RTSA HTTP Stream Server Endpoints

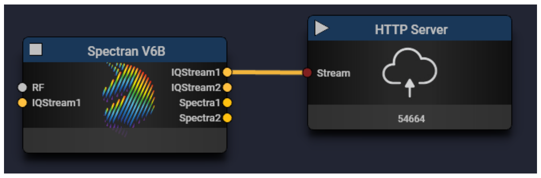
Features and Purpose

The http stream server (and client) block are used to stream measurement and detection data from and to the RTSA suite. It is also used to control and monitor the streaming. The protocol uses http as the underlying transport protocol and the REST paradigm for the API.

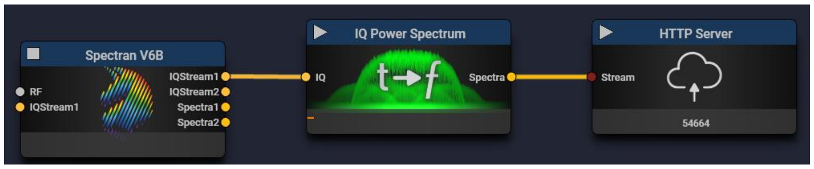
Block Graph

An HTTP server block in the block graph of the RTSA suite provides the access. More than one HTTP server blocks may be present in one graph, but will have to use different ports for their listening socket.

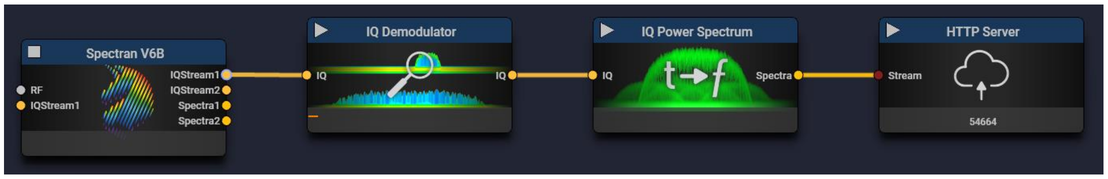
The data streamed will depend on the graph used in the RTSA suite.



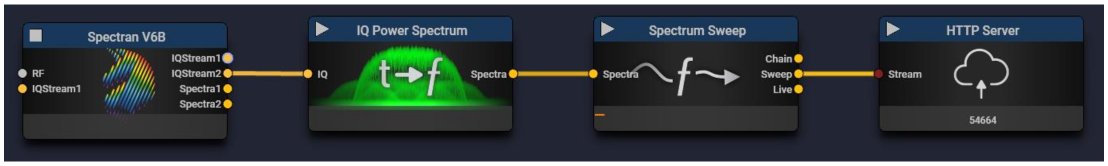
A power spectrum block provides flexible spectrum data.



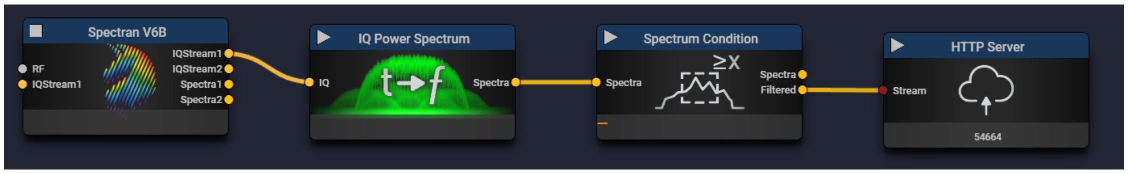
Including a demodulation block allows fine grained control of the spectrum range and sample rate down to the sub hertz frequencies.



A sweep block on the other hand extends the frequency range beyond the realtime span of the capture device.



Reducing the number of spectra by a condition block:



Authorization

The server supports two types of HTTP authorization, “Basic” with username and password and “RToken” with the user specific token, returned by the “user” endpoint for the current user itself.

Stream Format

The stream server supports three main data types for streaming data – the RTSA file format, JSON and a combined JSON and binary format. External applications will most likely use JSON or raw. The stream is split into packets comprising meta data and an array of samples.

Not all fields are present for all media types. The start and end time as well as the payload type are always present.

startTime	Start time of the packet in seconds since the start of the unix epoch.
endTime	End time of the packet in seconds since the start of the unix epoch.
unit	Unit of the sample values
payload	Payload type of the packet
minPower	Minimum power in dBm
maxPower	Maximum power in dBm
startFrequency	Start of a frequency range
endFrequency	End of a frequency range
sampleDepth	Number of sample sets per sample, e.g. bins in a histogram
sampleSize	Sample size, e.g. individual frequency bins in a spectrum or channels in an audio stream
samples	Array of actual sample data
antenna	Antenna specification
categories	Category specification
scale	Scale factor used for integer data when using a 16bit mixed JSON raw format

Spectrum Data

Spectrum data packets will look like this:

```
{
    "startTime" : 1501163970.1396854,
    "endTime"   : 1501163970.140799,

    "unit"     : "dbm"
    "payload"  : "spectra",

    "startFrequency" : 2400250000,
    "endFrequency"   : 2487750000,

    "minPower" : -95,
    "maxPower" : 5,

    "antenna" : {
        "name"        : "Block_IsoLOG_0"
        "latitude"    : 50.13646697998047,
        "longitude"   : 6.320250034332275,
        "azimuth"     : -2.748893976211548,
        "declination" : 0,
    },

    "sampleDepth" : 1,
    "sampleSize"  : 448,

    "samples" : [
        [ -90.05, -90.05, ... , -81.01 ],
        ...
        [ -81.65, -78.05, ... , -90.01 ]
    ],
}
```

IQ Data

IQ Samples are transmitted as a flat array of alternating I and Q values.

```
{
…
    "payload" : "iq",
    "unit"    : "generic"

    "minPower" : -2,
    "maxPower" : 2,

    "sampleDepth" : 1,
    "sampleSize"  : 2,

    "samples": [
         5.12e-05,  0.00132,
         0.000885,  0.00124,
         0.000566,  0.000654,
        -0.000615,  2.35e-05,
         0.00042,  -0.000276,
        -0.000723, -0.000343,
        -0.000672,  0.000195,
         0.000843, -0.000228,
         ...
         ]
}
```

Data that is captured from a source that is not calibrated will have a unit type of generic. The used value range will be given by the min and max power values.

Histogram Data

Histogram data transfers percentages of bin usage. The sample size is like the spectrum or category data, but the sample depth is used to separate the bins. The sample data is a 2D array with the dimensions sample size and sample depth packed into a flat 1D JS array.

```
{
    "startTime" : 1506933004.0587604,
    "endTime"   : 1506933004.0911448,

    "payload" : "histogram",
    "unit"    : "percentage"

    "startFrequency" : 2402250128,
    "endFrequency"   : 2489750128,

    "maxPower" : 5,
    "minPower" : -165,

    "sampleDepth" : 256,
    "sampleSize"  : 896,

    "samples": [
        [ 0.074, 0.0787, ... 0.0893 ]
    ],
}
```

Channel power or other category data

The samples in a category ordered packet will have one measurement per category. The categories are named and may cover an optional frequency range.

name	Name of the category item
startFrequency	Start of a frequency range
endFrequency	End of a frequency range

```
{
…
    "categories" : [
        {
            "name" : "Wifi Channel 1",

            "startFrequency" : 2401000000
            "endFrequency"   : 2423000000,
        },
        ...
    ],
…
}
```

Antenna Data

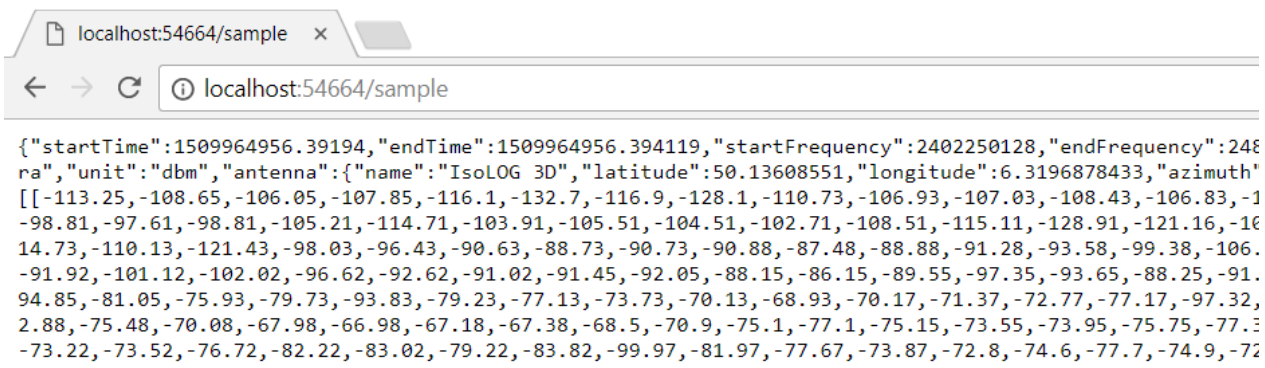
Data that was captured using an antenna with e.g. location or directional information will have an antenna specification field in the packet.

name	Name of the antenna
latitude	Latitude of the antenna
longitude	Longitude of the antenna
azimuth	Azimuth of a directional antenna
declination	Declination of a directional antenna

Data Endpoints

Single Samples

Polling single samples from the server block input is performed with the "/sample" endpoint. This can easily be tried using a standard web browser http://localhost:54664/sample :



```
{"startTime":1509964956.39194,"endTime":1509964956.394119,"startFrequency":2402250128,"endFrequency":248
ra","unit":"dbm","antenna":{"name":"IsoLOG 3D","latitude":50.13608551,"longitude":6.3196878433,"azimuth"
[[-113.25,-108.65,-106.05,-107.85,-116.1,-132.7,-116.9,-128.1,-110.73,-106.93,-107.03,-108.43,-106.83,-1
-98.81,-97.61,-98.81,-105.21,-114.71,-103.91,-105.51,-104.51,-102.71,-108.51,-115.11,-128.91,-121.16,-1
14.73,-110.13,-121.43,-98.03,-96.43,-90.63,-88.73,-90.73,-90.88,-87.48,-88.88,-91.28,-93.58,-99.38,-106.
-91.92,-101.12,-102.02,-96.62,-92.62,-91.02,-91.45,-92.05,-88.15,-86.15,-89.55,-97.35,-93.65,-88.25,-91.
94.85,-81.05,-75.93,-79.73,-93.83,-79.23,-77.13,-73.73,-70.13,-68.93,-70.17,-71.37,-72.77,-77.17,-97.32,
2.88,-75.48,-70.08,-67.98,-66.98,-67.18,-67.38,-68.5,-70.9,-75.1,-77.1,-75.15,-73.55,-73.95,-75.75,-77.
-73.22,-73.52,-76.72,-82.22,-83.02,-79.22,-83.82,-99.97,-81.97,-77.67,-73.87,-72.8,-74.6,-77.7,-74.9,-7
```

The result will contain one or no samples.

Receiving multiple samples can be achieved using the "/samples" endpoint. The number of samples can be controlled with the "limit" argument e.g. http://localhost:54664/samples?limit=1000 will try to capture 1000 samples or spectra. The result will be an array of sample objects.

The "input" argument may be used to select a different input than the "main".

Stream Data

Another option is the use of streaming with a chunked transfer setting using the "/stream" endpoint. The data is transmitted as line limited JSON. Each sample packet is encoded as individual JSON data separated by a line feed (ASCII 10) and a record separator (ASCII 30) character.

The stream can be limited to a maximum number of samples using the limit argument:
http://localhost:54664/stream?format=json

Additional optional arguments are "rate_reduction=n" to reduce the number of samples transmitted by a factor of n. Automatic rate adaption can be disabled by using "rate_adaption=0".

The "input" argument may be used to select a different input than the "main".

Alternative Inputs

Some connectors provide more than one stream. The "input" argument of the sample and stream endpoint is used to select this stream. The "inputs" endpoint returns an array of available inputs.

```
{"inputs":["main","{3b459e11-74e1-4b82-88ad-28459dfe2fe1}"]}
```

New inputs can be created based on existing inputs with a post request to the inputs endpoint. The argument is a JSON document with the following fields:

Input	Name of the original input
type	Type of processing to apply

The result is a JSON document that provides the name of the new input. Available types are:

average	Average of a series of samples
maxhold	Maximum of a series of samples
minhold	Minimum of a series of samples
maxfall	Falling maximum of a series of samples
histogram	Histogram of samples
waterfall	Time compressed samples
	Original samples

Each input does also provide a set of configuration parameters. The input endpoint together with the input parameter can be used to query the current settings (see configuration data).

Raw Data Format

High data rates of e.g. raw IQ data cannot be achieved using JSON formatted arrays (at least not in a compute efficient way). The combined raw format transmits alternating pairs of JSON meta data and binary sample data as 16bit signed integers, 16bit floats or 32bit floats. The binary part starts behind the record separator character. The "samples" element of the JSON meta data contains the number of samples (e.g. spectra or IQ pairs). This format is not available for e.g. structured data.

The float data is requested using the "float16" or "float32" format and the integer data using the "int16" format. A scale value is provided in the JSON metadata section to convert from the integer format to float values. An additional scale parameter can be used to scale the integer data into a meaningful numeric range: http://localhost:54664/stream?format=int16&scale=1000000

The RTSA HTTP server block will start dropping data when the outbound TCP buffer exceeds 8 Mbytes. A loss of data can be determined by comparing the timestamps of two adjacent data packets.

The float16 format follows the IEEE 754-2008 specification for half precision with five exponent and ten mantissa bits.

Control Endpoint

Depending on its functionality each configuration block of the Aaronia-RTSA-Suite accepts control commands for example the **start/stop** command. These commands are not addressed to a specific RTSA block and will be processed from all RTSA blocks in the block graph configuration.

Following **control** HTTP PUT request(http://localhost:54664/control) will start all streaming related blocks at the remote configuration site. To stop the streaming the value of **start** needs to be set to false:

```
{
    "start" : true,
    "type"  : "streaming"
}\n'
```

Sending a put request is problematic using a web browser, but can be done using e.g. “curl” as a command line tool.

```
curl -X PUT -d "{\"start\":false, \"type\":\"streaming\"}"
http://localhost:54664/control
```

To set the frequency range of the measurement devices, following fields are used:

```
{
    "frequencyCenter"   : 1200000000,
    "frequencySpan"     : 44000000,
    "frequencyBins"     : 448,
    "referenceLevel"    : -20,
    "type"              : "capture"
}\n'
```

Using curl:

```
curl -X PUT -d "{\"frequencyCenter\":1920.0e6, \"frequencySpan\":200.0e6,
\"type\":\"capture\"}" http://localhost:54664/control
```

An alternative way would be to specify the start and end frequency.

```
{
    "frequencyStart"   : 75.0e6,
    "frequencyEnd"     : 6000.0e6,
    "type"              : "capture"
}\n'
```

To start or stop the autorotation of all antennas in the remote block graph configuration, following fields are used:

```
{
    "rotate" : true,
    "type"   : "antenna"
}\n'
```

To start a record, following fields are used:

```
{
    "start"    : true,
    "filename" : “xy”
    "type"     : "recording"
}\n'
```

To save or reload the running mission, following fields are used:

```
{
    "save" : true,
    "type" : "mission"
}\n'
```

```
{
    "reload" : true,
    "type"   : "mission"
}\n'
```

Configuration Data

Server Info

The "/info" endpoint provides information of the http server block, such as name, title, port, features and the name and path of the mission http://localhost:54664/info.

```
{
    "name"    : "Block_HTTPServer_0",
    "title"   : "HTTP Server",
    "uuid"    : "aaf2a8f7-11fa-45a3-bcfc-26aaf5957629",
    "port"    : 54664,
    "mission" : ""
}
```

General

All configuration items visible in the configuration blocks of the Aaronia-RTSA-Suite have a representation as JSON. The list of available configuration items can be queried with the "/remoteconfig" endpoint http://localhost:54664/remoteconfig.

The resulting JSON looks like this.

```
{
    "request": 0,
    "config": {
        "type" : "group",
        "name" : "remoteconfig",
        "label": "RemoteConfig",
        "items": [{
            "type" : "group",
            "name" : "Block_FileReader_0",
            "label": "File Reader",
…
```

The configuration data forms a tree with the root being placed into the config member of the top-level object. The second tree level is populated with the blocks of the graph, all further levels by configuration elements of these blocks.

All config items share the following fields:

type	The type of the config element

name	The machine-readable name of the element
label	The human readable name of the element

Leaf nodes carry configuration data, internal nodes have the type “group” and contain their child nodes in a member named “items”. Leaf nodes provide their current value in the member “value” and the default value in the member “default”.

Leaf Node Metadata

The following members can be found in leaf nodes and provide meta data.

min	Minimum allowed value
max	Maximum allowed value
step	Distance between two valid values
unit	Unit represented by numeric values (e.g. time)
pattern	File name pattern (glob)
values	Comma separated list of names of an enumeration type
flags	Additional flags

Set Configuration Data

All configuration items visible and editable in a configuration block of the Aaronia-RTSA-Suite can be changed via the “/remoteconfig” endpoint and a HTTP PUT request. Each configuration block has a unique name in a Aaronia-RTSA-Suite mission. This name is used to address a specific RTSA block. The unique name for each RTSA block is provided in the response of the “/remoteconfig” HTTP GET request and is static in an unchanged mission. Each “/remoteconfig” HTTP PUT request will be answered with the same response a HTTP GET request will initiate.

A JSON “/remoteconfig” HTTP PUT request starts with the following fields:

```
{
    "request" : 0,
    "config"  : {
        "type"  : "group",
        "name"  : "Block_Spectran_0",
        "items" : [ {
…
```

request	Incrementing request number
name	Unique Name of a specific RTSA block to be addressed
items	Contains the leaf nodes of the configuration items which will be changed in the addressed RTSA block

A HTTP PUT request to enable the amplifier of a Spectran V5 RTSA block for example, looks like the following:

```
{
    "request" : 1,
    "config" : {
        "type"  : "group",
        "name"  : "Block_Spectran_0",
        "items" : [{
            "type"  : "group",
            "name"  : "main",
            "label" : "Main",
            "items" : [{
                "type"  : "bool",
                "name"  : "amplifier",
                "value" : true
            }]
        }]
    }
}
```

Status Endpoint

Depending on the functionality of a configuration block in the Aaronia-RTSA-Suite, a block can report its health status at the "/healthstatus" endpoint http://localhost:54664/healthstatus in JSON format. The health status contains the current state of the configuration blocks together with its latest error. Depending on the capabilities of the block additional information is provided, like temperature values or performance statistics.

The format of the health status is composed of the same config items as the configuration data format. It consists of one main group item, named "healthstatus" with one child group item per health aware block. Each one of these block health groups consists of up to five subgroups

Info	General Info about the block, such as name, type or UUID
status	Status Information, e.g. temperature, samples processed
health	Health status
settings	Unit specific settings
components	Recursive list of sub blocks when using a system with HTTP block connected satellites

The info group contains as least the following items:

date	Last update of the health status in seconds since the epoch
name	Internal name of the block
category	Category name of the block
title	User facing title of the block
description	User facing description of the block
uuid	Global unique identifier of the block

The health status group contains at least two members:

state	One of the following states: unknown, idle, booting, ready, starting, operational, running, warning, critical
error	User facing string describing the error

User Endpoint

The “/user “ endpoint provides information regarding the current user of the endpoint, such as name, email, position or the alternative authorization token.

name	Name of the current user
email	Email address of the current user if available
token	Authorization token to be used with the alternative “RToken” http authorization method
groups	Array of group names, this user is a member of

Performance Considerations

While it is unlikely to achieve the full 250M samples of IQ data using pure JSON or a gigabit network card, it is a simple matter of a fast parser to get the full rate using a loopback connection and an appropriate programming language. TCP loopback on windows can be improved by using the TCP Loopback Fast Path (enabled by **SIO_LOOPBACK_FAST_PATH**).