect.

Description: Method to retrieve/modify the Output level. This parameter is related to the sRX1 UI menu item "Audio Level" under "Audio Settings".

Example:

```
TX: {"osc":{"limits":[{"audio":{"out1":{"gain_db":null}}}]}}
RX: {"osc":{"limits":[{"audio":{"out1":{"gain_db":[{"type":"Number",
    "desc":"out1 gain","option":[0,1,2,3,4,5,6],"option_desc":["-24 dB",
    "-18 dB","-12 dB","-6 dB","0 dB","6 dB","12 dB"]}]}}}]}}
TX: {"audio":{"out1":{"gain_db":5}}}
RX: {"audio":{"out1":{"gain_db":5}}}
```

8.68 /audio/equalizer/preset

Parameter type: Number (limit range)

Permission: Read/Write, Subscribe-able

Pre-condition: The parameter value must be inside the index range [0-4].

Index	Description
0	Off
1	Female Speech
2	Male Speech
3	Media
4	Custom

Post-condition: The change has immediately effect.

Description: Method to retrieve/modify the equalizer preset. This parameter is related to the sRX1 UI menu item "Sound Profile" under "Audio Settings".

Example:

```
TX: {"audio":{"equalizer":{"preset":null}}}
RX: {"audio":{"equalizer":{"preset":0}}}
```

8.69 /audio/low_cut

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has immediately effect.

Description: Method to enable/disable low cut settings, equivalent to the "Low Cut" menu item under "Audio Settings" in the sRX1 UI.

Example:

```
TX: {"audio":{"low_cut":null}}
RX: {"audio":{"low_cut":false}}
```

8.70 /audio/effects_reset

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: The change has immediately effect.

Description: Method to restore the audio effects default settings, equivalent to the "Effects Reset" menu item under "Audio Settings" in the sRX1 UI.

Example:

```
TX: {"audio":{"effects_reset":true}}
RX: {"audio":{"effects_reset":true}}
```

8.71 /brightness

Parameter type: Number (limit range)

Permission: Read/Write, Subscribe-able

Pre-condition: The parameter value must an integer value in the range [0..100]

Post-condition: The change has immediately effect.

Description: Method to retrieve/modify the OLED display brightness in percentage of the sRX1. This parameter is related to the sRX1 UI menu item "Brightness" under "System Settings".

Example:

```
TX: {"brightness":100}
RX: {"brightness":100}
```

9. SSC Error List (SL Rack Receiver DW)

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.

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

The error code is a three-digit integer, defined in the style of Hypertext Transfer Protocol (HTTP) response status codes, the status code is part of the HTTP/1.1 standard (RFC 7231).

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 - Successful - The action was successfully received, understood, and accepted.

• 3xx - Redirection - 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 SSC 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.

9.1 1xx Informational

The 1xx (Informational) class of status code indicates an interim response for communicating connection status or request progress prior to completing the requested action and sending a final response.

9.1.1 100 Continue

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the SSC Server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response.

The SSC Server MUST send a final response after the request has been completed.

9.1.2 102 Processing

The 102 (Processing) status code is an interim response used to inform the client that the SSC Server has accepted the complete request, but has not yet completed it. This status code SHOULD only be sent when the SSC Server has a reasonable expectation that the request will take significant time to complete. As guidance, if a method is taking longer than 20 seconds (a reasonable, but arbitrary value) to process the SSC Server SHOULD return a 102 (Processing) response.

The SSC Server MUST send a final response after the request has been completed.

9.2 2xx Success

This class of status code indicates that the client's request was successfully received, understood, and accepted.

9.2.1 200 OK

Standard response for successful requests.

9.2.2 201 Created

The request has been fulfilled and resulted in a new resource being created. The origin SSC Server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the SSC Server SHOULD respond with 202 (Accepted) response instead.

9.2.3 202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

9.2.4 210 Partial Success

The request has been partially accepted for processing.

9.3 3xx Redirection

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user.

A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

9.3.1 310 Subscription Terminates

The subscription is terminated on the SSC Server, because the lifetime expired or the max number of occurrences exceeded.

9.4 4xx Client Error

The 4xx class of status code is intended for cases in which the client seems to have erred.

These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

9.4.1 400 Bad Request

The request could not be understood by the SSC Server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

9.4.2 401 Unauthorized

Error code response for missing or invalid authentication token. The request requires user authentication. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.

9.4.3 403 Forbidden

Error code for user not authorized to perform the operation or the resource is unavailable for some reason (e.g. time constraints, etc.). The SSC Server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated.

9.4.4 404 Not Found

The SSC Server has not found anything matching the request. No indication is given of whether the condition is temporary or permanent. The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible. Used when the requested resource is not found, whether it doesn't exist or if there was a 401 or 403 that, for security reasons, the service wants to mask.

9.4.5 406 Not Acceptable (E.g. wrong type for parameter)

The SSC Server is not capable of processing the request, f.i. the client request to change a read only parameter value.

9.4.6 408 Request Timeout

The client did not produce a request within the time that the SSC Server was prepared to wait. The client MAY repeat the request without modifications at any later time.

9.4.7 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations

where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.

9.4.8 410 Gone

Indicates that the resource requested is no longer available and will not be available again. This should be used when a resource has been intentionally removed and the resource should be purged. Upon receiving a 410 status code, the client should not request the resource again in the future.

9.4.9 413 Request Entity Too Large

The SSC Server is refusing to process a request because the request entity is larger than the SSC Server is willing or able to process.

9.4.10 414 Request Too Complex

The URI provided was too long for the SSC Server to process.

9.4.11 422 Unprocessable Entity

This status code means the SSC Server understands the content type of the request entity, and the syntax of the request entity is correct but was unable to process the contained instructions. For example, this error condition may occur if request is syntactically correct, but it is semantically erroneous.

9.4.12 423 Locked

The resource that is being accessed is locked.

9.4.13 424 Failed Dependency

The request failed due to failure of a previous request.

9.4.14 450 Answer Too Long

This status code is used by the SSC Server to inform the client that the message length for the response is to large and cannot be send.

9.4.15 454 Parameter Address Not Found

This status code is used by the SSC Server to inform the client that the request cannot be processed, because the requested addresses is hidden.

9.5 5xx Server Error

Response status codes beginning with the digit "5" indicate cases in which the SSC Server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the SSC Server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

9.5.1 500 Internal Server Error

A generic error message, given when no more specific message is suitable.

9.5.2 501 Not Implemented

The SSC Server does not support the functionality required to fulfill the request. This is the appropriate response when the SSC Server does not recognize the request method and is not capable of supporting it for any resource.

9.5.3 503 Service Unavailable

The SSC Server is currently unable to handle the request due to a temporary overloading or maintenance of the SSC Server. The implication is that this is a temporary condition which will be alleviated after some delay.

10. Developer’s Guide for CHG 4N

This chapter describes in detail how a developer should use the SSC interface as implemented for the CHG 4N network-based charger.

10.1 Limitations

10.1.1 SSC Transport Layer

The SSC Server implemented for CHG 4N devices supports only UDP/IP as transport protocol. All the devices support both IPv4 and IPv6.

10.1.2 Subscriptions

CHG4N devices support SSC Method subscriptions from up to eight different SSC clients simultaneously.

11. SSC Method List (CHG 4N)

11.1 /interface/version

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: SSC interface version.

Example:

```
TX: {"interface":{"version":null}}
RX: {"interface":{"version":"1.0"}}
```

11.2 /osc/xid

Parameter type: N.A.

Permission: N.A.

Pre-condition: N.A.

Post-condition: N.A.

Description: When an SSC Client 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.

Examples:

```
TX: {"osc":{"xid":1234567890,"ping":["AbCdEfGhIjKlMnOpQrStUvWxYz",3,
    14159265358979323846264338327950]}}
RX: {"osc":{"ping":["AbCdEfGhIjKlMnOpQrStUvWxYz",3,
    14159265358979323846264338327950],"xid":1234567890}}
TX: {"osc":{"xid":1234567890},"bays":{"active":null}}
RX: {"osc":{"xid":1234567890},"bays":{"active":[true,false,true,true]}
```

11.3 /osc/version

Parameter type: String.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: Reports the SSC version implemented in the server.

Example:

```
TX: {"osc":{"version":null}}
RX: {"osc":{"version":"1.0"}}
```

11.4 /osc/error

Parameter type: Number (limit range)

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

The error message MUST contain an integer numeric value, the error code. The error code SHOULD be chosen from the SSC Error List detailed in chapter 2.

Examples:

```
TX: {"out1":{"gain":10}}
RX: {"osc":{"error":[{{"out1":[404]}}]}}     <-- not found
TX: {"write_protection":true}
RX: {"osc":{"error":[{"write_protection":[406]}]}}    <-- read only
```

11.5 /osc/schema

Parameter type: N.A.

Permission: N.A..

Pre-condition: N.A.

Post-condition: N.A.

Description: The /osc/schema method exists to allow clients to query servers about what address schemes are available on a specific server. SSC clients MUST be able to understand both bundled and unbundled replies. 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.

Examples:

```
TX: {"osc":{"schema":null}}
RX: {"osc":{"schema":[{"bays":{},"device":{},"interface":{},"osc":{}}]}}
TX: {"osc":{"schema":[{"device":null}]}}
RX: {"osc":{"schema":[{"device":{"identity":{},"network":{},
    "timeprecision":null,"time":null,"warnings":null,"state":null,
    "reset":null,"ptxversion_sl":null,"ptxversion_d1":null,"progress":
    null,"name":null,"language":null,"group":null,"factory_reset":null}}]}}
```

11.6 /osc/limits

Parameter type: N.A.

Permission: N.A..

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

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

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
- units string String describing value units (preferably SI)
- desc string descriptive text, meant for display to the user
- option 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.

Examples:

```
TX: {"osc":{"limits":[{"bays":{"bat_bars":null}}]}}
RX: {"osc":{"limits":[{"bays":{"bat_bars":[{"type":"Number","max":6,
    "min":0,"inc":1,"units":"bars"}]}}]}}
TX: {"osc":{"limits":[{"bays":{"state":}}]}}
RX: {"osc":{"limits":[{"bays":{"state":[{"type":"Number","desc":
    "Bays status","option":[0,1,2],"option_desc":["NORMAL","FWU",
    "ERROR"]}]}}]}}
```

11.7 /osc/feature/pattern

Parameter type: String.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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

Example:

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

11.8 /osc/feature/baseaddr

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: Using a baseaddr helps to explore a device in a truly interactive manner, and may additionally be used to reduce message lengths by shortening addresses in SSC requests.

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

Example:

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

11.9 /osc/feature/subscription

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

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

Example:

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

11.10 /osc/feature/timetag

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

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

Example:

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

11.11 /osc/state/subscribe

Parameter type: N.A.

Permission: N.A.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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. Without a lifetime value (unit is seconds), the default lifetime for a subscription to be active is 10 seconds.

Examples 1 – with default parameter:

```
TX: {"osc":{"xid":1234567890,"state":{"subscribe":[{"bays":{"active":
    null}}]}}}
RX: {"osc":{"xid":1234567890},"osc":{"state":{"subscribe":[{"bays":
    {"active":[false,false,false,false]}}]}}}
RX: {"bays":{"active":[false,false,false,false]}
```

subscription terminates after 10 seconds with:

```
RX: {"osc":{"error":[{"bays":{"active":[310]}}]}}
```

Example 2 – with non-default parameter:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"bays":
    {"active":null}}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"bays":
    {"active":[false,false,false,false]}}]}}}
RX: {"bays":{"active":[false,false,false,false]}
```

Parameter value changes:

```
RX: {"bays":{"active":[true,false,false,false]}}
RX: {"bays":{"active":[true,true,false,false]}}
RX: {"bays":{"active":[true,true,true,false]}}
RX: {"bays":{"active":[false,true,true,false]}}
```

subscription terminates at end of subscription lifetime with:

```
RX: {"osc":{"error":[{"bays":{"active":[310]}}]}}
```

Example 3 – multi-parameter with non-default parameter:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"device":
    {"state":null},"bays":{"active":null,"device_type":null}}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"lifetime":2000},"device":
    {"state":null},"bays":{"active":null,"device_type":null}}]}}}
RX: {"device":{"state":0},"bays":{"active":[false,false,false,false],
    "device_type":[0,0,0,0]}}
```

subscription terminates at end of subscription lifetime with:

```
RX: {"osc":{"error":[{"device":{"state":[310]},"bays":{"active":[310],
    "device_type:[310]}}]}}
```

11.12 /osc/state/close

Parameter type: Boolean.

Permission: Write only.

Pre-condition: N.A.

Post-condition: N.A.

Description: When an SSC Client calls this SSC Method with a true argument, the SSC Server MUST close the connection immediately after the reply has been sent.

Example:

```
TX: {"osc":{"state":{"close":true}}}
RX: {"osc":{"state":{"close":true}}}
```

11.13 /osc/state/prettyprint

Parameter type: Boolean.

Permission: Read only.

Pre-condition: N.A.

Post-condition: N.A.

Description: 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

Examples:

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

Example

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

11.14 /device/name

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: 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; if it has a menu, this name should be configurable.

If the device is networked, this name shall be used as the name for device 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.

Example:

```
TX: {"device":{"name":null}}
RX: {"device":{"name":"CHG4N"}}
```

11.15 /device/group

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: A reduced charset is supported: ‘0’-‘9’, ‘-‘, ‘_’, ‘A’-‘Z’ and ‘a’-‘z’, and following rules should be followed:

1. A group name SHOULD start only with a letter.
2. A group name MAY end with a letter or a number.
3. A group name SHOULD NOT start or end with a ‘-’ (dash) or ‘_’ (underscore).
4. A group name MAY be up to 8 characters.

Post-condition: The change has immediately effect.

Description: Method to retrieve/modify the group name. This parameter is related to the sRX1 UI menu item "Location".

Example:

```
TX: {"device":{"group":"room"}}
RX: {"device":{"group":"room"}}
```

11.16 /device/language

Parameter type: String

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: 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. 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.

Example:

```
TX: {"device":{"language":null}}
RX: {"device":{"language":["en_GB"]}}
```

11.17 /device/identity/product

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Product identification, should be identical to the designation on the label on the product itself or to the included electronic/component.

Example:

```
TX: {"device":{"identity":{"product":null}}}
RX: {"device":{"identity":{"product":"CHG4N"}}}
```

11.18 /device/identity/version

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Device firmware version.

Example:

```
TX: {"device":{"identity":{"version":null}}}
RX: {"device":{"identity":{"version":"1.0.0"}}}
```

11.19 /device/identity/serial

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Unique product number, identical to the number on a product label. The value is given by production.

Example:

```
TX: {"device":{"identity":{"serial":null}}}
RX: {"device":{"identity":{"serial":"1454100930"}}}
```

11.20 /device/identity/vendor

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Device vendor name.

Example:

```
TX: {"device":{"identity":{"vendor":null}}}
RX: {"device":{"identity":{"vendor":"Sennheiser electronic GmbH & Co. KG"
    }}}
```

11.21 /device/network/ether/interfaces

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of all the user-relevant Ethernet interfaces of the device.

Example:

```
TX: {"device":{"network":{"ether":{"interfaces":null}}}}
RX: {"device":{"network":{"ether":{"interfaces":["LAN"]}}}}
```

11.22 /device/network/ether/macs

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: 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.

Example:

```
TX: {"device":{"network":{"ether":{"macs":null}}}}
RX: {"device":{"network":{"ether":{"macs":["00:1B:66:7D:F8:99"]}}}}
```

11.23 /device/network/ipv4/interfaces

Parameter type: Number

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array relating to /device/network/ether/interfaces. The array contains numbers indexing into the array of physical interface names. The interface index array contains the indices of all physical interfaces which share the IPv4 configuration.

Example:

```
TX: {"device":{"network":{"ipv4":{"interfaces":null}}}}
RX: {"device":{"network":{"ipv4":{"interfaces":[1]}}}}
```

11.24 /device/network/ipv4/auto

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: Array containing a list of boolean values that indicates whether IPv4 shall be configured automatically by means of DHCP and ZeroConf (Auto-IP) of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. If the value is set to false the values of /device/network/ipv4/fixed_ipaddr, /device/network/ipv4/fixed_ipaddr and /device/network/ipv4/fixed_ipaddr will be used during target initialization at start-up. A value change takes effect after device reset!

Example:

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

11.25 device/network/ipv4/ipaddr

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the current IPv4 addresses of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"ipaddr":null}}}}
RX: {"device":{"network":{"ipv4":{"ipaddr":["192.168.1.203"]}}}}
```

11.26 /device/network/ipv4/netmask

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the current IPv4 netmasks of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 netmasks are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"netmask":null}}}}
RX: {"device":{"network":{"ipv4":{"netmask":["255.255.255.0"]}}}}
```

11.27 /device/network/ipv4/gateway

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing a list of the IPv4 gateways of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 gateways are specified as strings in standard dot-decimal notation.

Example:

```
TX: {"device":{"network":{"ipv4":{"gateway":null}}}}
RX: {"device":{"network":{"ipv4":{"gateway":["192.168.1.1"]}}}}
```

11.28 /device/network/ipv4/fixed_ipaddr

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 addresses in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4 addresses in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_ipaddr":["192.168.1.203"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_ipaddr":["192.168.1.203"]}}}}
```

11.29 /device/network/ipv4/fixed_netmask

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 netmasks in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4 netmasks in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_netmask":["255.255.255.0"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_netmask":["255.255.255.0"]}}}}
```

11.30 /device/network/ipv4/fixed_gateway

Parameter type: String

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: An array containing a list of the stored IPv4 gateways in EEPROM of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv4 addresses are specified as strings in standard dot-decimal notation. The stored IPv4 gateways in EEPROM are used by the device if /device/network/ipv4/auto is set to false during start-up of the device.

Example:

```
TX: {"device":{"network":{"ipv4":{"fixed_gateway":["192.168.1.1"]}}}}
RX: {"device":{"network":{"ipv4":{"fixed_gateway":["192.168.1.1"]}}}}
```

11.31 /device/network/ipv6/interfaces

Parameter type: Number

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array relating to /device/network/ether/interfaces. The array contains numbers indexing into the array of physical interface names. The interface index array contains the indices of all physical interfaces which share the IPv6 configuration.

Example:

```
TX: {"device":{"network":{"ipv6":{"interfaces":null}}}}
RX: {"device":{"network":{"ipv6":{"interfaces":[1]}}}}
```

11.32 /device/network/ipv6/ipaddr

Parameter type: String

Permission: Read only

Pre-condition: N.A.

Post-condition: N.A.

Description: Read-only array containing an array of all the IPv6 addresses of all the user-relevant Ethernet interface of the device. The order of the list matches /device/network/ether/interfaces. The IPv6 addresses are specified as strings in standard notation. Each IPv6 network interface can contain maximal 3 IPv6 addresses.

Example:

```
TX: {"device":{"network":{"ipv6":{"ipaddr":null}}}}
RX: {"device":{"network":{"ipv6":{"ipaddr":[["fe80::21b:66ff:fe7d:f899",
    "2001:db8::b2f4:4bff:fa71:1a56"]]}}}}
```

11.33 /device/network/mdns_responder

Parameter type: Boolean

Permission: Read/Write, Subscribe-able

Pre-condition: N.A.

Post-condition: The change has effect after restart of the device.

Description: Method to retrieve/modify the setting to enable/disable mDNS. This parameter is related to the sRX1 UI menu item "mDNS" under "Network Settings".

Example:

```
TX: {"device":{"network":{"mdns_responder":null}}}
RX: {"device":{"network":{"mdns_responder":true}}}
```

11.34 /device/state

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Read value is a ‘0’ (Normal mode) or a ‘1’ (FWU mode). At the end of a firmware update process the charger does a restart to run the new firmware. This means that all subscriptions are obsolete.

Example:

```
TX: {"device":{"state":null}}
RX: {"device":{"state":0}}
```

11.35 /device/progress

Parameter type: Number

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Read value is a percentage value between ‘0’ and ‘100’ to indicate the progress of a charger firmware update process. At the end of a firmware update process the charger does a restart to run the new firmware. This means that all subscriptions are obsolete.

Example:

```
TX: {"device":{"progress":null}}
RX: {"device":{"progress":17}}
```

11.36 /device/ptxversion_sl

Parameter type: String

Permission: Read, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Get the version of the currently stored speechline ptx firmware image. This version can then be used with "/bays/update" method.

Example:

```
TX: {"device":{"ptxversion_sl":null}}
RX: {"device":{"ptxversion_sl":"0.7.4"}}
```

11.37 /device/ptxversion_d1

Parameter type: String

Permission: Read, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Get the version of the currently stored ewd1 ptx firmware image. This version can then be used with "/bays/update" method.

Example:

```
TX: {"device":{"ptxversion_d1":null}}
RX: {"device":{"ptxversion_d1":"0.7.4"}}
```

11.38 /device/warnings

Parameter type: Boolean

Permission: Read

Pre-condition: N.A.

Post-condition:

Description:

Example:

```
TX: {"device":{"warnings":null}}
RX: {"device":{"warnings":[""]}}
```

11.39 /device/reset

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: N.A.

Description: A soft reset may be necessary for some configuration settings to take effect.

Example:

```
TX: {"device":{"reset":true}}
RX: {"device":{"reset":true}}
```

11.40 device/factory_reset

Parameter type: Boolean

Permission: Read/Write

Pre-condition: N.A.

Post-condition: The device will restart itself so that after restart the new settings are used.

Description: Boolean value to re-configure the device with factory defaults.

Example:

```
TX: {"device":{"factory_reset":true}}
RX: {"device":{"factory_reset":true}}
```

11.41 /bays/active

Parameter type: Array of 4 booleans

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to get active bays. Returns an array with the bays where a portable device is inserted.

Example:

```
TX: {"bays":{"active":null}}
RX: {"bays":{"active":[true,true,false,false]}}
```

11.42 /bays/device_type

Parameter type: Array of 4 numbers

Permission: Read only

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Returns the device type of the PP (1 – pTXh; 2 – pTXb).

Example:

```
TX: {"bays":{"device_type":null}}
RX: {"bays":{"device_type":[0,0,2,1]}}
```

11.43 /bays/market_type

Parameter type: Array of 4 numbers (internal feature)

Permission: Read only

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Returns the market ID of the PP (1 – MI; 2 – VJ; 4 – IS).

Example:

```
TX: {"bays":{"market_type":null}}
RX: {"bays":{"market_type":[0,4,4,4]}}
```

11.44 /bays/serial

Parameter type: Array of 4 strings

Permission: Read only, Subscribe-able

Pre-Condition: Portable device must be inserted in one of the CHG4N bays.

Post-Condition: N.A.

Description: Charger returns the serial number of the inserted PPs.

Example:

```
TX: {"bays":{"serial":null}}
RX: {"bays":{"serial":["","1106103018","1106103029","1106103023"]}}
```

11.45 bays/version

Parameter type: Array of 4 strings

Permission: Read only, Subscribe-able

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns the firmware linkdate of the inserted PPs. (WEBO version look-a-like)

Example:

```
TX: {"bays":{"version":null}}
RX: {"bays":{"version":["","99.99.99","0.7.4","0.7.3"]}}
```

11.46 /bays/linkdate

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns the firmware linkdate of the inserted PPs.

Example:

```
TX: {"bays":{"linkdate":null}}
RX: {"bays":{"linkdate":[0000000000,1606021157,1605201510,1605101917]}}
```

11.47 /bays/charging

Parameter type: Array of 4 booleans

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the charge status (true – corresponding device will be charged; false – corresponding slot is empty or device in corresponding slot is fully charged).

Examples:

```
TX: {"bays":{"charging":null}}
RX: {"bays":{"charging":[false,false,true,true]}}
```

11.48 /bays/bat_gauge

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns the remaining capacity of the inserted pTx in percent.

Example:

```
TX: {"bays":{"bat_gauge":null}}
RX: {"bays":{"bat_gauge":[0,100,0,100]}}
```

11.49 /bays/bat_timetofull

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns the remaining time in minutes until the inserted transmitters are fully charged.

Example:

```
TX: {"bays":{"bat_timetofull":null}}
RX: {"bays":{"bat_timetofull":[0,10,100,85]}}
```

11.50 /bays/bat_bars

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns the number of bars in the display of the PP.

Example:

```
TX: {"bays":{"bat_bars":null}}
RX: {"bays":{"bat_bars":[0,6,5,6]}}
```

11.51 /bays/bat_health

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the health status of the batteries of the inserted transmitters.

Example:

```
TX: {"bays":{"bat_health":null}}
RX: {"bays":{"bat_health":[0,95,100,100]}}
```

11.52 /bays/bat_cycles

Parameter type: Array of 4 numbers

Permission: Read only, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to retrieve the number of charging cycles of the inserted transmitters.

Example:

```
TX: {"bays":{"bat_cycles":null}}
RX: {"bays":{"bat_cycles":[0,20,28,16]}}
```

11.53 /bays/identify

Parameter type: Array of 4 booleans

Permission: Write, Subscribe-able

Pre-condition: N.A.

Post-condition: N.A.

Description: Method to identify the slots of a CHG4N. Send a boolean array in the order SLT1 – SLT4 to make all LEDs of that slot flash (true) or to turn them off (false). Send null to not change the identify state. Identify state is held for 10s.

Example:

```
TX: {"bays":{"identify":[true,false,false,false]}}
RX: {"bays":{"identify":[true,false,false,false]}}
TX: {"bays":{"identify":[true,null,null,null]}}
RX: {"bays":{"identify":[true,false,false,false]}}
```

11.54 /bays/state

Parameter type: 2 arrays of 4 numbers each

Permission: Read only

Pre-Condition: N.A.

Post-Condition: N.A.

Description: Charger returns two arrays of 4 bytes. The first array indicates the state of the bay (0 – Normal; 1 – Update; 2 – Error) while the second array indicates the kind of error as a bit map.

Error indication (Bit set to '1' means error):

- Bit 0 – communication error
- Bit 1 – update error
- Bit 2 – overcurrent error
- Bit 3 – 5V charging voltage missing

Example:

```
TX: {"bays":{"state":null}}
RX: {"bays":{"state":[[0,0,0,0],[0,0,0,0]]}}
```

12. SSC Error List (CHG 4N)

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.

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

The error code is a three-digit integer, defined in the style of Hypertext Transfer Protocol (HTTP) response status codes, the status code is part of the HTTP/1.1 standard (RFC 7231).

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 - Successful - The action was successfully received, understood, and accepted.
- 3xx - Redirection - 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 SSC 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.

12.1 1xx Informational

The 1xx (Informational) class of status code indicates an interim response for communicating connection status or request progress prior to completing the requested action and sending a final response.

12.1.1 100 Continue

The client SHOULD continue with its request. This interim response is used to inform the client that the initial part of the request has been received and has not yet been rejected by the SSC Server. The client SHOULD continue by sending the remainder of the request or, if the request has already been completed, ignore this response.

The SSC Server MUST send a final response after the request has been completed.

12.1.2 102 Processing

The 102 (Processing) status code is an interim response used to inform the client that the SSC Server has accepted the complete request, but has not yet completed it. This status code SHOULD only be sent when the SSC Server has a reasonable expectation that the request will take significant time to complete. As guidance, if a method is taking longer than 20 seconds (a reasonable, but arbitrary value) to process the SSC Server SHOULD return a 102 (Processing) response.

The SSC Server MUST send a final response after the request has been completed.

12.2 2xx Success

This class of status code indicates that the client's request was successfully received, understood, and accepted.

12.2.1 200 OK

Standard response for successful requests.

12.2.2 201 Created

The request has been fulfilled and resulted in a new resource being created. The origin SSC Server MUST create the resource before returning the 201 status code. If the action cannot be carried out immediately, the SSC Server SHOULD respond with 202 (Accepted) response instead.

12.2.3 202 Accepted

The request has been accepted for processing, but the processing has not been completed. The request might or might not eventually be acted upon, as it might be disallowed when processing actually takes place. There is no facility for re-sending a status code from an asynchronous operation such as this.

12.2.4 210 Partial Success

The request has been partially accepted for processing.

12.3 3xx Redirection

This class of status code indicates that further action needs to be taken by the user agent in order to fulfill the request. The action required MAY be carried out by the user agent without interaction with the user.

A client SHOULD detect infinite redirection loops, since such loops generate network traffic for each redirection.

12.3.1 310 Subscription Terminates

The subscription is terminated on the SSC Server, because the lifetime expired or the max number of occurrences exceeded.

12.4 4xx Client Error

The 4xx class of status code is intended for cases in which the client seems to have erred.

These status codes are applicable to any request method. User agents SHOULD display any included entity to the user.

12.4.1 400 Bad Request

The request could not be understood by the SSC Server due to malformed syntax. The client SHOULD NOT repeat the request without modifications.

12.4.2 401 Unauthorized

Error code response for missing or invalid authentication token. The request requires user authentication. If the request already included Authorization credentials, then the 401 response indicates that authorization has been refused for those credentials.

12.4.3 403 Forbidden

Error code for user not authorized to perform the operation or the resource is unavailable for some reason (e.g. time constraints, etc.). The SSC Server understood the request, but is refusing to fulfill it. Authorization will not help and the request SHOULD NOT be repeated.

12.4.4 404 Not Found

The SSC Server has not found anything matching the request. No indication is given of whether the condition is temporary or permanent. The requested resource could not be found but may be available again in the future. Subsequent requests by the client are permissible. Used when the requested resource is not found, whether it doesn't exist or if there was a 401 or 403 that, for security reasons, the service wants to mask.

12.4.5 406 Not Acceptable (E.g. wrong type for parameter)

The SSC Server is not capable of processing the request, f.i. the client request to change a read only parameter value.

12.4.6 408 Request Timeout

The client did not produce a request within the time that the SSC Server was prepared to wait. The client MAY repeat the request without modifications at any later time.

12.4.7 409 Conflict

The request could not be completed due to a conflict with the current state of the resource. This code is only allowed in situations

where it is expected that the user might be able to resolve the conflict and resubmit the request. The response body SHOULD include enough information for the user to recognize the source of the conflict. Ideally, the response entity would include enough information for the user or user agent to fix the problem; however, that might not be possible and is not required.

12.4.8 410 Gone

Indicates that the resource requested is no longer available and will not be available again. This should be used when a resource has been intentionally removed and the resource should be purged. Upon receiving a 410 status code, the client should not request the resource again in the future.

12.4.9 413 Request Entity Too Large

The SSC Server is refusing to process a request because the request entity is larger than the SSC Server is willing or able to process.

12.4.10 414 Request Too Complex

The URI provided was too long for the SSC Server to process.

12.4.11 422 Unprocessable Entity

This status code means the SSC Server understands the content type of the request entity, and the syntax of the request entity is correct but was unable to process the contained instructions. For example, this error condition may occur if request is syntactically correct, but it is semantically erroneous.

12.4.12 423 Locked

The resource that is being accessed is locked.

12.4.13 424 Failed Dependency

The request failed due to failure of a previous request.

12.4.14 450 Answer Too Long

This status code is used by the SSC Server to inform the client that the message length for the response is to large and cannot be send.

12.4.15 454 Parameter Address Not Found

This status code is used by the SSC Server to inform the client that the request cannot be processed, because the requested addresses is hidden.

12.5 5xx Server Error

Response status codes beginning with the digit "5" indicate cases in which the SSC Server is aware that it has erred or is incapable of performing the request. Except when responding to a HEAD request, the SSC Server SHOULD include an entity containing an explanation of the error situation, and whether it is a temporary or permanent condition. User agents SHOULD display any included entity to the user. These response codes are applicable to any request method.

12.5.1 500 Internal Server Error

A generic error message, given when no more specific message is suitable.

12.5.2 501 Not Implemented

The SSC Server does not support the functionality required to fulfill the request. This is the appropriate response when the SSC Server does not recognize the request method and is not capable of supporting it for any resource.

12.5.3 503 Service Unavailable

The SSC Server is currently unable to handle the request due to a temporary overloading or maintenance of the SSC Server. The implication is that this is a temporary condition which will be alleviated after some delay.