Contents

Introduction

Smaart exposes capabilities and control to external and/or remote applications through its application-programming interface (API). The API provides a mechanism for finding instances of Smaart on a local machine or on a network. Once identified, an application can use the WebSocket protocol to drive Smaart as well as query for measurement data.

The content that follows defines the schema of the requests made of, and the responses made by, the Smaart API server.

Libraries which implement the WebSocket protocol exist for many languages and platforms. Some examples include: websockets for Python, Lightstreamer for Java, WebSocketListener for .NET, Boost.beast for C++ and noPoll for ANSI C.

Discovery

Smaart API server listens for broadcast (UDP) queries on a given port. The query is expected to have the following structure:

```
bytes 0-3 signature of the product to which a connection is desired
bytes 4-7 the TCP port on which the client is expecting a response
```

When the server receives discovery queries, it responds by sending the same structure as above to the client (via TCP connection), with the same signature, but with the port the server is utilizing for API communication (as shown in the API tab of the Options dialog). Even though the structure has room for a 32-bit port number, it is expected to be a 16-bit unsigned number. The structure is expected to be in little-endian format, as are all requests and responses.

The following is a list of products with the expected signature and discovery port:

Product	Signature	Port
Smaart	0x656E7544	25752
Smaart Di	0x73646179	25752
Smaart SPL	0x5A4E5254	25523

Legacy API Support

API v1 (Smaart 7) and API v2 (Smaart 8) are still supported. The discovery details are different for each version of the API.

API version	Product	Signature	Port
3	Smaart 8.3	0x656E7544	25752
	Smaart Di 2.1	0x73646179	25752
2	Smaart 8.0	0x49504154	25052
	Smaart Di 2.0	0x5A495247	25052
1	Smaart 7	0x70696E67	25000
	Smaart Di 1.0 (v7 Di)	0x64696E67	25000

Connection & Communication

Once the address and port of the server is known, a standard WebSocket connection (RFC6455) is made to interact with the API server. Requests of the Smaart server are made in the form of JSON messages, as are the responses.

Supported Versions

A list of supported API versions is returned when any query is sent to the root of the server (`ws://<address>:<port>`). The response has the following format:

```
{
    "supportedApiVersions": [
        { "3": "/api/v3/" },
        { "2": "" },
        { "1": "" }
    ]
}
```

The rest of this document refers to Smaart API v3, to which a connection is made via:

```
ws://<Smaart server address>:<Smaart server port>/api/v3/
```

Format of Requests

The server expects requests that are comprised soley of parsable JSON of the form:

```
{
    "sequenceNumber": 42,
    "action": "set",
    "target": "signalGenerator",
    "properties": [ { "gain": -22 } ]
}
```

Request Parameters

Name	Type	Description
sequenceNumber	*integer*	optional number used to synchronize server responses with client requests
action	*string*	indicates whether the request is a query (get) or a command (set, capture, etc.).
target	*string* or *object*	optional parameter that is either a string or an object containing key-value pairs (root/server assumed if omitted)
properties	*array*	required array for commands containing key-value pairs indicating the property name and desired value

Format of Responses

Responses vary depending on the query/command, but are always of the form:

```
{
    "sequenceNumber": 42,
    "response": {
        ...
    }
}
```

The sequence number is only included if it is specified in the request, and is non-zero. A sequence number will not be included in examples that follow. The order of attributes in lists and arrays may or may not be sorted.

Errors

If a request is not successful, the response will be of the form:

```
{
    "response": {
        "error": "<error message>"
    }
}
```

The following are possible errors returned from requests:

Error	Condition
parse error	an error occurred while parsing the JSON request
timeout	the timeout elapsed while an asynchronous operation was being carried out (making a window or tab active, etc.)
unknown target	the target of the request was not recognized - name misspelled, measurement was specified that was not in the active tab/window. etc.
unknown action	the action was not recognized by the target
unkown property	the property does not apply to the target
unknown value	the value does not apply to the property
read only	an attempt was made to ‘set’ a property that is read-only
not implemented	a hole in the implementation, or an unreasonable request has been made
signal generator required	an attempt was made to start a measurement that requires the signal generator to be active - this will not be returned when attempting to start all measurements of a tab
measurement not active	an attempt to find the delay of a measurement was made while the measurement was not active and the `automaticallyStart` property was not set

authentication required	the API requires a password
incorrect password	the submitted password was incorrect
internal error	theoretical impossibility

Streaming Data

When opening a WebSocket connection to the value contained in the `streamEndpoint` or `lirStreamEndpoint` property of a measurement or calibrated input, the API server will continuously send live measurement data. Although stream-altering commands may be sent while connected to a stream, no response will be sent from these commands.

API Specification

Root Object

The root object exposes server properties.

Root Object - properties

Querying the root object is simply a matter of using the action `get` without a target. The properties of the server are returned.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{ "action": "get" }
```

Response Example

```
{
  "response": {
    "applicationName": "Smaart v8",
    "applicationVersion": "8.3.0.2",
    "authenticationRequired": false,
    "machineName": "SmaartServer",
    "marshallingTimeout": 2000,
    "supportedSerializationFormats": [
      "clear text",
      "BSON",
      "MessagePack",
      "CBOR",
      "UBJSON"
    ],
    "serializationFormat": "clear text"
  }
}
```

Response Parameters

Name	Type	Description
applicationName	*string*	name of the application server
applicationVersion	*string*	version of the server of the form: `major .minor.patch.platform`
authenticationRequired	*boolean*	indicates whether API access requires a password

machineName	*string*	hostname of the server
marshallingTimeout	*integer*	milliseconds allowed to pass before an asynchronous command is abandoned **default:** `2000`
supportedSerializationFormats	*array*	list of serialization formats the server can encode/decode JSON data to/from
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

Root Object - setting the marshalling timeout

The marshalling timeout dictates the duration an asynchronous call could take before it is abandoned. Setting to a value of zero requires an instantaneous result (unlikely for an asynchronous command), and a value less than zero indicates the response should wait indefinitely for the command to complete (perhaps subject to socket timeout). Commands that effect the user interface require proxying from the API server thread pool to the main thread. The timeout can be set to 1800 milliseconds with the following request:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "properties": [ { "marshallingTimeout": 1800 } ]
}
```

Response Example

```
{
  "response": { "marshallingTimeout": 1800 }
}
```

Root Object - authenticating

If authentication is required, the password can be submitted to the server as follows:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "properties": [ { "password": "<password>" } ]
}
```

Root Object - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis. A separate request is required to utilize serialization on measurement/metric/log streams.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Response Example

```
{
  "response": {
    "serializationFormat": "MessagePack"
  }
}
```

Signal Generator Object

The signal generator object exposes the most commonly used functionality of the server's signal generator.

Signal Generator Object - properties

An action of `get` and a target of `signalGenerator` are used to query the properties of the signal generator object:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "signalGenerator"
}
```

Response Example

```
{
  "response": {
    "type": "Pink Noise",
    "active": false,
    "gain": -42,
    "device": "Smaart I-O",
    "channel1": "Front Left",
    "channel2": "Front Right"
  }
}
```

Response Parameters

Name	Type	Description
type	*string*	type of signal generator **one of:**`"Pink Noise"` or `"File"` or `"Pink Sweep"` or `"Sine"` or `"Dual Sine"`
active	*boolean*	indicates whether the signal generator is active or not
gain	*integer*	level of the signal generator **Range:** `value <= 0`
device	*string*	device to which the generated signal is delivered
channel1	*string*	channel to which the generated signal is delivered
channel2	*string*	channel to which the generated signal is mirrored

Signal Generator Object - enable

The signal generator can be enabled by setting the `active` property to `true`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "signalGenerator",
  "properties": [ { "active": true } ]
}
```

Response Example

```
{ "response": { "active": true } }
```

Signal Generator Object - setting level

The level of the generator can be set to -22 dB with the following request:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "signalGenerator",
  "properties": [ { "gain": -22 } ]
}
```

Response Example

```
{ "response": { "gain": -22 } }
```

Signal Generator Object - changing type

The `type` property can be set to change the output generated. When changing the type of signal, the generator will deactivate. It can be re-activated by setting the `active` property. The following request switches to a `Sine` signal generator:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "signalGenerator",
  "properties": [ { "type": "Sine" } ]
}
```

Response Example

```
{ "response": { "type": "Sine" } }
```

Settings Object

The settings object contains the server's global settings.

Settings Object - properties

The server's global settings are returned with the following request:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "settings"
}
```

Response Example

```
{
  "response": {
    "spectrumSettings": {
      "averaging": "None",
      "banding": "1/3 Octave"
    },
    "transferFunctionSettings": {
      "averaging": "1 Second",
      "magnitudeSmoothing": "None",
      "phaseSmoothing": "None"
    }
  }
}
```

Response Parameters

Name	Type	Description

spectrumSettings:averaging	*string*	type of averaging applied **one of:**"None" or "2 FIFO" or "4 FIFO" or "8 FIFO" or "16 FIFO" or "1 Second" or "2 Seconds" or "3 Seconds" or "4 Seconds" or "5 Seconds" or "6 Seconds" or "7 Seconds" or "8 Seconds" or "9 Seconds" or "10 Seconds" or "Infinite" or "Fast" or "Slow"
spectrumSettings:banding	*string*	type of banding applied **one of:**"None" or "Octave" or "1/3 Octave" or "1/6 Octave" or "1/12 Octave" or "1/24 Octave" or "1/48 Octave"
transferFunctionSettings:averaging	*string*	type of averaging applied **one of:**"None" or "2 FIFO" or "4 FIFO" or "8 FIFO" or "16 FIFO" or "1 Second" or "2 Seconds" or "3 Seconds" or "4 Seconds" or "5 Seconds" or "6 Seconds" or "7 Seconds" or "8 Seconds" or "9 Seconds" or "10 Seconds" or "Infinite" or "Fast" or "Slow"
transf erFunctionSettings:magnitudeSmoothing	*string*	type of smoothing applied **one of:**"None" or "Octave" or "1/3 Octave" or "1/6 Octave" or "1/12 Octave" or "1/24 Octave" or "1/48 Octave"
tr ansferFunctionSettings:phaseSmoothing	*string*	type of smoothing applied **one of:**"None" or "Octave" or "1/3 Octave" or "1/6 Octave" or "1/12 Octave" or "1/24 Octave" or "1/48 Octave"

Settings Object - setting spectrum averaging

The global spectrum averaging can be set to `2 Seconds` by setting the `spectrumSettings.averaging` property:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "settings",
  "properties": [
    {
      "spectrumSettings.averaging": "2 Seconds"
    }
  ]
}
```

Response Example

```
{
  "response": {
    "spectrumSettings": { "averaging": "2 Seconds" }
  }
}
```

Measurements Object

The measurements object is a container revealing all defined measurements without prior knowledge of the window or tab in which they exist.

Measurements Object - properties

When querying the properties of the measurements collection, a tree of windows their tabs, and the measurements for each is returned:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "measurements"
}
```

Response Example


```
{
  "response": {
    "windows": [
      {
        "windowName": "Smaart",
        "active": true,
        "tabs": [
          {
            "tabName": "Default Tab",
            "active": true,
            "spectrumMeasurements": [
              {
                "measurementName": "Front Left",
                "active": true,
                "streamEndpoint": "/api/v3/tabs/Default%20Tab/measureme..."
              },
              {
                "measurementName": "Front Right",
                "active": false
              }
            ],
            "transferFunctionMeasurements": [
              {
                "measurementName": "Mic 1",
                "active": true,
                "streamEndpoint": "/api/v3/tabs/Default%20Tab/measureme...",
                "lirStreamEndpoint": "/api/v3/tabs/Default%20Tab/measur..."
              }
            ]
          },
          {
            "tabName": "Tab A",
            "active": false,
            "spectrumMeasurements": [],
            "transferFunctionMeasurements": []
          }
        ]
      },
      {
        "windowName": "Window 2",
        "active": false,
        "tabs": [
          {
            "tabName": "Tab B",
            "active": true,
            "spectrumMeasurements": [],
            "transferFunctionMeasurements": []
          }
```

```
        ]
      }
    ]
  }
}
```

Response Parameters

Name	Type	Description
windowName	*string*	name of the window
active	*boolean*	indicates whether the window is active or not
tabs/tabName	*string*	name of the tab
tabs/active	*boolean*	indicates whether the tab is active or not
tabs /spectrumMeasurements/measurementName	*string*	measurement name
tabs/spectrumMeasurements/active	*boolean*	indicates whether the measurement is active or not
tab s/spectrumMeasurements/streamEndpoint	*string*	URL-encoded endpoint to which a WebSocket can be opened to stream the measurement's data

Measurements Object - active measurements

When querying for active measurements, the same structure is returned, but only those measurements that are active are included.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "activeMeasurements"
}
```

Response Example


```
{
  "response": {
    "windows": [
      {
        "windowName": "Smaart",
        "active": true,
        "tabs": [
          {
            "tabName": "Default Tab",
            "active": true,
            "spectrumMeasurements": [
              {
                "measurementName": "Front Left",
                "active": true,
                "streamEndpoint": "/api/v3/tabs/Default%20Tab/measureme..."
              }
            ],
            "transferFunctionMeasurements": [
              {
                "measurementName": "Mic 1",
                "active": true,
                "streamEndpoint": "/api/v3/tabs/Default%20Tab/measureme...",
                "lirStreamEndpoint": "/api/v3/tabs/Default%20Tab/measur..."
              }
            ]
          },
          {
            "tabName": "Tab A",
            "active": false,
            "spectrumMeasurements": [],
            "transferFunctionMeasurements": []
          }
        ]
      },
      {
        "windowName": "Window 2",
        "active": false,
        "tabs": [
          {
            "tabName": "Tab B",
            "active": true,
            "spectrumMeasurements": [],
            "transferFunctionMeasurements": []
          }
        ]
      }
    ]
```

```
  }
}
```

Measurements Object - resetting averages

The averages for active measurements can be reset with the following request:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "activeMeasurements",
  "properties": [ { "runningAverage": 0 } ]
}
```

Response Example

```
{
  "response": { "status": "running averages reset" }
}
```

Tabs Object

The `tabs` object controls which tab is active.

Tabs Object - getting the active tab

Querying the `tabs` object yields a list of the active window's tabs:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{ "action": "get", "target": "tabs" }
```

Response Example

```
{
  "response": {
    "activeWindow": "Smaart",
    "activeTab": "Default Tab",
    "tabNames": [ "Default Tab", "Tab 2" ]
  }
}
```

Tabs Object - setting the active tab

The tab named `Tab 2` is brought to the front of its window with the following request:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": "tabs",
  "properties": [ { "activeTab": "Tab 2" } ]
}
```

Response Example

```
{
  "response": {
    "activeWindow": "Smaart",
    "activeTab": "Tab 2"
  }
}
```

Making a tab active will cause the window hosting the tab to become active.

Measurement Object

The `measurement` object contains all sorts of wonderful things about measurements.

Measurement Object - query

The properties of a measurement are requested as follows:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": {
    "tabName": "Default Tab",
    "measurementName": "Front Left"
  }
}
```

Request Parameters

Name	Type	Description
target:tabName	*string*	name of the tab of interest
target:measurementName	*string*	name of the measurement of interest

If the tab name is not specified, the active tab of the active window is assumed.

Measurement Object - live spectrum measurements

The data associated with spectrum measurements includes:

Response Example


```
{
  "response": {
    "windowName": "Smaart",
    "tabName": "Default Tab",
    "measurementName": "Front Left",
    "type": "spectrum",
    "measurementDevice": "Smaart I-O",
    "measurementChannel": "Front Left",
    "measurementChannelIndex": 0,
    "active": true,
    "averaging": "2 Seconds",
    "banding": "1/3 Octave",
    "calibrationOffset": 0,
    "color": {
      "red": 120,
      "green": 0,
      "blue": 215,
      "alpha": 255
    },
    "dataWindow": "Hann",
    "fft": 16384,
    "requiresSignalGenerator": false,
    "sampleRate": 48000,
    "bitDepth": 24,
    "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/Front%20Left"
  }
}
```


Response Parameters

Name	Type	Description
windowName	*string*	name of the host window
tabName	*string*	name of the host tab
measurementName	*string*	name of the measurement of interest
type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
measurementDevice	*string*	device name of the measurement
measurementChannel	*string*	channel name of the measurement

measurementChannelIndex	*number*	index of the measurement channel
active	*boolean*	indicates whether the measurement is active or not
averaging	*string*	type of averaging applied **one of:**`"None"` or `"2 FIFO"` or `"4 FIFO"` or `"8 FIFO"` or `"16 FIFO"` or `"1 Second"` or `"2 Seconds"` or `"3 Seconds"` or `"4 Seconds"` or `"5 Seconds"` or `"6 Seconds"` or `"7 Seconds"` or `"8 Seconds"` or `"9 Seconds"` or `"10 Seconds"` or `"Infinite"` or `"Fast"` or `"Slow"`
banding	*string*	type of banding applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
calibrationOffset	*number*	calibration offset
color:red	*integer*	red component of the color **Range:** `0 <= value <= 255`
color:green	*integer*	green component of the color **Range:** `0 <= value <= 255`
color:blue	*integer*	blue component of the color **Range:** `0 <= value <= 255`
color:alpha	*integer*	alpha component of the color **Range:** `0 <= value <= 255`
dataWindow	*string*	type of windowing **one of:**`"None"` or `"Bartlett"` or `"Blackman"` or `"Blackman-Harris"` or `"Flat Top"` or `"Hamming"` or `"Hann"` or `"Parzen"` or `"Tukey"` or `"Welch"`
fft	*integer*	FFT size **one of:**`32768` or `16384` or `8192` or `4096` or `2048` or `1024` or `512` or `256` or `128`
requiresSignalGenerator	*boolean*	indicates whether the signal generator is required or not

sampleRate	*integer*	number of samples per second
bitDepth	*integer*	number of bits per sample
streamEndpoint	*string*	URL-encoded endpoint to which a connection will yield continuous measurement data

The `streamEndpoint` property is only included for active measurements.

Measurement Object - live transfer function measurements

The data associated with transfer function measurements includes:

Response Example


```
{
  "response": {
    "windowName": "Smaart",
    "tabName": "Default Tab",
    "measurementName": "EQ",
    "type": "transfer function",
    "measurementDevice": "Smaart I-O",
    "measurementChannel": "Front Left",
    "measurementChannelIndex": 0,
    "referenceDevice": "Smaart I-O",
    "referenceChannel": "Front Right",
    "referenceChannelIndex": 1,
    "active": true,
    "averaging": "1 Second",
    "magnitudeSmoothing": "None",
    "phaseSmoothing": "None",
    "color": {
      "red": 0,
      "green": 155,
      "blue": 5,
      "alpha": 255
    },
    "dataWindow": "Hann",
    "fft": "MTW",
    "delay": 3.12,
    "trackingDelay": false,
    "inverted": false,
    "magnitudeAveragingType": "polar",
    "magnitudeThreshold": -70,
    "requiresSignalGenerator": false,
    "sampleRate": 48000,
    "bitDepth": 24,
    "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/EQ",
    "lirStreamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/EQ/lir"
  }
}
```


Response Parameters

Name	Type	Description
windowName	*string*	name of the host window
tabName	*string*	name of the host tab
measurementName	*string*	name of the measurement of interest

type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
measurementDevice	*string*	device name of the measurement
measurementChannel	*string*	channel name of the measurement
measurementChannelIndex	*number*	index of the measurement channel
referenceDevice	*string*	device name of the reference
referenceChannel	*string*	channel name of the reference
referenceChannelIndex	*number*	index of the reference channel
active	*boolean*	indicates whether the measurement is active or not
averaging	*string*	type of averaging applied **one of:**`"None"` or `"2 FIFO"` or `"4 FIFO"` or `"8 FIFO"` or `"16 FIFO"` or `"1 Second"` or `"2 Seconds"` or `"3 Seconds"` or `"4 Seconds"` or `"5 Seconds"` or `"6 Seconds"` or `"7 Seconds"` or `"8 Seconds"` or `"9 Seconds"` or `"10 Seconds"` or `"Infinite"` or `"Fast"` or `"Slow"`
magnitudeSmoothing	*string*	type of smoothing applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
phaseSmoothing	*string*	type of smoothing applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
color:red	*integer*	red component of the color **Range:** `0 <= value <= 255`

color:green	*integer*	green component of the color **Range:** `0 <= value <= 255`
color:blue	*integer*	blue component of the color **Range:** `0 <= value <= 255`
color:alpha	*integer*	alpha component of the color **Range:** `0 <= value <= 255`
dataWindow	*string*	type of windowing **one of:**`"None"` or `"Bartlett"` or `"Blackman"` or `"Blackman-Harris"` or `"Flat Top"` or `"Hamming"` or `"Hann"` or `"Parzen"` or `"Tukey"` or `"Welch"`
fft	*object*	FFT type **one of:**`"MTW"` or `"MTW+"` or `32768` or `16384` or `8192` or `4096` or `2048` or `1024` or `512` or `256` or `128`
delay	*number*	millisecond delay applied to the measurement **Range:** `0 <= value <= 1000`
trackingDelay	*boolean*	indicates whether the measurement is tracking delay or not
inverted	*boolean*	indicates whether magnitude plots are displayed up-side down
magnitudeAveragingType	*string*	type of averaging for magnitude traces **one of:**`"polar"` or `"complex"`
magnitudeThreshold	*number*	threshold applied to magnitude for averaging
requiresSignalGenerator	*boolean*	indicates whether the signal generator is required or not
sampleRate	*integer*	number of samples per second
bitDepth	*integer*	number of bits per sample
streamEndpoint	*string*	URL-encoded endpoint to which a connection will yield continuous measurement data
lirStreamEndpoint	*string*	URL-encoded endpoint to which a connection will yield continuous live impulse response data

The `streamEndpoint` and `lirStreamEndpoint` properties are only included for active measurements.

Measurement Object - spectrum average measurements

The properties included in spectrum average measurements include:

Response Example


```
{
  "response": {
    "windowName": "Smaart",
    "tabName": "Default Tab",
    "measurementName": "average",
    "type": "spectrum average",
    "averagedMeasurementNames": [ "Front Left", "Front Right" ],
    "active": false,
    "activeAveragedMeasurementNames": [],
    "averagedAs": "power",
    "banding": "1/3 Octave",
    "calibrationOffset": 0,
    "color": {
      "red": 120,
      "green": 0,
      "blue": 215,
      "alpha": 255
    },
    "requiresSignalGenerator": false
  }
}
```


Response Parameters

Name	Type	Description
windowName	*string*	name of the host window
tabName	*string*	name of the host tab
measurementName	*string*	name of the measurement of interest
type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
averagedMeasurementNames	*array*	list of measurement names included in the average
active	*boolean*	indicates whether the measurement is active or not

activeAveragedMeasurementNames	*array*	list of measurement included in the average that are active
averagedAs	*string*	average type **one of:**`"power"` or `"dB"`
banding	*string*	type of banding applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
calibrationOffset	*number*	calibration offset
color:red	*integer*	red component of the color **Range:** `0 <= value <= 255`
color:green	*integer*	green component of the color **Range:** `0 <= value <= 255`
color:blue	*integer*	blue component of the color **Range:** `0 <= value <= 255`
color:alpha	*integer*	alpha component of the color **Range:** `0 <= value <= 255`
requiresSignalGenerator	*boolean*	indicates whether the signal generator is required or not

If the measurement is active, a `streamEndpoint` property is also included.

Measurement Object - transfer function average measurements

The properties included in transfer function average measurements:

Response Example

```
{
  "response": {
    "windowName": "Smaart",
    "tabName": "Default Tab",
    "measurementName": "average",
    "type": "transfer function average",
    "averagedMeasurementNames": [ "Mic 1", "Mic 2" ],
    "active": false,
    "activeAveragedMeasurementNames": [],
    "averagedAs": "dB",
    "coherenceWeighted": true,
    "color": {
      "red": 210,
      "green": 0,
      "blue": 85,
      "alpha": 255
    },
    "requiresSignalGenerator": false
  }
}
```

Response Parameters

Name	Type	Description
windowName	*string*	name of the host window
tabName	*string*	name of the host tab
measurementName	*string*	name of the measurement of interest
type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
averagedMeasurementNames	*array*	list of measurement names included in the average
active	*boolean*	indicates whether the measurement is active or not
activeAveragedMeasurementNames	*array*	list of measurement included in the average that are active
averagedAs	*string*	average type **one of:**`"power"` or `"dB"`

coherenceWeighted	*boolean*	indicates whether measurements are weighted by coherence
color:red	*integer*	red component of the color **Range:** `0 <= value <= 255`
color:green	*integer*	green component of the color **Range:** `0 <= value <= 255`
color:blue	*integer*	blue component of the color **Range:** `0 <= value <= 255`
color:alpha	*integer*	alpha component of the color **Range:** `0 <= value <= 255`
requiresSignalGenerator	*boolean*	indicates whether the signal generator is required or not

If the measurement is active, `streamEndpoint` and `lirStreamEndpoint` properties are also included.

Measurement Object - starting a measurement

Starting a measurement is achieved by setting the `active` property to `true`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": { "measurementName": "Front Left" },
  "properties": [ { "active": true } ]
}
```

Request Parameters

Name	Type	Description
target:measurementName	*string*	name of the measurement of interest
properties:active	*boolean*	desired active state (true to start)

Response Example

```
{ "response": { "active": true } }
```

Measurement Object - stopping a measurement

Stopping a measurement is achieved by setting the `active` property to `false`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": { "measurementName": "Front Left" },
  "properties": [ { "active": false } ]
}
```

Request Parameters

Name	Type	Description
target:measurementName	*string*	name of the measurement of interest
properties:active	*boolean*	desired active state (false to stop)

Response Example

```
{ "response": { "active": false } }
```

Measurement Object - starting all spectrum measurements

All of a tab's spectrum measurements can be started by setting the `active` property to `true` for the measurement named `allSpectrumMeasurements`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": {
    "tabName": "Default Tab",
    "measurementName": "allSpectrumMeasurements"
  },
  "properties": [ { "active": true } ]
}
```

Response Example

```
{
  "response": {
    "tabName": "Default Tab",
    "active": true,
    "spectrumMeasurements": [
      {
        "measurementName": "Front Left",
        "active": true,
        "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/..."
      },
      {
        "measurementName": "Front Right",
        "active": true,
        "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/..."
      }
    ]
  }
}
```

The `active` property is returned for each measurement to verify each has started.

All spectrum measurements can be stopped by setting the `active` property to `false`.

All measurements, both spectrum and transfer function measurements, can be started or stopped using the measurement named `allMeasurements`.

Measurement Object - starting all transfer function measurements

All of a tab's transfer function measurements can be started by setting the `active` property to `true` for the measurement named `allTransferFunctionMeasurements`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": {
    "tabName": "Default Tab",
    "measurementName": "allTransferFunctionMeasurements"
  },
  "properties": [ { "active": true } ]
}
```

Response Example

```
{
  "response": {
    "tabName": "Default Tab",
    "active": true,
    "transferFunctionMeasurements": [
      {
        "measurementName": "EQ 1",
        "active": true,
        "trackingDelay": false,
        "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/...",
        "lirStreamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/..."
      },
      {
        "measurementName": "EQ 2",
        "active": true,
        "trackingDelay": true,
        "streamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/...",
        "lirStreamEndpoint": "/api/v3/tabs/Default%20Tab/measurements/..."
      }
    ]
  }
}
```

All transfer function measurements can be stopped by setting the `active` property to `false`.

All measurements, both spectrum and transfer function measurements, can be started or stopped using the measurement named `allMeasurements`.

Measurement Object - setting delay tracking

Enabling (and disabling) delay tracking for a measurement is achieved by setting the `trackingDelay` property to `true` (`false`):

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": { "measurementName": "EQ" },
  "properties": [ { "trackingDelay": true } ]
}
```

Request Parameters

Name	Type	Description

target:measurementName	*string*	name of the measurement of interest
properties:trackingDelay	*boolean*	indicates whether the delay is being tracked for the measurement or not

Response Example

```
{
  "response": { "trackingDelay": true }
}
```

Setting the `measurementName` property to `allTransferFunctionMeasurements` will cause delay tracking to be enabled/disabled for all transfer function measurements.

Measurement Object - setting delay

Setting the delay for a transfer function measurement is achieved by setting, you guessed it, the `delay` property:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": { "measurementName": "Front Left" },
  "properties": [ { "delay": 21.3 } ]
}
```

Request Parameters

Name	Type	Description
target:measurementName	*string*	name of the measurement of interest
properties:delay	*number*	millisecond delay applied to the measurement **Range:** `0 <= value <= 1000`

Response Example

```
{ "response": { "delay": 21.3 } }
```

Measurement Object - finding delay

To find the delay of a transfer function measurement, use `findDelay` as the `action`:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "findDelay",
  "target": { "measurementName": "EQ" },
  "properties": [
    { "automaticallyStart": true },
    { "automaticallyInsert": true },
    { "automaticallyStop": true }
  ]
}
```

Request Parameters

Name	Type	Description
target:measurementName	*string*	name of the measurement of interest
properties:automaticallyStart	*boolean*	optionally indicates whether the measurement should automatically be started or not **default:** `false`
properties:automaticallyInsert	*boolean*	optionally indicates whether the calculated delay should be inserted into the measurement or not **default:** `false`
properties:automaticallyStop	*boolean*	optionally indicates whether the measurement should automatically be stopped after the delay is found or not **default:** `false`

Response Example

```
{ "response": { "delay": 21.3 } }
```

Measurement Object - capturing a trace

The `capture` action is used to capture a trace in a data file:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "capture",
  "target": { "measurementName": "Front Left" }
}
```

Request Parameters

Name	Type	Description
target:measurementName	*string*	name of the measurement of interest

Response Example

```
{
  "response": {
    "measurementName": "Front Left",
    "type": "spectrum",
    "traceFilePath": "/Users/Karl/Documents/Smaart v8/Traces/Spectrum/Ses..."
  }
}
```

Response Parameters

Name	Type	Description
measurementName	*string*	name of the measurement captured
type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
traceFilePath	*string*	fully qualified path of the captured trace on the server

Measurement Object - capturing all traces

The measurement name of `allMeasurements` can be used to capture all of the active measurements of the active tab:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "capture",
  "target": {
    "measurementName": "allMeasurements"
  }
}
```

Response Example

```
{
  "response": {
    "tabName": "Default Tab",
    "traceFiles": [
      {
        "measurementName": "Front Left",
        "type": "spectrum",
        "traceFilePath": "/Users/Karl/Documents/Smaart v8/Traces/Spectru..."
      },
      {
        "measurementName": "Front Right",
        "type": "spectrum",
        "traceFilePath": "/Users/Karl/Documents/Smaart v8/Traces/Spectru..."
      }
    ]
  }
}
```

Response Parameters

Name	Type	Description
tabName	*string*	tab from which the measurements were captured
traceFiles/measurementName	*string*	name of the measurement captured
traceFiles/type	*string*	type of measurement **one of:**`"spectrum"` or `"spectrum average"` or `"transfer function"` or `"trans fer function average"`
traceFiles/traceFilePath	*string*	fully qualified path of the captured trace on the server

Using a measurement name of `allMeasurements` will capture both spectrum and transfer function measurements that are active. The captured measurements can be restricted to spectrum measurements by

using a measurement name of `allSpectrumMeasurements`. The captured measurements can be restricted to transfer function measurements by using a measurement name of `allTransferFunctionMeasurements`.

Data Library Object

The data library object allows access to captured trace files.

Data Library Object - renaming

After a measurement is captured in a trace file, that file can be renamed by setting the `name` property. If a naming conflict occurred, the returned path will indicate the new name.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": {
    "traceFilePath": "<path to captured trace>"
  },
  "properties": [
    {
      "name": "<new captured trace name>"
    }
  ]
}
```

Request Parameters

Name	Type	Description
target:traceFilePath	*string*	fully qualified path of the captured trace file on the server
properties/name	*string*	desired name of the captured trace (file)

Data Library Object - relocating

A captured trace file can be relocated or moved by setting it's `path` property. If a naming conflict occurred, the returned path will indicate the new location.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "set",
  "target": { "traceFilePath": "<source path>" },
  "properties": [ { "path": "<destination path>" } ]
}
```

Data Library Object - acquiring trace data

The contents of a captured trace file can be acquired by querying for the base 64 encoding. The content can be decoded and stored locally as an SRF or TRF file.

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": { "traceFilePath": "<source path>" },
  "properties": [ { "base64Encoding": "RFC4648" } ]
}
```

Request Parameters

Name	Type	Description
target:traceFilePath	*string*	fully qualified path of the captured trace file on the server

Response Example

```
{
  "response": {
    "traceFilePath": "/Users/Karl/Documents/Smaart v8/Traces...",
    "base64Encoding": "131768.JEzPKIU..."
  }
}
```

Response Parameters

Name	Type	Description
traceFilePath	*string*	fully qualified path of the captured trace file on the server

base64Encoding	*string*	RFC4648 base 64 encoding of the captured trace file on the server

Spectrum Measurement Stream

A stream of spectrum measurement data can be initiated by opening a connection to the `streamEndpoint` property of the measurement when the measurement is active.

Spectrum Measurement Stream - connecting

When connected to a stream endpoint for an active spectrum measurement, the following object will be received as data is calculated:

```
WebSocket Endpoint /api/v3/tabs/<tab name>/measurements/<measurement name>/
```

Response Example

```
{
  "timestamp": "2018-02-09:T12:34:39.125-5:00",
  "description": "frequency vs magnitude",
  "banding": "1/3 Octave",
  "dB FS Peak": -26.22,
  "data": [
    [ 12.59, -130.97 ],
    [ 15.85, -126.04 ]
  ]
}
```

Response Parameters

Name	Type	Description
timestamp	*string*	time in ISO8601 format
description	*string*	indicates what data is reported
banding	*string*	type of banding applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
dB FS Peak	*number*	full scale peak signal level
data	*array*	a list of arrays of tabulated measurement data

Spectrum Measurement Stream - adjusting banding

Octave banding (which will reduce payload size) can be specified with the following request:

Request Example

```
{
  "action": "set",
  "properties": [ { "banding": "Octave" } ]
}
```

Request Parameters

Name	Type	Description
banding	*string*	type of banding applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`

Spectrum Measurement Stream - adjusting stream update frequency

The frequency of the callbacks on the stream can be throttled to only deliver as needed. Measurement streams start at twenty-three frames per second, which is the maximum. This can be adjusted by setting the `targetFPS` property once the stream is open. The following request transitions the frame rate to two frames per second:

Request Example

```
{
  "action": "set",
  "properties": [ { "targetFPS": 2 } ]
}
```

Request Parameters

Name	Type	Description
targetFPS	*integer*	target frames per second **Range:** `value <= 23`

Spectrum Measurement Stream - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis.

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Request Parameters

Name	Type	Description
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

Transfer Function Measurement Stream

A stream of transfer function measurement data can be initiated by opening a connection to the `streamEndpoint` property of the measurement when the measurement is active.

Transfer Function Measurement Stream - connecting

When connected to a stream endpoint for an active transfer function measurement, the following object will be recieved as data is calculated:

```
WebSocket Endpoint /api/v3/tabs/<tab name>/measurements/<measurement name>
```

Response Example

```
{
  "timestamp": "2018-02-09:T12:34:39.123-5:00",
  "description": "frequency vs magnitude phase coherence",
  "magnitudeSmoothing": "1/6 Octave",
  "phaseSmoothing": "1/3 Octave",
  "dB FS Peak (Measurement)": -25.02,
  "dB FS Peak (Reference)": -24.98,
  "data": [
    [ 10.74, 6.12, 99.98, 0.58 ],
    [ 11.72, 7.16, 65.22, 0.49 ]
  ]
}
```

Response Parameters

Name	Type	Description
timestamp	*string*	time in ISO8601 format
description	*string*	indicates what data is reported
magnitudeSmoothing	*string*	type of smoothing applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
phaseSmoothing	*string*	type of smoothing applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`
dB FS Peak (Measurement)	*number*	peak level of measurement channel
dB FS Peak (Reference)	*number*	peak level of reference channel
data	*array*	a list of arrays of tabulated measurement data

A value of 999999.0 for magnitude, phase or coherence indicates that the data is invalid and the sample should not be considered. This can occur when the magnitude threshold is not reached or the coherence is below the blanking threshold.

Transfer Function Measurement Stream - adjusting smoothing

The magnitude smoothing is changed to `Octave` with the following request:

Request Example

```
{
  "action": "set",
  "properties": [ { "magnitudeSmoothing": "Octave" } ]
}
```

Request Parameters

Name	Type	Description

magnitudeSmoothing	*string*	type of smoothing applied **one of:**`"None"` or `"Octave"` or `"1/3 Octave"` or `"1/6 Octave"` or `"1/12 Octave"` or `"1/24 Octave"` or `"1/48 Octave"`

Likewise with the `phaseSmoothing` property.

Transfer Function Measurement Stream - selecting data to report

The columns reported in the `data` sub arrays can be included or excluded with the following request:

Request Example

```
{
  "action": "set",
  "properties": [
    { "includeMagnitude": true },
    { "includePhase": false },
    { "includeCoherence": false }
  ]
}
```

Request Parameters

Name	Type	Description
includeMagnitude	*boolean*	optional parameter to include (`true`) or exclude (`false`) the magnitude column
includePhase	*boolean*	optional parameter to include (`true`) or exclude (`false`) the phase column
includeCoherence	*boolean*	optional parameter to include (`true`) or exclude (`false`) the coherence column

If all columns are excluded, only the timestamp will be reported. The magnitude column always precedes the phase column which always precedes the coherence column.

Transfer Function Measurement Stream - adjusting stream update frequency

The frequency of the callbacks on the stream can be throttled to only deliver as needed. Measurement streams start at twenty-three frames per second, which is the maximum. This can be adjusted by setting the `targetFPS` property once the stream is open. The following request transitions the frame rate to two frames per second:

Request Example

```
{
  "action": "set",
  "properties": [ { "targetFPS": 2 } ]
}
```

Request Parameters

Name	Type	Description
targetFPS	*integer*	target frames per second **Range:** `value <= 23`

Transfer Function Measurement Stream - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis.

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Request Parameters

Name	Type	Description
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

Transfer Function Measurement LIR Stream

A stream of a transfer function measurement's live impulse response data can be initiated by opening a connection to the `lirStreamEndpoint` property of the measurement when the measurement is active.

Transfer Function Measurement LIR Stream - connecting

When connected to a stream endpoint for an active transfer function measurement's live impulse response, the following object will be recieved as data is calculated:

```
WebSocket Endpoint /api/v3/tabs/{tab}/measurements/{measurement}/lir
```

Response Example

```
{
  "timestamp": "2018-02-09:T12:34:39.123-5:00",
  "delay": 3.27,
  "type": "impulse response",
  "data": [ [ 0.22, 0.01 ] ]
}
```

Response Parameters

Name	Type	Description
timestamp	*string*	time in ISO8601 format
delay	*number*	millisecond delay applied to the measurement **Range:** `0 <= value <= 1000`
type	*string*	indicates the type of data reported in an LIR data stream **one of:**`"impulse response"` or `"envelope time curve"`
data	*array*	a list of arrays of tabulated measurement data

Transfer Function Measurement LIR Stream - LIN

The LIN data can be selected with the following request:

Request Example

```
{
  "action": "set",
  "properties": [
    { "envelopeTimeCurve": false },
    { "logScale": false }
  ]
}
```

Transfer Function Measurement LIR Stream - LOG

The LOG data can be selected with the following request:

Request Example

```
{
  "action": "set",
  "properties": [
    { "envelopeTimeCurve": false },
    { "logScale": true }
  ]
}
```

Transfer Function Measurement LIR Stream - ETC

The ETC data can be selected with the following request:

Request Example

```
{
  "action": "set",
  "properties": [ { "envelopeTimeCurve": true } ]
}
```

Transfer Function Measurement LIR Stream - trimming

The live impulse response data can be enormous, so a window of time around the delay is used to trim less than interesting information. The start and end times of the window are relative to the delay and are set as follows:

Request Example

```
{
  "action": "set",
  "properties": [
    { "startTime": -15 },
    { "endTime": 15 }
  ]
}
```

Request Parameters

Name	Type	Description
properties:startTime	*number*	time, in milliseconds, relative to the delay, before which, data will not be reported **Range:** `value <= 0.0`

properties:endTime	*number*	time, in milliseconds, relative to the delay, after which, data will not be reported **Range:** `0.0 < value`

Transfer Function Measurement LIR Stream - data sequencing

The returned measurement data will be sequenced according to the calculation output (`relative`) by default. In this sequence, the data starts at the delay time, or zero if the delay is not set. The x-values then increase until the end of the data window, as set above. They then continue from the start time of the data window back to the delay time, or zero.

In the `natural` sequence, the data starts at the start of the data window and continues to the end of the data window. The following request will change the sequence of measurement data:

Request Example

```
{
  "action": "set",
  "properties": [ { "sequence": "natural" } ]
}
```

Request Parameters

Name	Type	Description
properties:sequence	*string*	**one of:**`"natural"` or `"relative"`

Transfer Function Measurement LIR Stream - adjusting stream update frequency

The frequency of the callbacks on the stream can be throttled to only deliver as needed. Measurement streams start at twenty-three frames per second, which is the maximum. This can be adjusted by setting the `targetFPS` property once the stream is open. The following request transitions the frame rate to two frames per second:

Request Example

```
{
  "action": "set",
  "properties": [ { "targetFPS": 2 } ]
}
```

Request Parameters

Name	Type	Description
targetFPS	*integer*	target frames per second **Range:** `value <= 23`

Transfer Function Measurement LIR Stream - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis.

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Request Parameters

Name	Type	Description
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

ActiveCalibratedInputs Object

A collection of logging inputs are exposed via the active calibrated inputs object.

ActiveCalibratedInputs Object - querying

A list of devices with inputs that are calibrated and actively logging is returned with the following query:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "activeCalibratedInputs"
}
```

Response Example


```
{
  "response": {
    "devices": [
      {
        "deviceName": "Smaart I-O",
        "activeCalibratedChannels": [
          {
            "channelIndex": 0,
            "channelName": "Front Left",
            "streamEndpoint": "/api/v3/devices/...",
            "logEndpointPrefix": "/api/v3/logs/..."
          },
          {
            "channelIndex": 1,
            "channelName": "Front Right",
            "streamEndpoint": "/api/v3/devices/...",
            "logEndpointPrefix": "/api/v3/logs/..."
          }
        ]
      },
      {
        "deviceName": "OCTA-CAPTURE",
        "activeCalibratedChannels": [
          {
            "alarms": [
              {
                "level": 110,
                "metric": "SPL A Slow"
              }
            ],
            "channelIndex": 3,
            "channelName": "Mic 1",
            "streamEndpoint": "/api/v3/devices/...",
            "logEndpointPrefix": "/api/v3/logs/..."
          }
        ]
      }
    ],
    "metrics": [ "FS Peak", "Peak C", "SPL Fast" ]
  }
}
```


Response Parameters

Name	Type	Description
deviceName	*string*	name of a device with active calibrated inputs

activeCalibratedChannels/alarms/level	*number*	level at which the alarm will trigger for the given input
a ctiveCalibratedChannels/alarms/metric	*string*	name of the metric the value of which will trigger the alarm
activeCalibratedChannels/channelIndex	*integer*	index of the logging channel **Range:** `0 <= value`
activeCalibratedChannels/channelName	*string*	name of the logging channel
ac tiveCalibratedChannels/streamEndpoint	*string*	URL-encoded endpoint to which a connection will yield instantaneous metric values
activ eCalibratedChannels/logEndpointPrefix	*string*	URL-encoded prefix to which a URL-encoded metric can be appended to form an endpoint a connection to which will yield logged data from the input for the given metric
metrics	*array*	list of available metric names

SPL Metric Stream

A stream of instantaneous metric values for an active calibrated input can be initiated by opening a connection to the `streamEndpoint` property of the `activeCalibratedChannel`.

SPL Metric Stream - connecting

When connected to an instantaneous stream endpoint for an active calibrated input, the following object will be recieved:

```
WebSocket Endpoint /api/v3/devices/{device_name}/channels/{channel_name}
```

Response Example

```
{
  "timestamp": "2018-04-02:T16:20:00.000-5:00",
  "deviceName": "Smaart I-O",
  "channelName": "Front Left",
  "metrics": [
    { "FS Peak": -54.41 },
    { "Peak C": 80.77 },
    { "SPL Fast": 74.78 },
    { "SPL A Fast": 69.86 },
    { "SPL C Fast": 73.21 },
    { "SPL Slow": 75.98 },
    { "SPL A Slow": 75.86 },
    { "SPL C Slow": 76.54 },
    { "Leq 1": 73.2 },
    { "LAeq 1": 74.9 },
    { "LCeq 1": 74.1 },
    { "Leq 10": 74.2 },
    { "LAeq 10": 74.9 },
    { "LCeq 10": 74.1 }
  ]
}
```

Response Parameters

Name	Type	Description
timestamp	*string*	time in ISO8601 format
deviceName	*string*	name of the device from which the data is reported
channelName	*string*	name of the channel from which the data is reported
metrics	*array*	list of metric names and values

The last three metrics (`Leq 10`, `LAeq 10`, & `LCeq 10` are the default user Leq metrics. If these have been modified, appended to, or removed, the list of metrics will reflect those changes.

If the device is a 10EaZy microphone, the configured 10EaZy Leq metric will be included.

If an alarm is configured for a metric of the input (as reported in the `alarms` property of the `activeCalibratedChannels` item of the `activeCalibratedInputs` query), and the value of that metric surpasses the level of the configured alarm, the object with that metric's value will contain a property named `violation` with a value of `true`.

SPL Metric Stream - adjusting stream update frequency

The frequency of the callbacks on the stream can be throttled to only deliver as needed. SPL metric streams start at eight frames per second, which is the maximum. This can be adjusted by setting the `targetFPS` property once the stream is open. The following request transitions the frame rate to two frames per second:

Request Example

```
{
  "action": "set",
  "properties": [ { "targetFPS": 2 } ]
}
```

Request Parameters

Name	Type	Description
targetFPS	*integer*	target frames per second **Range:** `value <= 8`

SPL Metric Stream - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis.

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Request Parameters

Name	Type	Description
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

Log Metric Stream

A stream containing a logged metric value for an active calibrated input can be initiated by opening a connection to the `logEndpointPrefix` property of the `activeCalibratedChannel` to which a metric name has been appended.

Log Metric Stream - connecting

When connected to a logging stream endpoint for an active calibrated input, the following object will be received as data is logged:

```
WebSocket Endpoint /api/v3/logs/{device_name}/{channel_name}/{metric_name}
```

Response Example

```
{
  "deviceName": "Smaart I-O",
  "channelName": "Front Left",
  "metricName": "SPL A Slow",
  "loggedData": [
    {
      "timestamp": "2018-04-02:T16:20:00.000-5:00",
      "value": 57.16
    }
  ]
}
```

Response Parameters

Name	Type	Description
deviceName	*string*	name of the device from which the data is reported
channelName	*string*	name of the channel from which the data is reported
metricName	*string*	name of the metric whose value is reported
loggedData/timestamp	*string*	time in ISO8601 format
loggedData/value	*number*	value of the metric at the given time

When connecting to a stream which already contains logged data, the `loggedData` list will contain multiple items.

If an alarm is configured for the reported metric of the input (as reported in the `alarms` property of the `activeCalibratedChannels` item of the `activeCalibratedInputs` query), and the value of the metric surpasses the level of the configured alarm, the `loggedData` object will contain a property named `violation` with a value of `true`.

If an overload occurs on the input, the `loggedData` object will contain a property named `overload` with a value of `true`.

The logged data stream pushes data as it is logged. It will not respond to an attempt to set the `targetFPS` property.

Log Metric Stream - setting the serialization format

The server can decode the JSON requests it receives as well as encode the responses. When setting the serialization scheme, the incoming messages as well as outgoing responses are both expected to be in the specified format. Setting the serialization format is done on a per-connection basis.

Request Example

```
{
  "action": "set",
  "properties": [
    {
      "serializationFormat": "MessagePack"
    }
  ]
}
```

Request Parameters

Name	Type	Description
serializationFormat	*string*	format of serialization applied to JSON data **one of:**`"clear text"` or `"BSON"` or `"MessagePack"` or `"CBOR"` or `"UBJSON"`

Command Handler

The Smaart API Server exposes a command handler which will invoke commands associated with given keypresses.

Command Handler - issuing commands

Use the following request to issue a command encoded with the given keypress:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "issueCommand",
  "properties": [ { "keypress": "alt + G" } ]
}
```

Request Parameters

Name	Type	Description
action	*string*	must be `issueCommand`
keypress	*string*	keypress encoding of the desired command

Response Example

```
{ "status": "Measurement Config..." }
```

Response Parameters

Name	Type	Description
status	*string*	description of the command which was issued

Command Handler - querying supported commands

Use the following request to retrieve a list of supported commands:

```
WebSocket Endpoint /api/v3/
```

Request Example

```
{
  "action": "get",
  "target": "commands"
}
```

Response Example

```
{
  "response": {
    "commands": [
      {
        "description": "Cycle Skin",
        "keypresses": [ "ctrl + shift + X" ]
      },
      {
        "description": "I-O Config...",
        "keypresses": [ "option + A" ]
      }
    ]
  }
}
```

Response Parameters

Name	Type	Description
commands/description	*string*	description of the command
commands/keypresses	*array*	list of keypresses that will trigger the command

Appendices

Smaart Product API Implementation

The following is a table conveying the types of queries supported by each of the Smaart products.

API query	Smaart	Smaart Di	Smaart SPL
supported version	✔	✔	✔
server properties	✔	✔	✔
signal generator	✔	✔	
measurement settings	✔	partial	
measurement collection	✔	✔	
tabs	✔	✔	
individual measurement	✔	✔	
Data Library	✔	✔	
measurement streams	✔	✔	
calibrated inputs	✔		✔
metric streams	✔		✔
log streams	✔		✔

Legacy Command Equivalent Requests

The following is a list of legacy commands and the replacement requests for each in the WebSocket API.

`sapiCmdEcho`

There is no replacement for this legacy command.

`sapiCmdGetAPIVersions`

When connected to a server that implements version three of the API, sending a request to the root of the server (`ws://<server address>:<port>`) will yield a list of supported API versions. See also: Supported Versions

`sapiCmdGetSmaartVersion`

A query of the root object without a target will return the version of the Smaart API server in the `applicationVersion` property. See also: Root

`sapiCmdIsAuthenticationRequired`

A query of the root object without a target will return the value of interest in the `authenticationRequired` property. See also: Root

sapiCmdAuthenticate

To authenticate, a command (action = set) is sent without a target, specifying the `password` property. See also: Root - authenticating

sapiCmdGetMachineName

A query of the root object without a target will return the host name of the API server in the `machineName` property. See also: Root

sapiCmdWindowGetNames

A query of the measurements object will yield a list of windows. See also: Measurements

sapiCmdWindowGetCurrent

A query of the measurements object will yield a list of windows with the active state of each in the `windows/active` property. See also: Measurements

A query of the tabs object will yield the active window's name, the name of its active tab, as well as a list of all of its tabs. See also: Tabs - getting the active tab

sapiCmdWindowSetCurrent

The following command will make the `Smaart` window active:

```
{
    "action": "set",
    "target": { "windowName": "Smaart" },
    "properties": [ { "active": true } ]
}
```

sapiCmdGroupGetNames

A query of the tabs object will produce a list of tabs for the active window in the `tabNames` property. See also: Tabs - getting the active tab

sapiCmdGroupGetStatus

Most tab properties are acquired by querying the measurements object. See also: Measurements

sapiCmdGroupSetCurrent

Setting the current tab is achieved by sending a request to the tabs object. See also: Tabs - setting the active tab

sapiCmdGroupGetCurrent

A query of the tabs object will yield the name of the active tab in the active window in the `activeTab` property. See also: Tabs - getting the active tab

`sapiCmdGroupStartAll` & `sapiCmdGroupStopAll`

A request of the measurement object with the `target:measurementName` property set to `allMeasurements` and the `properties/active` property set to `true` (`false`) will start (stop) all a tab's measurements. Specifying `allSpectrumMeasurements` or `allTransferFunctionMeasurements` will start (stop) only those types of measurements. See also: Measurement - starting all spectrum measurements

`sapiCmdGroupTrackAllOn` & `sapiCmdGroupTrackAllOff`

A request of the measurement object with the `target:measurementName` property set to `allTransferFunctionMeasurments` and the `properties/trackingDelay` property set to `true` (`false`) will enable (disable) delay tracking for all the tab's transfer function measurements. See also: Measurement - setting delay tracking

`sapiCmdMeasStop`

A request of the measurement object with the `properties/active` property set to `false` will stop the specified measurement. See also: Measurement - stopping a measurement

`sapiCmdMeasStart`

A request of the measurement object with the `properties/active` property set to `true` will start the specified measurement. See also: Measurement - starting a measurement

`sapiCmdMeasGetStatus`

A request of the measurement object will yield most of the desired properties. See also: Measurement

`sapiCmdMeasGetData`

Live measurement data is streamed from a separate endpoint of the form:

```
/api/v3/tabs/Default%20Tab/measurements/Front%20Left
```

which is returned in the measurement's `streamEndpoint` property. See also: Spectrum Measurement and Transfer Function Measurement

`sapiCmdMeasRequiresSignalGenerator`

When querying a measurement object, the `requiresSignalGenerator` property indicates whether the measurement is dependent upon the signal generator. See also: Measurement

`sapiCmdAvgSet`

See Settings - setting spectrum averaging

`sapiCmdAvgReset`

See Measurements - resetting averages

`sapiCmdAvgGetSpectrum` *&* `sapiCmdAvgGetResponse`

A query of the settings object returns the global spectrum averaging in `spectrumSettings.averaging` and the global transfer function averaging in `transferFunctionSettings.averaging`. See also: Settings

`sapiCmdGetMagnitudeSmoothing` *&* `sapiCmdGetPhaseSmoothing`

A query of the settings object returns the magnitude smoothing in `transferFunctionSettings.magnitudeSmoothing` and the phase smoothing in `transferFunctionSettings.phaseSmoothing`. See also: Settings

`sapiCmdDelaySet`

See Measurement - setting delay

`sapiCmdDelayFind`

See Measurement - finding delay

`sapiCmdDelayTrack`

See Measurement - setting delay tracking

`sapiCmdGeneratorStop` *&* `sapiCmdGeneratorStart`

See Signal Generator - enable

`sapiCmdGeneratorSetLevel`

See Signal Generator - setting level

`sapiCmdGeneratorGetStatus`

All legacy properties are reported in a query of the signal generator object. See also: Signal Generator

`sapiCmdTraceCapture`

See Measurement - capturing a trace

`sapiCmdTraceRename`

See Data Library - renaming

`sapiCmdTraceRelocate`

See Data Library - relocating

History

This appendix contains a list of additions and changes made to the Smaart API.

Smaart 8.5.1

- The `sequence` option was exposed for live impulse response streams. A `natural` option was added which sequences the data in a more intuitive form.
- The live impulse response stream was corrected to center the returned data on the delay.

Smaart 8.5

- Trace file data is now encoded using RFC4648 (base 64).
- In the legacy API (raw sockets), when smoothing is disabled, a return value of -1 was replaced with the documented value 1 or `sapiSmoothNone`.
- The keypress command handler was added.

Smaart 8.4

- The property `applicationName` was added to the response when querying for server properties.
- Alarm details were added to the response to the `activeCalibratedInputs` query.
- Streams of logging data were exposed, per metric.
- The log data stream endpoint prefix was added to the response to the `activeCalibratedInputs` query.
- A list of available metrics was added to the response of the `activeCalibratedInputs` query.
- The property `violation` with a value of `true` was added to the SPL & Log Metric Streams to a metric item whose value exceed the limit of a configured alarm.
- The property `overload` with a value of `true` was added to the Log Metric Stream if the input was overloaded.
- Serialization of the incoming and outgoing JSON data was added. The Root Object now has the `supportedSerializationFormats` property to reveal the supported serialization schemes as well as the `serializationFormat` property to indicate which scheme is in use. The streams can be serialized independently from the main connection.

Smaart 8.3

Smaart 8.3 was the first version to expose API v3 via the WebSocket protocol.