Sennheiser Sound Control Protocol (SSC)

versatile command, control, and configuration for networked audio systems

Developer‘s guide for Digital 6000

EM 6000
L 6000

Sennheiser electronic GmbH & Co. KG

Am Labor 1, 30900 Wedemark, Germany, www.sennheiser.com
TI 1109 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
                        level_db 5  address "/audio/out1/level_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"
                        "level_db": 5,  boolean property  "audio.out1.level_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/level_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": { "level_db": 5 } } }
```

can be understood as a SSC Method Call of the SSC Method "/audio/out1/level_db" with the argument 5, presumably to set the level to that level, or as an SSC Method Reply message stating the current 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 level of "audio" of "out1" to 1 then looks like this:

```
TX: { "audio": { "out1": { "level_db": 1 }}}
RX: { "audio": { "out1": { "level_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": { "level_db": 20000 }}}
RX: { "audio": { "out1": { "level_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": { "level_db": null }}}
RX: { "audio": { "out1": { "level_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": { "level_db": 3, "label": null }}}
RX: { "audio": { "out1": { "level_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 level of OUT1 output of AUDIO module:

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

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

```
TX: { "audio": { "out1": { "level_db": 999 }}}
RX: { "audio": { "out1": { "level_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 notification rate parameters

Optional subscription request parameters related to notification rate:

- "min" minimum notification period (ms), 0=none, default 0
- "max" maximum notification period (ms), 0=none, default 0

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

5.2 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.

5.3 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.

5.4 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.

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.

The device implements UDP over IPv4.

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.

7. Developer’s Guide for EM 6000

This chapter describes in detail how a developer should use the SSC interface as implemented for the EM 6000.

7.1 Limitations

7.1.1 SSC Transport Layer

The SSC Server implemented for Digital 6000 devices supports only UDP/IP as transport protocol. All the devices support IPv4.

7.1.2 Subscriptions

The EM 6000 receiver supports SSC Method subscriptions from up to eight different SSC clients simultaneously.

8. SSC Method List (EM 6000)

8.1 /audio/out1/level_db

This method sets or returns the analogue audio level (AF out) of each cannel.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - units: dB
  - max: 18
  - min: -10
  - inc: 1

Example:

```
Tx: {"audio":{"out1":{"level_db":null}}}
Rx: {"audio":{"out1":{"level_db":3}}}
```

8.2 /audio/out2/level_db

see ""8.1 /audio/out1/level_db""

8.3 /rx1/audio_mute

This methods sets the audio mute of each receiver channel on or off.

- type: Read/Write
- value: boolean
- subscribable

8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99

This methods returns the System Bank/Channel frequencies of each receiver channel.

- type: Read only
- value: number

Example:

```
Tx: {"rx1":{"freq":{"b1":{"00":null}}}}
Rx: {"rx1":{"freq":{"b1":{"00":470200}}}}
```

8.5 /rx1/freq/b2/00 - /rx1/freq/b2/99

see ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.6 /rx1/freq/b3/00 - /rx1/freq/b3/99

see ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.7 /rx1/freq/b4/00 - /rx1/freq/b4/99

see ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.8 /rx1/freq/b5/00 - /rx1/freq/b5/99

see ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.9 /rx1/freq/b6/00 - /rx1/freq/b6/99

see ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99

This methods sets or returns the User Bank/Channel of each receiver channel.

- type: Read/Write
- value: number

Example:

```
Tx: {"rx1":{"freq":{"u1":{"00":null}}}}
Rx: {"rx1":{"freq":{"u1":{"00":470200}}}}
```

8.11 /rx1/freq/u2/00 - /rx1/freq/u2/99

see ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.12 /rx1/freq/u3/00 - /rx1/freq/u3/99

see ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.13 /rx1/freq/u4/00 - /rx1/freq/u4/99

see ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.14 /rx1/freq/u5/00 - /rx1/freq/u5/99

see ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.15 /rx1/freq/u6/00 - /rx1/freq/u6/99

see ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.16 /rx1/active_bank_channel

This method sets or returns active bank/channel for receiver channel 1.

- type: Read/Write
- value: [String Number]
- bank limits: "b1/B1","b2/B2","b3/B3","b4/B4","b5/B5","b6/B6","u1/U1","u2/U2","u3/U3","--"
- number limits: 0..99
- Notes: if bank is set to "--" (Manual) then channel is don't care.

Example:

```
Tx: {"rx1":{"active_bank_channel":["u2",63]}}
Rx: {"rx1":{"active_bank_channel":["u2",63]}}
Tx: {"rx1":{"active_bank_channel":null}}
Rx: {"rx1":{"active_bank_channel":["--",0]}} Manual mode active
```

8.17 /rx1/carrier

This method sets or returns the carrier Frequency in kHz for receiver channel 1.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - units: kHz
  - max: 713900
  - min: 470100
  - inc: 25

Example:

```
Tx: {"rx1":{"carrier":null}}
Rx: {"rx1":{" carrier ":470100}}
```

8.18 /rx1/scan/config

This method starts or stops a frequency fullscan. To start a fullscan session the config need the following structure [1, start_frequency_in_hz, stop_frequency_in_hz, stepsize_in_hz]. Frequencies and stepsize will be autocorrected, stepsize to a multiple of 25000 Hz. If starting succeeded the following array will be returned [1,start_frequency_in_hz, stop_frequency_in_hz, stepsize_in_hz, session_id]. The session_id is a number that represents the fullscan session which is started. If by some reason starting failed the returned value will be [0].

A running fullscan session can be stopped by sending [0]. The server return [0] and the server will terminate a runnings session (approx. ~100 ms). Note that a fullscan can only be started while it is not running! If you want to restart a fullscan while it is running you first have to stop it. You can check the session state by reading the node with null. The returned value will be [0] if no session is running and [1] if a session is currently running.

- type: Read/Write
- value: [Number]

Example:

```
Tx: {"rx1":{"scan":{"config":[1,470100000,713900000,25000]}}}
Rx: {"rx1":{"scan":{"config":[1,470100000,713900000,25000,33]}}}

Tx: {"rx1":{"scan":{"config":[0]}}}
Rx: {"rx1":{"scan":{"config":[0]}}}

Tx: {"rx1":{"scan":{"config":null}}}
Rx: {"rx1":{"scan":{"config":[1]}}}
```

8.19 /rx1/scan/result

If a session with ""8.18 /rx1/scan/config"" is started, this method gives the fullscan data blockwise back.

A returned array has the form:

[start_frequency_in_hz, stop_frequency_in_hz, stepsize_in_hz, block_start_frequency_in_hz, block_stop_frequency_in_hz, session_id, (RF1-ch1_val1),(RF2-ch1_val1),(RF1-ch1_val2),(RF2-ch1_val2),...].

See ""8.18 /rx1/scan/config"" for the meaning of the first three values. The values at the end of the array are rf level pairs, see ""8.100 /mm"" for mapping information. block_start_frequency_in_hz and block_stop_frequency_in_hz span their x-axis.

Subscribed clients will be notified with a rate of 100 single scans. If you read the node with null during a session the last available values are returned. If no session is running an empty array [] will be returned.

If you send an array of the form [block_start_frequency_in_hz, block_stop_frequency_in_hz] we will return an array within the requested range. This also works when no session is running.

- type: Read/Write
- value: [Number]
- subscribable

Example:

```
Tx:{"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
"lifetime":60},"rx1":{"scan":{"result":null}}}]}}}
Rx:{"rx1":{"scan":{"result":[470125000,713875000,25000,712625000,713875000,
33,39,30,45,35,46,34,45,35,46,34,46,35,42,32,34,27,30,25,29,25,29,26,29,26,
28,25,28,25,27,26,26,26,26,25,26,25,26,26,26,26,27,26,29,27,30,27,28,26,27,
26,27,25,29,27,28,27,27,26,27,26,26,26,26,26,26,25,27,26,27,27,29,28,28,28,
27,27,27,26,27,26,27,26,26,26,26,26,26,26,26,25,26,26,26,26,27,27,27,27,28,
28,29,27]}}}
Tx:{"rx1":{"scan":{"result":[470125000,500125000]}}}
Rx:{"rx1":{"scan":{"result":[470125000,713875000,25000,470125000,472600000,
1,54,77,51,74,45,65,40,55,40,54,39,51,38,48,38,47,38,46,38,44,38,43,38,42,
38,42,38,40,38,40,38,40,38,39,38,38,38,38,38,38,38,38,38,37,38,36,38,36,38,
36,39,36,39,36,38,34,40,35,40,35,40,34,40,33,40,32,42,32,46,51,54,64,49,54,
48,42,48,34,47,33,47,32,45,32,43,32,42,31,41,32,40,31,39,31,39,30,38,31,39,
32,39,31,38,32,38,32,38,32,39,32,38,33,38,32,38,31,38,31,38,31,39,31,38,31,
38,30,38,32,38,32,38,31,38,30,38,31,39,31,40,32,40,33,40,32,41,32,41,32,41,
32,40,31,40,30,41,30,42,30,42,31,41,30,40,30,40,30,40,31,40,31,40,31,39,32,
39,31,39,31,39,31,38,30,38,30,39,31,39,31,38,32,38,31,38,31,38,31,38,31,38,
31]}}}
```

8.20 /rx1/skx/type

This method returns the type of the transmitter, the frequency range and the lowest and highest values of the freq.

- type: Read only
- value: string
- subscribable

example:

```
Tx: {"rx1":{"skx":{"type":null}}}
Rx: {"rx1":{"skx":{"type":["SK A1-A4","470 MHz","558 MHz"]}}}
```

8.21 /rx1/skx/name

This method returns the transmitter name. An empty string indicates that transmitter is not present or doesn't send valid information.

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"name":null}}}]}}}
Rx: {"rx1":{"skx":{"name":"DEVICE2 "}}}.
```

8.22 /rx1/skx/lowcut

This method returns the Transmitter lowcut information as a string. An empty string indicates that transmitter is not present or doesn't send valid information.

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"lowcut":null}}}]}}}
Rx: {"rx1":{"skx":{"lowcut":"60 Hz"}}}.
```

8.23 /rx1/skx/gain

This method returns the transmitter gain information stored as a string. An empty string indicates that transmitter is not present or doesn't send valid information.

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"gain":null}}}]}}}
Rx: {"rx1":{"skx":{"gain":"42 dB"}}}.
```

8.24 /rx1/skx/display

This method returns the transmitter display information stored as a string. An empty string indicates that transmitter is not present or doesn't send valid information. Valid strings: "Frequency", "Name" and "Preset".

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"display":null}}}]}}}
Rx: {"rx1":{"skx":{"display":"Preset"}}}.
```

8.25 /rx1/skx/capsule

This method returns the transmitter capsule information stored as a string. An empty string indicates that skx is not present or doesn't send valid information. Valid strings: "unknown", "KK 205", "KK 204", "ME 9002", "ME 9004", "ME 9005", "MME 865", "MD 9235", "MMD 945", "MMD 935", "MMD 845", "MMD 835", "MMD 815-1", "MMK 965 sc", "MMK 965 c", "MMD42-I".

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"capsule":null}}}]}}}
Rx: {"rx1":{"skx":{"capsule":"MMD 935"}}}.
```

8.26 /rx1/skx/cable_emulation

This method returns the transmitter cable emulation information stored as a string. An empty string indicates the transmitter is not present or doesn't send valid information. Valid strings: "Line", "Type 1", "Type 2" and "Type 3".

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"display":null}}}]}}}
Rx: {"rx1":{"skx":{"display":"Preset"}}}.
```

8.27 /rx1/skx/battery

This method returns the transmitter battery information stored as a string array with 2 elements [State,Time]. An empty array indicates that the transmitter is not present or doesn't send valid battery information. States: {"100%", "70%", "30%", "low"}. Time notation: "x:xx" or "-:--" if time information is not available.

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"rx1":{"skx":{"battery":null}}}
Rx: {"rx1":{"skx":{"battery":[]}}}
Rx: {"rx1":{"skx":{"battery":["70%","5:12"]}}}
Rx: {"rx1":{"skx":{"battery":["low","-:--"]}}}.
```

8.28 /rx1/skx/autolock

This method returns the transmitter autolock information stored as a string ("On" or "Off"). An empty string indicates that the transmitter is not present or doesn't send valid information.

- type: Read only
- value: string
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":1000,
    "lifetime":60},"rx1":{"skx":{"autolock":null}}}]}}}
Rx: {"rx1":{"skx":{"autolock":"Off"}}}
```

8.29 /rx1/sync_settings/low_cut_frequency

This method sets or returns the sync settings for the low cut frequency.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - max: 4
  - min: 0
  - inc: 1
  - options , option_desc
    1. 0 , 30 Hz
    2. 1 , 60 Hz
    3. 2 , 80 Hz
    4. 3 , 100 Hz
    5. 4 , 120 Hz

Example:

```
Tx: {"rx1":{"sync_settings":{"low_cut_frequency":null}}}
Rx: {"rx1":{"sync_settings":{"low_cut_frequency":0}}}
```

8.30 /rx1/sync_settings/ignore_low_cut_frequency

This method sets or returns the sync settings to ignore lowcut frequency.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"sync_settings":{"ignore_low_cut_frequency":null}}}
Rx: {"rx1":{"sync_settings":{"ignore_low_cut_frequency":false}}}
```

8.31 /rx1/sync_settings/gain

This method sets or returns the sync settings for gain.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - units: dB
  - max: 60
  - min: -6
  - inc: 3

Example:

```
Tx: {"rx1":{"sync_settings":{"gain":null}}}
Rx: {"rx1":{"sync_settings":{"gain":0}}}
```

8.32 /rx1/sync_settings/ignore_gain

This method sets or returns sets the sync settings to ignore gain.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"sync_settings":{"ignore_gain":null}}}
Rx: {"rx1":{"sync_settings":{"gain":true}}}
```

8.33 /rx1/sync_settings/display

This method sets or returns the sync settings for the display (Frequency or Name).

- type: Read/Write
- value: Number
- limits
  - type: Number
  - max: 1
  - min: 0
  - inc: 1
  - options , option_desc
    1. 0 , Frequency
    2. 1 , Name

Example:

```
Tx: {"rx1":{"sync_settings":{"display":null}}}
Rx: {"rx1":{"sync_settings":{"display":0}}}
```

8.34 /rx1/sync_settings/ignore_display

This method sets or returns the sync settings to ignore display.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"sync_settings":{"ignore_display":null}}}
Rx: {"rx1":{"sync_settings":{"ignore_display":false}}}
```

8.35 /rx1/sync_settings/cable_emulation

This method sets or returns the sync settings for cable emulation.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - max: 3
  - min: 0
  - inc: 1
  - options , option_desc
    1. 0 , Line
    2. 1 , Type1
    3. 2 , Type2
    4. 3 , Type3

Example:

```
Tx: {"rx1":{"sync_settings":{"cable_emulation":null}}}
Rx: {"rx1":{"sync_settings":{"cable_emulation":2}}}
```

8.36 /rx1/sync_settings/ignore_cable_emulation

This method sets or returns the sync settings to ignore cable emulation.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"sync_settings":{"ignore_cable_emulation":null}}}
Rx: {"rx1":{"sync_settings":{"ignore_cable_emulation":false}}}
```

8.37 /rx1/sync_settings/auto_lock

This method sets or returns sync settings for auto lock.

- type: Read/Write
- value: Number
- limits
  - type: Number
  - max: 1
  - min: 0
  - inc: 1
  - options , option_desc
    1. 0 , Off
    2. 1 , On

Example:

```
Tx: {"rx1":{"sync_settings":{"auto_lock":null}}}
Rx: {"rx1":{"sync_settings":{"auto_lock":0}}}
```

8.38 /rx1/sync_settings/ignore_auto_lock

This method sets or returns the sync settings to ignore the auto lock.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"sync_settings":{"ignore_auto_lock":null}}}
Rx: {"rx1":{"sync_settings":{"ignore_auto_lock":false}}}
```

8.39 /rx1/walktest/start

This method starts the walk test feature. Any value set starts a new walk test and returns corresponding session id. Note that ids must match when when polling rx1/walktest/info. Reading simply returns 0.

- type: Read/Write
- value: Number

Example:

```
Tx: {"rx1":{"walktest":{"start":1}}}
Rx: {"rx1":{"walktest":{"start":1}}}
```

8.40 /rx1/walktest/info

This method returns the walk test info array:

[ID, RF1_min, RF1_max, RF1_peak, RF2_min, RF2_max, RF2_peak, LQI_min, LQI_max, AF_min, AF_max, AF_peak].

- type: Read only
- values: See ""8.100 /mm"" for mapping information. Note that all values except RF1_min and RF2_min are based on raw metering data. RF1_min and RF2_min are based on an attack/decay-filtered data.
- ID: id for running session.

Example:

```
Tx: {"rx1":{"walktest":{"info":null}}}
Rx: {"rx1":{"walktest":{"info":[1,169,177,0,153,160,0,230,255,128,190,0]}}}
```

8.41 /rx1/identify

This method pop up "Identified" window on EM6000 receiver channel 1 and let the triangle LED blink. Returns always true.

- type: Read-only
- value: boolean

Example:

```
Tx: {"rx1":{"identify":null}}
Rx: {"rx1":{"identify":true}}
```

8.42 /rx1/wsm_master_cnt

This method returns an "On Change Counter" for receiver channel 1: Sync Settings, Frequency Settings, AF Output, Channel Name, Encryption. The counter will be incremented when the values getting changed over the receiver UI.

- type: Read Only
- value: Number
- subscribable

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":10000, "max":0,"count":1000,
    "lifetime":60},"rx1":{"wsm_master_cnt":null}}]}}}
Rx:{"rx1":{"wsm_master_cnt":2}}
```

8.43 /rx1/testtone

This method returns the Testtone status. Testtone is on for values from -60 to 0 (dBFS). Tone is off for value=1.

- type: Read only
- value: number

Example:

```
Tx: {"rx1":{"testtone":null}}
Rx: {"rx1":{"testtone":-52}}
```

8.44 /rx1/name

This method sets or returns the receiver channel name.

- type: Read/Write
- value: String

Example:

```
Tx: {"rx1":{"name":null}}
Rx: {"rx1":{"name":"D-6000"}}
```

8.45 /rx1/encryption

This method sets or returns the encryption status.

- type: Read/Write
- value: boolean

Example:

```
Tx: {"rx1":{"encryption":null}}
Rx: {"rx1":{"name":true}}
```

8.46 /rx1/active_warnings

This method returns active warnings. Warnings: {RFPeak, AFPeak, LowSignal, NoLink, LowBattery, BadClock, NoClock, Aes256Error, AnTxYBNCShorted}. Note that BadClock and NoClock both are displayed as "Clock Error" in the EM.

- type: Read only
- value: [String]
- subscribable

Example:

```
Tx: {"rx1":{"active_warnings":null}}
Rx: {"rx1":{"active_warnings":["RFPeak","AFPeak"]}}
```

8.47 /rx1/active_status

This method returns active status and corresponding substatus stored as array with 2 elements. States: {SyncOK, SyncFail, Identified, SwUpdatePass, SwUpdateFail}. Substates: {None,{SyncResultACK,SyncResultAdapted,SyncResultNACK,SyncResultFreqRejected,SyncResultIncompatibleTx-,SyncResultCancelled,SyncResultTimeout},None}

- type: Read only
- value: [String]
- subscribable

Example:

```
Tx: {"rx1":{"active_status":null}}
Rx: {"rx1":{"active_status":["SyncFail","SyncResultTimeout"]}}
```

8.48 /rx2/audio_mute

See “/rx1/audio_mute”

8.49 /rx2/freq/b1/00 - /rx2/freq/b1/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.50 /rx2/freq/b2/00 - /rx2/freq/b2/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.51 /rx2/freq/b3/00 - /rx2/freq/b3/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.52 /rx2/freq/b4/00 - /rx2/freq/b4/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.53 /rx2/freq/b5/00 - /rx2/freq/b5/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.54 /rx2/freq/b6/00 - /rx2/freq/b6/99

See ""8.4 /rx1/freq/b1/00 - /rx1/freq/b1/99""

8.55 /rx2/freq/u1/00 - /rx2/freq/u1/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.56 /rx2/freq/u2/00 - /rx2/freq/u2/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.57 /rx2/freq/u3/00 - /rx2/freq/u3/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.58 /rx2/freq/u4/00 - /rx2/freq/u4/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.59 /rx2/freq/u5/00 - /rx2/freq/u5/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.60 /rx2/freq/u6/00 - /rx2/freq/u6/99

See ""8.10 /rx1/freq/u1/00 - /rx1/freq/u1/99""

8.61 /rx2/active_bank_channel

see ""8.16 /rx1/active_bank_channel""

8.62 /rx2/carrier

see ""8.17 /rx1/carrier""

8.63 /rx2/scan/config

See ""8.18 /rx1/scan/config""

8.64 /rx2/scan/result

See ""8.19 /rx1/scan/result""

8.65 /rx2/skx/type

See ""8.20 /rx1/skx/type""

8.66 /rx2/skx/name

See ""8.21 /rx1/skx/name""

8.67 /rx2/skx/lowcut

See ""8.22 /rx1/skx/lowcut""

8.68 /rx2/skx/gain

See ""8.23 /rx1/skx/gain""

8.69 /rx2/skx/display

See ""8.24 /rx1/skx/display""

8.70 /rx2/skx/capsule

See ""8.25 /rx1/skx/capsule""

8.71 /rx2/skx/cable_emulation

See ""8.26 /rx1/skx/cable_emulation""

8.72 /rx2/skx/battery

See ""8.27 /rx1/skx/battery""

8.73 /rx2/skx/autolock

See ""8.28 /rx1/skx/autolock""

8.74 /rx2/sync_settings/low_cut_frequency

See ""8.29 /rx1/sync_settings/low_cut_frequency""

8.75 /rx2/sync_settings/ignore_low_cut_frequency

See ""8.30 /rx1/sync_settings/ignore_low_cut_frequency""

8.76 /rx2/sync_settings/gain

See ""8.31 /rx1/sync_settings/gain""

8.77 /rx2/sync_settings/ignore_gain

See ""8.32 /rx1/sync_settings/ignore_gain""

8.78 /rx2/sync_settings/display

See ""8.33 /rx1/sync_settings/display""

8.79 /rx2/sync_settings/ignore_display

See ""8.34 /rx1/sync_settings/ignore_display""

8.80 /rx2/sync_settings/cable_emulation

See ""8.35 /rx1/sync_settings/cable_emulation""

8.81 /rx2/sync_settings/ignore_cable_emulation

See ""8.36 /rx1/sync_settings/ignore_cable_emulation""

8.82 /rx2/sync_settings/auto_lock

See ""8.37 /rx1/sync_settings/auto_lock""

8.83 /rx2/sync_settings/ignore_auto_lock

See ""8.38 /rx1/sync_settings/ignore_auto_lock""

8.84 /rx2/walktest/start

See ""8.39 /rx1/walktest/start""

8.85 /rx2/walktest/info

See ""8.40 /rx1/walktest/info""

8.86 /rx2/identify

This method returns and triggers the "Identified" Message on the receiver channel 2 display.

See ""8.41 /rx1/identify""

8.87 /rx2/wsm_master_cnt

See ""8.42 /rx1/wsm_master_cnt""

8.88 /rx2/testtone

See ""8.43 /rx1/testtone""

8.89 /rx2/name

See ""8.44 /rx1/name""

8.90 /rx2/encryption

See ""8.45 /rx1/encryption""

8.91 /rx2/active_warnings

See ""8.46 /rx1/active_warnings""

8.92 /rx2/active_status

See ""8.47 /rx1/active_status""

8.93 /sys/dante/version

This method returns the Dante module SW version.

- type: Read only
- value: String

Example:

```
Tx: {"sys":{"dante":{"version":null}}}
Rx: {"sys":{"dante":{"version":"1.1.1.21"}}}
```

8.94 /sys/dante/name

This method returns the Dante device advertised name.

- type: Read-only
- value: String

Example:

```
Tx: {"sys":{"dante":{"name":null}}}
Rx: {"sys":{"dante":{"name":"EM6000"}}}
```

8.95 /sys/wsm_master_cnt

This method returns an "On Change Counter" for: Network Settings, Device Name, System Name, Clock Settings, Booster Feed, Dante Name, Dante Network Settings. The counter will be incremented when the values getting changed over the receiver UI.

- type: Read Only
- value: Number
- subscribable

Example:

```
Tx: {"sys":{"wsm_master_cnt":null}}
Rx: {"sys":{"wsm_master_cnt":4}}
```

8.96 /sys/clock_frequency_measured

This method returns the clock frequency measured on the Wordclock input of the receiver.

Possible values are ['< 15378 Hz', '15378 Hz'...'1999998 Hz', '> 1999998 Hz'].

- type: Read only
- value: string

Example:

```
Tx: {"sys":{"clock_frequency_measured":null}}
Rx: {"sys":{"clock_frequency_measured":"48000 Hz"}}
```

8.97 /sys/clock

This method returns the Clock mode for the Wordclock input.

- type: Read/Write
- value: number
- limits
  - type: Number
  - options , option_desc
    1. 1 , Clock State Internal 48kHz
    2. 2 , Clock State Internal 96kHz
    3. 3 , Clock State External
    4. 4 , Clock State Internal MAN

Example:

```
Tx: {"sys":{"clock":null}}
Rx: {"sys":{"clock":2}}
```

8.98 /sys/brightness

This method sets or returns the brightness level of both displays in percent.

- type: Read/Write
- value: number

Example:

```
Tx: {"sys":{"brightness":null}}
Rx: {"sys":{"brightness":80}}
```

8.99 /sys/booster

This method sets and returns the boosterfeed status.

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"sys":{"booster":null}}
Rx: {"sys":{"booster":true}}
```

8.100 /mm

Metering array: [[(RF1-ch1),(RF1-PEAK-ch1),(RF2-ch1),(RF2-PEAK-ch1),(DIV1-ch1),(DIV2-ch1),(LQI-ch1),(AF-ch1),(AF-PEAK-ch1)],[(RF1-ch2),(RF1-PEAK-ch2),(RF2-ch2),(RF2-PEAK-ch2),(DIV1-ch2),(DIV2-ch2),(LQI-ch2),(AF-ch2),(AF-PEAK-ch2)]].

- type: Read only
- value: 2x9 array of bytes. Array contains raw metering data for both channels.
- RF1/2: RF level for antenna 1/2. Value in dBm=(Value-255)/2
- RF1/2-PEAK: Peak indicates a digital clip in the RF section (for at least one second). 0:no peak, 1:peak.
- DIV1/2: Diversity for antenna 1/2. 0:antenna off, 1:antenna on
- LQI: (Audio)Link Quality Indicator. 255 means best, 0 worst.
- AF: AF level (full scale audio level). dBFS = (Value+1)/2-128;
- PEAK: Peak indicates a digital clip in the audio section (for at least one second). 0:no peak, 1:peak
- Note that this node can be subscribed, i.e.: {"osc":{"state":{"subscribe":[{"#":{"min":480, "max":480,"lifetime":20,"count":1000},"mm":null}]}}}.
- subscribable

Example:

```
{"mm":[[0,0,0,0,0,0,0,0,0],[83,0,53,0,1,1,128,165,0]]}
```

8.101 /device/identity/version

This method returns the SW version information.

- type: Read-only
- value: String

Example:

```
Tx: {"device":{"identity":{"version":null}}}
Rx: {"device":{"identity":{"version":"1.1.4.74"}}}
```

8.102 /device/identity/vendor

This method returns the vondor string.

- type: Read-only
- value: String

Example:

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

8.103 /device/identity/product

This method returns the Product label "EM6000" or "EM6000-Dante".

- type: Read-only
- value: String

Example:

```
Tx: {"device":{"identity":{"product":null}}}
Rx: {"device":{"identity":{"vendor":"EM6000-Dante"}}}
```

8.104 /device/network/ether/macs

This method returns a list of MAC addresses which are used in the receiver.

- type: Read-only
- value: [String]

Example:

```
Tx: {"device":{"network":{"ether":{"macs":null}}}}
Rx: {"device":{"network":{"ether":{"macs":["00:1b:66:xx:xx:xx",
    "00:1d:c1:xx:xx:xx"]}}}}
```

8.105 /device/network/ether/interfaces

This method returns a list of all ethernet interfaces.

- type: Read-only
- value: [String]

Example:

```
Tx: {"device":{"network":{"ether":{"interfaces":null}}}}
Rx: {"device":{"network":{"ether":{"interfaces":["CONTROL","DANTE"]}}}}
```

8.106 /device/network/ipv4/auto

This method sets or returns the IP mode "Auto". This settings configure the IP setting for the receiver automatically. (true: use DHCP and ZeroConf (Auto-IP); false: set ip address/netmask/gateway manually).

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"device":{"network":{"ipv4":{"auto":null}}}}
Rx: {"device":{"network":{"ipv4":{"auto":true}}}}
```

8.107 /device/network/ipv4/mdns

This method sets and returns the mdns mode (auto discovery) of the receiver. This mode is an mdns only network configuration without DHCP and ZeroConf (link local) (true: mdns active and set ip address/netmask/gateway manually; false: manual mode active set ip address/netmask/gateway manually).

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"device":{"network":{"ipv4":{"mdns":null}}}}
Rx: {"device":{"network":{"ipv4":{"mdns":true}}}}
```

8.108 /device/network/ipv4/interfaces

This method returns a list of ipv4 interface indices.

- type: Read-only
- value: [Number]

Example:

```
Tx: {"device":{"network":{"ipv4":{"interfaces":null}}}}
Rx: {"device":{"network":{"ipv4":{"interfaces":[0]}}}}
```

8.109 /device/network/ipv4/static_ipaddr

This method sets and returns the IPv4 settings for IP address in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_ipaddr":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_ipaddr":"192.168.1.1"}}}}
```

8.110 /device/network/ipv4/static_netmask

This method sets and returns the IPv4 settings for IP netmask in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_netmask":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_netmask":"255.255.255.0"}}}}
```

8.111 /device/network/ipv4/static_gateway

This method sets and returns the IPv4 settings for IP gateway in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_gateway":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_gateway":"192.168.0.1"}}}}
```

8.112 /device/network/ipv4/ipaddr

This method returns the actual IP address for all modes.

- type: Read-only
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"ipaddr":null}}}}
Rx: {"device":{"network":{"ipv4":{"ipaddr":"192.168.0.1"}}}}
```

8.113 /device/network/ipv4/netmask

This method returns the actual IP netmask for all modes.

- type: Read-only
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"netmask":null}}}}
Rx: {"device":{"network":{"ipv4":{"netmask":"255.255.255.0"}}}}
```

8.114 /device/network/ipv4/gateway

This method returns the actual IP gateway for all modes.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"gateway":null}}}}
Rx: {"device":{"network":{"ipv4":{"gateway":"192.168.1.1"}}}}
```

8.115 /device/network/ipv4_dante/auto

This method sets and returns the mode for the Ethernet configuration in the Dante™ audio stream, if the EM6000 DANTE variant is present. All the time auto discovery is active (mdns)

(true: use DHCP or ZeroConf (link local); false: set ip address/netmask/gateway manually).

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"device":{"network":{"ipv4_dante":{"auto":null}}}}
Rx: {"device":{"network":{"ipv4_dante":{"auto":true}}}}
```

8.116 /device/network/ipv4_dante/ipaddr

This method sets and returns the IPv4 address for the Dante™ audio stream, if the EM6000 DANTE variant is present.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4_dante":{"ipaddr":null}}}}
Rx: {"device":{"network":{"ipv4_dante":{"ipaddr":"192.168.1.1"}}}}
```

8.117 /device/network/ipv4_dante/netmask

This method sets and returns the IPv4 netmask for the Dante™ audio stream, if the EM6000 DANTE variant is present.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4_dante":{"netmask":null}}}}
Rx: {"device":{"network":{"ipv4_dante":{"netmask":"255.255.255.0"}}}}
```

8.118 /device/network/ipv4_dante/gateway

This method sets the IPV4 gateway for the Dante™ audio stream, if the EM6000 DANTE variant is present.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4_dante":{"gateway":null}}}}
Rx: {"device":{"network":{"ipv4_dante":{"gateway":"255.255.255.0"}}}}
```

8.119 /device/network/ipv4_dante/settings

This method sets the IPv4 configuration at once.

- type: Write
- value: String Array

Example:

```
Tx: {"device":{"network":{"ipv4_dante":{"settings":["manual",
    "192.168.1.203","255.255.0.0", "192.168.0.1"]}}}}
Rx: {"device":{"network":{"ipv4_dante":{"settings":["manual",
    "192.168.1.203","255.255.0.0", "192.168.0.1"]}}}}
```

8.120 /device/name

User-settable persistent device name (Device ID). This name should be the preferred, and most convenient way for the customer to identify devices. The device name is reflected in the EM6000 menu as well.

The Device ID is reflected in the device discovery. The MAC address will be added to the device name automatically to avoid naming conflicts in the device discovery.

(in the example is the MAC address shown as xxxxxxxxxxxx)

- type: Read/Write
- value: String

Example:

```
TX: {"device":{"name":null}}
RX: {"device":{"name":"Digital6000-001b66xxxxxx"}}
TX: {"device":{"name":"EM6000"}}
RX: {"device":{"name":"EM6000-001b66xxxxxx"}}
```

8.121 /device/language

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

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

Languages are encoded with 2-3-letter-codes as per the locale convention. 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.

- type: Read-only
- value: [String]

Example:

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

8.122 /osc/state/prettyprint

SSC reply output style is not supported. Returns false.

8.123 /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.

Examples:

```
Tx: {"osc":{"state":{"subscribe":[{"sys":{"wsm_master_cnt":null}}]}}}
Rx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,
    "lifetime":10,"count":1000},"sys":{"wsm_master_cnt":null}}]}}}
Rx: {"sys":{"wsm_master_cnt":1}}
```

Subscribing with non-default parameters:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"min":1000, "max":0,"count":1000,
    "lifetime":60},"sys":{"wsm_master_cnt":null}}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"min":1000, "max":0,"count":1000,
    "lifetime":60},"sys":{"wsm_master_cnt":null}}]}}}
```

Parameter value changes:

```
RX: {"sys":{"wsm_master_cnt":1}}
RX: {"sys":{"wsm_master_cnt":2}}
RX: {"sys":{"wsm_master_cnt":3}}
RX: {"sys":{"wsm_master_cnt":4}}
```

Subscription terminates:

```
RX: {"osc":{"error":[{"sys":{"wsm_master_cnt":[310,{"desc":
    "subscription terminates"}]}}]}}
```

8.124 /osc/feature/timetag

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}}}
```

8.125 /osc/feature/baseaddr

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}}}
```

8.126 /osc/feature/subscription

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}}}
```

8.127 /osc/feature/pattern

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":"*?"}}}
```

8.128 /osc/limits

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 "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":[{"rx1":{"carrier":null}}]}}
RX: {"osc":{"limits":[{"rx1":{"carrier":[{"type":"Number","max":713900,
    "min":470100,"inc":25,"units":"kHz"}]}}]}}
```

8.129 /osc/schema

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":{},"rx1":{},"rx2":{},"sys":{},"device":{},
    "osc":{},"mm":null}]}}
TX: {"osc":{"schema":[{"audio":null}]}}
RX: {"osc":{"schema":[{"audio":{"out1":{},"ou2":{}}}]}}
```

8.130 /osc/version

Description: Reports the SSC version implemented in the server.

Example:

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

8.131 /osc/xid

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.

Example:

```
TX: {"osc":{"xid":1234567890},"sys":{"booster":null}}
RX: {"osc":{"xid":1234567890},"sys":{"booster":false}}
```

8.132 /osc/ping

SSC Ping.

8.133 /osc/error

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,{"desc":"address not found"}]}}]}}
TX: {"sys":{"booster":123456789}}
RX: {"osc":{"error":[{"sys":{"booster":[406,{"desc":"not acceptable"}]}}]}}
```

9. SSC Error List (EM 6000)

- 100 : continue
- 102 : processing
- 200 : OK
- 201 : Created
- 202 : Adapted
- 210 : Partial Success
- 310 : subscription terminates
- 400 : message not understood
- 401 : authorisation needed
- 403 : forbidden
- 404 : address not found
- 406 : not acceptable
- 408 : request time out
- 409 : conflict
- 410 : gone
- 413 : request too long
- 414 : request too complex
- 422 : unprocessable entity
- 423 : locked
- 424 : failed dependency
- 450 : answer too long
- 454 : parameter address not found
- 500 : internal server error
- 501 : not implemented
- 503 : service unavailable

10. Developer’s Guide for L 6000

This chapter describes in detail how a developer should use the SSC interface as implemented for the L 6000.

10.1 Limitations

10.1.1 SSC Transport Layer

The SSC Server implemented for Digital 6000 devices supports only UDP/IP as transport protocol. All the devices support IPv4.

10.1.2 Subscriptions

The L 6000 charger supports SSC Method subscriptions from up to sixteen different SSC clients simultaneously.

11. SSC Method List (L 6000)

11.1 /slot1/subslot1/led

This method returns the subslot LED status.

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

Example:

```
Tx: {"slot1":{"subslot1":{"led":null}}}
Rx: {"slot1":{"subslot1":{"led":"GREEN"}}}
```

11.2 /slot1/subslot1/identify

This method identifies a subslot by changing colours of leds.

- type: Read-only

Example:

```
Tx: {"slot1":{"subslot1":{"identify":null}}}
Rx: {"slot1":{"subslot1":{"identify":true}}}
```

11.3 /slot1/subslot1/accu_parameter

This method returns the accu parameter as follows: [temperature [°C], voltage [mV], capacity [mAh], current [mA], energy [mWh], operating time [h], operating time [min], state of charge [%], cycle count [-], state of health [%], time to full [h], time to full [min]] Please note: The returned array is only valid if the battery is present.

- type: Array
- value: Number
- subscribable

Example:

```
Tx: {"slot1":{"subslot1":{"accu_parameter":null}}}
Rx: {"slot1":{"subslot1":{"accu_parameter":[24.0,3939,1459,0,5235,4,
    0,69,0,100,340,30]}}}
```

11.4 /slot1/subslot1/accu_detection

This method returns the battery detection status.

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

Example:

```
Tx: {"slot1":{"subslot1":{"accu_detection":null}}}
Rx: {"slot1":{"subslot1":{"accu_detection":true}}}
```

11.5 /slot1/subslot2/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.6 /slot1/subslot2/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.7 /slot1/subslot2/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.8 /slot1/subslot2/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.9 /slot1/type

This method returns the type of slot.

- type: Read
- value: Number
- subscribable
- limits
  - type: Number
  - options , option_desc
    1. 0 , dummy panel (not connected)
    2. 1 , LM6060 slot
    3. 2 , LM6061 slot
    4. 3 , unknown / fault

Example:

```
Tx: {"slot1":{"type":null}}
Rx: {"slot1":{"type":1}}}
```

11.10 /slot2/subslot1/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.11 /slot2/subslot1/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.12 /slot2/subslot1/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.13 /slot2/subslot1/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.14 /slot2/subslot2/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.15 /slot2/subslot2/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.16 /slot2/subslot2/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.17 /slot2/subslot2/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.18 /slot2/type

See ""11.9 /slot1/type"".

- type: Read
- value: Number
- subscribable
- limits
  - type: Number
  - options , option_desc
    1. 0 , dummy panel (not connected)
    2. 1 , LM6060 slot
    3. 2 , LM6061 slot
    4. 3 , unknown / fault

11.19 /slot3/subslot1/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.20 /slot3/subslot1/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.21 /slot3/subslot1/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.22 /slot3/subslot1/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.23 /slot3/subslot2/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.24 /slot3/subslot2/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.25 /slot3/subslot2/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.26 /slot3/subslot2/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.27 /slot3/type

See ""11.9 /slot1/type"".

- type: Read
- value: Number
- subscribable
- limits
  - type: Number
  - options , option_desc
    1. 0 , dummy panel (not connected)
    2. 1 , LM6060 slot
    3. 2 , LM6061 slot
    4. 3 , unknown / fault

11.28 /slot4/subslot1/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.29 /slot4/subslot1/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.30 /slot4/subslot1/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.31 /slot4/subslot1/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.32 /slot4/subslot2/led

See ""11.1 /slot1/subslot1/led"".

- type: Read
- value: String
- subscribable
- limits
  - type: String
  - options , option_desc
    1. OFF , No battery detected
    2. GREEN , State of charge in range 97% - 100%
    3. GREEN_FLASHING , State of charge in range 81% - 96%
    4. YELLOW , State of charge in range 0% - 80%
    5. YELLOW_FLASHING , Battery in regeneration
    6. RED , Battery defect
    7. RED_FLASHING , Temperature out of range (normal operating temp: 0°C - 50°C)
    8. GREEN_RED_FLASHING , Storage Mode: no battery detected
    9. YELLOW_RED_FLASHING , Storage Mode: when battery is out of storage capacity (69% - 71%)
    10. YELLOW_GREEN_FLASHING , Storage Mode: battery has storage capacity (69% - 71%)
    11. DEV_IDENTIFY , Identify battery

11.33 /slot4/subslot2/identify

See ""11.2 /slot1/subslot1/identify"".

- type: Read-only

11.34 /slot4/subslot2/accu_parameter

See ""11.3 /slot1/subslot1/accu_parameter"".

- type: Array
- value: Number
- subscribable

11.35 /slot4/subslot2/accu_detection

See ""11.4 /slot1/subslot1/accu_detection"".

- type: Read
- value: Bool
- subscribable
- limits
  - type: Boolean

11.36 /slot4/type

See ""11.9 /slot1/type"".

- type: Read
- value: Number
- subscribable
- limits
  - type: Number
  - options , option_desc
    1. 0 , dummy panel (not connected)
    2. 1 , LM6060 slot
    3. 2 , LM6061 slot
    4. 3 , unknown / fault

11.37 /device/identity/version

This method returns the SW version information.

- type: Read-only
- value: String

Example:

```
Tx: {"device":{"identity":{"version":null}}}
Rx: {"device":{"identity":{"version":"1.1.3.29"}}}
```

11.38 /device/identity/vendor

This method returns the vondor string.

- type: Read-only
- value: String

Example:

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

11.39 /device/identity/product

This method returns the Product label "L6000".

- type: Read-only
- value: String
- subscribable

Example:

```
Tx: {"device":{"identity":{"product":null}}}
Rx: {"device":{"identity":{"product":"L6000"}}}
```

11.40 /device/network/ether/macs

This method returns a list of MAC addresses which are used in the device.

- type: Read-only
- value: [String]

Example:

```
Tx: {"device":{"network":{"ether":{"macs":null}}}}
Rx: {"device":{"network":{"ether":{"macs":["00:1b:66:xx:xx:xx"]}}}}
```

11.41 /device/network/ether/interfaces

This method returns a list of all ethernet interfaces.

- type: Read-only
- value: [String]

Example:

```
Tx: {"device":{"network":{"ether":{"interfaces":null}}}}
Rx: {"device":{"network":{"ether":{"interfaces":["CONTROL"]}}}}
```

11.42 /device/network/ipv4/auto

This method sets or returns the IP mode "Auto". This settings configure the IP setting for the receiver automatically. (true: use DHCP and ZeroConf (Auto-IP); false: set ip address/netmask/gateway manually).

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"device":{"network":{"ipv4":{"auto":null}}}}
Rx: {"device":{"network":{"ipv4":{"auto":true}}}}
```

11.43 /device/network/ipv4/mdns

This method sets and returns the mdns mode (auto discovery) of the receiver. This mode is an mdns only network configuration without DHCP and ZeroConf (link local) (true: mdns active and set ip address/netmask/gateway manually; false: manual mode active set ip address/netmask/gateway manually).

- type: Read/Write
- value: Boolean

Example:

```
Tx: {"device":{"network":{"ipv4":{"mdns":null}}}}
Rx: {"device":{"network":{"ipv4":{"mdns":true}}}}
```

11.44 /device/network/ipv4/interfaces

This method returns a list of ipv4 interface indices.

- type: Read-only
- value: [Number]

Example:

```
Tx: {"device":{"network":{"ipv4":{"interfaces":null}}}}
Rx: {"device":{"network":{"ipv4":{"interfaces":[0]}}}}
```

11.45 /device/network/ipv4/static_ipaddr

This method sets and returns the IPv4 settings for IP address in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_ipaddr":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_ipaddr":"192.168.1.1"}}}}
```

11.46 /device/network/ipv4/static_netmask

This method sets and returns the IPv4 settings for IP netmask in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_netmask":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_netmask":"255.255.255.0"}}}}
```

11.47 /device/network/ipv4/static_gateway

This method sets and returns the IPv4 settings for IP gateway in manual mode.

- type: Read/Write
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"static_gateway":null}}}}
Rx: {"device":{"network":{"ipv4":{"static_gateway":"192.168.1.2"}}}}
```

11.48 /device/network/ipv4/ipaddr

This method returns the actual IP address for all modes.

- type: Read/Write*
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"ipaddr":null}}}}
Rx: {"device":{"network":{"ipv4":{"ipaddr":"192.168.0.1"}}}}
```

11.49 /device/network/ipv4/netmask

This method returns the actual IP netmask for all modes.

- type: Read/Write*
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"netmask":null}}}}
Rx: {"device":{"network":{"ipv4":{"netmask":"255.255.255.0"}}}}
```

11.50 /device/network/ipv4/gateway

This method returns the actual IP gateway for all modes.

- type: Read/Write*
- value: String

Example:

```
Tx: {"device":{"network":{"ipv4":{"gateway":null}}}}
Rx: {"device":{"network":{"ipv4":{"gateway":"192.168.0.2"}}}}
```

11.51 /device/language

User-settable value determining the language to be used for values returned by the server which are meant to be displayed to the user. Examples are /osc/error/desc, /osc/limits/.../desc, /osc/limits/.../option_desc. An SSC Client can determine the possible language options by querying /osc/limits/device/language. Languages are encoded with 2-3-letter-codes as per the locale convention. 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.

- type: Read-only
- value: [String]

Example:

```
Tx: {"device":{"language":null}}
Rx: {"device":{"language":["en_GB"]}}
```

11.52 /device/warnings

This method returns the number(s) of active warnings.

- type: Read-only
- value: Array of Numbers
- subscribable
- limits
  - – type: Number
  - – options , option_desc
    1. 0 , accu too hot slot/subslot 1/1
    2. 1 , accu too hot slot/subslot 1/2
    3. 2 , accu too hot slot/subslot 2/1
    4. 3 , accu too hot slot/subslot 2/2
    5. 4 , accu too hot slot/subslot 3/1
    6. 5 , accu too hot slot/subslot 3/2
    7. 6 , accu too hot slot/subslot 4/1
    8. 7 , accu too hot slot/subslot 4/2
    9. 8 , accu in regeneration slot/subslot 1/1
    10. 9 , accu in regeneration slot/subslot 1/2
    11. 10 , accu in regeneration slot/subslot 2/1
    12. 11 , accu in regeneration slot/subslot 2/2
    13. 12 , accu in regeneration slot/subslot 3/1
    14. 13 , accu in regeneration slot/subslot 3/2
    15. 14 , accu in regeneration slot/subslot 4/1
    16. 15 , accu in regeneration slot/subslot 4/2
    17. 16 , accu defect slot/subslot 1/1
    18. 17 , accu defect slot/subslot 1/2
    19. 18 , accu defect slot/subslot 2/1
    20. 19 , accu defect slot/subslot 2/2
    21. 20 , accu defect slot/subslot 3/1
    22. 21 , accu defect slot/subslot 3/2
    23. 22 , accu defect slot/subslot 4/1
    24. 23 , accu defect slot/subslot 4/2
    25. 24 , fan 1 defect
    26. 25 , fan 2 defect
    27. 26 , fan 3 defect
    28. 27 , fan 4 defect
    29. 28 , device too hot
    30. 29 , accu too cold slot/subslot 1/1
    31. 30 , accu too cold slot/subslot 1/2
    32. 31 , accu too cold slot/subslot 2/1
    33. 32 , accu too cold slot/subslot 2/2
    34. 33 , accu too cold slot/subslot 3/1
    35. 34 , accu too cold slot/subslot 3/2
    36. 35 , accu too cold slot/subslot 4/1
    37. 36 , accu too cold slot/subslot 4/2

Example:

```
Tx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0,"count":
    1000, "lifetime":60},"device":{"warnings"::null}}]}}}
Rx: {"device":{"warnings":[3,5]}}
```

11.53 /device/storage_mode

This method sets and returns the the storage mode. When the L6000 is in the storagemode the batteries will be charged or discharged until 70% capasity.

- type: Read-only

Example:

```
Tx: {"device":{"storage_mode":null}}
Rx: {"device":{"storage_mode":true}}}
```

11.54 /device/name

User-settable persistent device name. This name should be the preferred, and most convenient way for the customer to identify devices. The device name is reflected in the device discovery. The MAC address will be added to the device name automatically to avoid naming conflicts in the device discovery. (in the example the MAC address is shown as 001b66xxxxxx)

- type: Read/Write
- value: String

Example:

```
TX: {"device":{"name":null}}
RX: {"device":{"name":"Digital6000-001b66xxxxxx"}}
TX: {"device":{"name":"EM6000"}}
RX: {"device":{"name":"EM6000-001b66xxxxxx"}}
```

11.55 /device/identify

This method is to identify the L6000 by changing colours of leds.

- type: Read-only

Example:

```
Tx: {"device":{"identify":null}}
Rx: {"device":{"identify":true}}}
```

11.56 /osc/state/prettyprint

SSC reply output style is not supported. Returns false.

11.57 /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.

Examples:

```
Tx: {"osc":{"state":{"subscribe":[{"slot1":{"subslot1":
    {"accu_detection":null}}}]}}}
Rx: {"osc":{"state":{"subscribe":[{"#":{"min":0, "max":0, "lifetime":10,
    "count":1000},"slot1":{"subslot1":{"accu_detection":null}}}]}}}
Rx: {"slot1":{"subslot1":{"accu_detection":true}}}
```

Subscribing with non-default parameters:

```
TX: {"osc":{"state":{"subscribe":[{"#":{"min":1000, "max":0,"count":1000,
    "lifetime":60},"slot1":{"subslot1":{"accu_detection":}}}]}}}
RX: {"osc":{"state":{"subscribe":[{"#":{"min":1000, "max":0,"count":1000,
    "lifetime":60},"slot1":{"subslot1":{"accu_detection":}}}]}}}
RX: {"slot1":{"subslot1":{"accu_detection":true}}}
```

Parameter value changes:

```
RX: {"slot1":{"subslot1":{"accu_detection":true}}}
RX: {"slot1":{"subslot1":{"accu_detection":false}}}
RX: {"slot1":{"subslot1":{"accu_detection":true}}}
RX: {"slot1":{"subslot1":{"accu_detection":false}}}
```

Subscription terminates:

```
RX: {"osc":{"error":[{"slot1":{"subclot1":{"accu_detection":[310,{"desc":
    "subscription terminates"}]}}]}}
```

11.58 /osc/feature/timetag

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}}}
```

11.59 /osc/feature/baseaddr

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}}}
```

11.60 /osc/feature/subscription

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}}}
```

11.61 /osc/feature/pattern

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":"*?"}}}
```

11.62 /osc/limits

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 "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":[{"slot1":{"type":null}}]}}
RX: {"osc":{"limits":[{"slot1":{"type":[{"type":"Number",
    "option":[0,1,2,3],"option_desc":["dummy panel (not connected)",
    "L 6060 slot","L 6061 slot","unknown / fault"]}]}}]}}
```

11.63 /osc/schema

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":[{"slot1":{},"slot2":{},"slot3":{},"slot4":{},
    "device":{},"sys":{},"osc":{}}]}}
TX: {"osc":{"schema":[{"audio":null}]}}
RX: {"osc":{"schema":[{"audio":{"out1":{},"ou2":{}}}]}}
```

11.64 /osc/version

Description: Reports the SSC version implemented in the server.

Example:

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

11.65 /osc/xid

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.

Example:

```
TX: {"osc":{"xid":1234567890},"slot1":{"type":null}}
RX: {"osc":{"xid":1234567890},"slot1":{"type":2}}
```

11.66 /osc/ping

SSC Ping.

11.67 /osc/error

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,{"desc":"address not found"}]}}]}}
TX: {"slot1":{"type":123456789}}
RX: {"osc":{"error":[{"slot1":{"type":[406,{"desc":"not acceptable"}]}}]}}
<-- read only
```

12. SSC Error List (L 6000)

- 100 : continue
- 102 : processing
- 200 : OK
- 201 : Created
- 202 : Adapted
- 210 : Partial Success
- 310 : subscription terminates
- 400 : message not understood
- 401 : authorisation needed
- 403 : forbidden
- 404 : address not found
- 406 : not acceptable
- 408 : request time out
- 409 : conflict
- 410 : gone
- 413 : request too long
- 414 : request too complex
- 422 : unprocessable entity
- 423 : locked
- 424 : failed dependency
- 450 : answer too long
- 454 : parameter address not found
- 500 : internal server error
- 501 : not implemented
- 503 : service unavailable