IN3041UG: AVALOT API

Website: https://intona.eu

Contents

1 Introduction

1.1 AVALOT is a AES67 compatible network audio device

AVALOT devices receive and generate network audio streams that meet the AES67 standard. The configuration is easily and portably done in a human-readable JSON format over UDP or UART.
Some design decisions have been made which will be discussed briefly in the following.

1.1.1 Mediaclock, phase-coherence and PTP

As a receiver, the media clock is always extrapolated from the actual audio stream. The separately received PTP time is used exclusively for aligning the link offset, which ultimately ensures the phase coherence of all devices on the network with each other. The advantage of this procedure is that even in the absence or failure of the PTP system, there will be no disruption to the audio playback. Therefore, there are multiple lock levels. The audio playback is interrupted only if RTP lock is lost. However, this is only the case if the data stream does not arrive at the device.

1.1.2 SAP

SAP (Session Announcement Protocol) is required for AVALOT, but optional in AES67. Other session management protocols may be supported in the future. It's also possible to configure a static AES67 session without using SAP.

1.1.3 SAP Caching

AVALOT devices send out cached SAPs at a relatively high rate. The SAP was specified for old network technology and prioritized reducing traffic. AVALOT devices are likely used in environments where every second can count. This "SAP replication" can help if devices need to be setup very quickly. Likewise, SAP timeouts were reduced to avoid listing stale sessions.

1.1.4 Session Identification

The stream name (SDP "s" field) is used to uniquely identify sessions. Duplicate names are considered an invalid misconfiguration

1.1.5 IGMP in consumer switches

IGMP is the protocol for controlling multicasts in the network. The state of IGMP implementations in many consumer switches is very poor. This situation has been thoroughly analyzed and a workaround has been implemented to improve the situation. This is supposed to help against badly implemented IGMP-snooping. (See igmp_hack.)

1.2 The API protocol

The device can be configured over network. It exchanges commands and responses as JSON objects. The main intention is that this is used as API by control programs, although you can also send and receive raw commands manually.

Each UDP packet contains one or multiple commands as JSON objects, formatted as single line of text. These UDP packets are sent to port 7054. The device will respond with an UDP packet containing JSON object, sent back to the sender's IP, MAC and UDP port.

Here is an example:

User => Device

```
{"command": "set_params", "api_version": 6, "seq": 123, "stream": {"name":
"HDLM8-1302 : 8", "link_offset": 48, "output_channels": [0, 1, 2, 3] }}
```

Device reply

```
{"seq": 123}
```

Here the ”set_params” command is used to subscribe the AES67 stream that was propagated as ”HDLM8-1302 : 8”. The reply indicates success, because it does not contain an ”error” field. The ”seq” field is for free use by the sender (but must be a JSON number), and will be sent back with the reply. It can be used to associate responses with commands.

Erroneous requests (including JSON syntax errors etc.) will result in a response with the ”error” field set.

1.2.1 UART Interface

The hardware module can be addressed internally in the same API language via serial UART interface (115200 Baud, 8N1), with the advantage that the network connection does not have to be configured. This is specifically tailored to the OEM integrator to access the module internally, and in every situation.

Every packet sent over UART must be either terminated with a null byte or a new line character (either ’\n’, ’\r’ or both). The sent data is immediately sent back as an echo. This facilitates human interaction with the interface and creates a practical termnial. However, this might be a problem if software is to use the interface automatically. To disable echo, the character 0x01 (i.e. byte value = 1) can be sent. AVALAOT will no longer send an echo for the remaining power cycle. In your software code, it is recommended that each command just starts with a 0x01 byte.

All packets sent by AVALOT end with ’\n’ plus null byte.

Note: it is quite possible that additional debug information is sent via the serial port. This is not packaged in JSON format. The recipient should therefore check whether the lines start with ’{’ and end with ’}’ in order to distinguish the debug output from JSON data.

1.3 Getting started

This section shows how to initialize the device and how to start audio streaming.

1.3.1 Finding the device on the network

Unless configured otherwise, each device uses an automatically assigned link-local IP address (RFC 3927)[1]. Addresses start at **`169.254.1.0`**, but are generally unpredictable and unstable. For example, if a device is restarted, it may pick a different address. Usually, it will attempt to reuse the previous address, but you cannot rely on this. (You could use the set_params command to assign a static address to a device to avoid this.)

You should perform device discovery by sending **`device_info`** commands to the broadcast address (**`169.254.255.255`**). These commands can be sent periodically to provide live updates on settings, and whether the device is still up. To avoid overloading the device, an update period of no shorter than 500 ms is recommended. This is especially important if several clients are on the network.

[1] https://tools.ietf.org/html/rfc3927

Use the **device_id** field (queried with **device_info**) as unique, stable device identifier.

1.3.2 Sending and receiving commands (via CLI on Linux/macOS/msys)

Since the protocol simply sends and receives UDP packets containing JSON text, you can send and receive commands with common open source tools:

```
rlwrap -S '> ' nc -u 169.254.1.0 7054 | jq
```

nc sends and receives UDP packets (as text in this case), **rlwrap** provides line editing and a history, and **jq** pretty prints JSON responses.

1.3.3 Setting up audio streaming (via CLI)

The device supports SAP (Session Announcement Protocol, RFC 2974)[2] AES67 session discovery. Passive discovery is always active. The device does by default not play any audio from network. The network API can be used to play a specific stream.

All important settings and information can be queried with:

User => **Device**

```
{"command": "device_info"}
```

Which results in this reply (only AES67 relevant parts shown):

[2] https://tools.ietf.org/html/rfc2974

Device reply

```
{
  ...
  "stream": {
    "name": "BKLYN-II-0d0c9a : 31",
    "link_offset": 48,
    "gain_db": 0,
    "output_channels": [ 0, 1 ]
  },
  "streams": {
    "list": [
      {
        "n": "BKLYN-II-0d0c9a : 31",
        "i": "1 channels: Analog R",
        "c": 1
      },
      ...
    ]
  },
  "rtp": {
    "lock": 3
  },
  ...
}
```

This example run returned two AES67 sessions. The device is streaming the first one (according to the stream.name field).

A specific session can be selected with:

User => device

```
{"command": "set_params", "stream": {"name": "BKLYN-II-0d0c9a : 32"}}
```

The set_params command has a large number of parameters, which can be used to control every detail about the device. The stream.name parameter controls which session should be selected. Internally, the name is matched against the internal list of sessions, and the first matching session is selected. By default, the name is empty, and no session matches.

The streaming status is indicated by the rtp.lock field.

Using this command persists all set parameters to the device flash memory (see section below).

1.4 Persistence and reboots

Most device settings are persistent. They are written to flash memory. After boot, the settings are restored from flash. There is a delay of about 1 second before the device starts writing the changed settings to the flash. This delay counts since the most recent change. For this reason, the device should not be power cycled immediately after a change. (The device should survive power cycles at any possible moment, but recently changed settings will not be restored.)

It can take a while until a rebooted device recovers and streams audio. The following processes incur additional delays:

- Waiting for SAP announcements. (Unless they are cached on flash; if they are not cached, not replicated by other devices, and the sender is a Dante device, it can take up to 30 seconds.)
- Waiting for PTP clock stabilization. (Requires receiving at least 2 PTP announcements, streaming may start before; can take about 2 seconds.)

However, when already subscribed to an active stream, the time until valid audio data is played then depends only on establishing the network link in the PHY chip. This is usually one to three seconds after powering up.

The processes listed above happen in parallel. For network API access, the device needs to negotiating a link layer IP address, which takes about about 7 seconds on average. If static IP configuration is used, the API is available as soon as the network link is up.

1.5 Firmware updates

The device runs a TFTP server for firmware updates. It accepts firmware binaries under the name "firmware.bin". All other files are rejected. Update can be done with any TFTP client, for example curl:

```
curl -T path/to/firmware.bin tftp://169.254.1.0
```

Firmware updates require a reboot. Firmware updates can reset all or some setting.

2 API protocol definition

2.1 Basics

2.1.1 Network layer

The protocol uses JSON encoded in UTF-8 text, encapsulated in UDP. Each UDP packet payload contains exactly 1 JSON object (a list of fields enclosed with "**{**" and "**}**"). A JSON object cannot span multiple UDP packets. A JSON object represents a single command. The convention is to add a trailing newline character ("\n", ASCII 0x0A) at the end of the JSON text.

UDP packets are exchanged between devices and API clients. Clients send commands as UDP packets to port 7054. The device will attempt to send a response packet back to the client immediately, using the command packet's source IP/port as destination IP/port for the response packet.

The device does not support receiving fragmented IP packets. All commands are assumed to fit within 1472 bytes of UDP payload. On the other hand, responses by the device may be sent as fragmented UDP packets if they exceed the MTU.

A MTU of 1500 bytes is assumed.

Be aware that UDP is an unreliable protocol. Packet loss and reordering is possible. The device sends exactly one response for each request, and this response could get dropped by the network. If you do not receive any response, you cannot know whether the request or response was dropped (or whether the device went offline). You may have to repeat the request until you get a response (assuming the command is idempotent - most are). Since the network has to reliably transport AES67 streams, it is assumed that everything operates with some headroom below the maximum possible network bandwidth, and packet dropping does not actually happen in practice.

2.1.2 JSON

JSON, as used by the AVALOT API protocol, is specified by RFC 8259[3] and RFC 7493[4]. The protocol requires that all top-level JSON values are JSON objects, and that each packet contains a single JSON object. In particular, it is assumed that JSON numbers use double floats (binary64 in IEEE 754-2008[5]), which limits their integer precision to about 53 bits (see RFC 7493 section 2.2[6] for details).

The command reference will use the term "integer" for numbers which are representable as signed 32 bit integers (sometimes unsigned 32 bit integers).

2.1.3 Commands

The UDP payload consists of text that can be parsed as a single JSON object. There must be no 0 bytes or other non-text data. Whitespace can be present, but should be avoided to reduce the packet size. A final newline should be added at the end of the JSON text. (It's currently not required, but this is a threat that this may change.)

On the IP level, the command must be a single IP fragment.

Each JSON object sent to the device represents a single command. It can contain the following fields:

Name	Type	Meaning
command	String	Required. Name of the command to run.

[3] https://tools.ietf.org/html/rfc8259
[4] https://tools.ietf.org/html/rfc7493
[5] https://en.wikipedia.org/wiki/IEEE_754-2008
[6] https://tools.ietf.org/html/rfc7493#section-2.2

Name	Type	Meaning
seq	Number	Sent back with responses, and is otherwise not interpreted or changed by the device.
api_version	Integer	Optional, but passing this is highly recommended. The current protocol version is 7.
add_crc	Boolean	If true, a CRC will be appended to the JSON data of the response. (See Responses section below.)
(other)	...	Command specific parameters are located in the same top-level JSON object.

All fields except ”command” are optional. The ”api_version” should also be passed.

The most important commands are ”device_info” and ”set_params”. All other commands are obscure and rarely needed.

2.1.4 Reponses

Responses indicate success or failure of a previously sent command. There is exactly one response per command. (Minus dropped network packets.)

The payload contains a single line of JSON text, with redundant whitespace trimmed (clients shouldn’t rely on this) and a single newline character at the end (clients can rely on this).

On the IP level, large responses are sent as multiple IP fragments. These are reassembled to a large UDP packet, whose payload follows the described format. Large responses usually happen with with ”device_info” commands, and can be mostly avoided by using the ”select” parameter.

The response JSON object contains the following fields:

Name	Type	Meaning
seq	Number	Always present. The same value as the command packet, or 0 if it was absent.
response	String	Command which triggered this response. May be missing if unknown or the firmware is old. Present since api_version 7.
api_version	Integer	API revision used by device. Present since api_version 7.
error	String	If present, a protocol error of some kind has happened. This is usually used for malformed commands only. It’s a string value, that contains an error message. It is not intended to be machine-readable.
warning	String	If present, warnings or errors. This is a string value and contains newline characters (one message per line). This is not necessarily an error. It is not intended to be machine-readable. The main use of this field and the ”error” field is to help with debugging.
(other)	...	Command specific response fields are located in the same top-level JSON object.

If ”add_crc” was set to true in the command, the response packet uses a slightly modified format. The four byte sequence ”\n//#” (0a 2f 2f 23 in hex) is appended after the JSON payload (instead of a final ”\n” byte), followed by 8 ASCII digits representing the CRC32 of the payload before the four byte header. For example, the byte sequence ”0a 2f 2f 23 61 39 34 64 61 31 65 61” at the end of the packet indicates the CRC32 value 0x a94d a1ea. If present, this byte sequence will always be 12 bytes long and end with the packet’s payload. The CRC32 uses the same polynomial as Ethernet and zlib. This can be useful if fragmented UDP packets occur, and the UDP checksum is not trusted. (In single fragment cases, the packet is protected sufficiently by the Ethernet CRC.) The format was chosen such that we can claim it’s still a text based protocol. Naive clients can just not set this field. Then the ”\n” byte is always the last byte of the payload, and the only one in the payload.

2.1.5 Extensibility

JSON was picked as base of the AVALOT protocol to make it easier to extend the protocol in the future. For this reason, both device and clients must make the following assumptions:

- The ordering of object members is arbitrary. Both the device and the client are required to deal with any ordering of members in an object. It is possible that members change their order on firmware updates, or randomly.
- Unknown object members should be ignored. The device will add a warning to the response object if an object member is unknown, but will otherwise ignore it.
- If new parameters are added to a command, it must be done in a backward compatible way.
- Any code handling commands or responses must be forward compatible, and ignore unknown parameters.

2.1.6 Packet drops and reordering

UDP can drop or reorder packets. Clients can use the ”seq” field to avoid confusion due to reordering. It can also be used to detect packet drops (for example, if no response for a command is received, the client can issue a redundant command to check whether the device is still operating). If a response is lost, the client does not know whether the command was executed, or the result status. In this case the client needs to check the device state, or send the command again to be sure.

2.1.7 Retrieving updates and events from a device

There is no mechanism for receiving any kind of events with the API. Poll the device by sending **`device_info`** commands. A polling period of 500 ms or higher is recommended.

3 API protocol reference

All commands can include standard parameter fields described in the section above. The same applies to responses. Only command-specific request/response fields are listed in the tables below.

3.1 device_info

Retrieve information about device firmware, AES67 settings/status/session list, and more.

3.1.1 Description

This command reports general information about device and network status. This command should be used for device discovery.

The ”set_params” command is an important companion command. It mirrors most fields in the ”device_info” command, and can be used to set the corresponding fields. All references to setting fields in the parameter description above is in context of the ”set_params” command.

The ”select” field can be used to retrieve only some information. It is an array of strings, where each string is the name of a JSON sub-object. Only sub-objects mentioned in this list are sent in the response. For example, if you send **`select:[ "net"]`**, then other sub-objects like ”streams” are not sent, only the ”net” sub-object. If the field is missing, everything is sent.

3.1.2 Example

User => **Device**

```
{"command": "device_info", "seq": 123, "api_version": 6}
```

Device reply (prettified)


```
{
  "seq": 123,
  "product": "ISAAC",
  "firmware_version": "v1.5.2",
  "api_version": 6,
  "fw_date": 1654077358,
  "device_id": "00313753384d37373038303432303139",
  "product_id": 1001,
  "net": {
    "mac": "1A:7D:9C:4F:26:3A",
    "ip": "169.254.225.237",
    "static_ip": "0.0.0.0",
    "igmp_hack": true
  },
  "stream": {
    "name": "AVIOUSB-5346cb : 2",
    "link_offset": 48,
    "gain_db": 0,
    "output_channels": [ 0, 1 ]
  },
  "streams": {
    "list": [
      {
        "n": "AVIOUSB-5346cb : 2",
        "i": "1 channels: Left",
        "c": 1
      }
    ]
  },
  "rtp": {
    "lock": 0
  },
  "ui": {
    "order": 0,
    "name": "",
    "loc": "",
    "memo": ""
  },
  "logging": {
    "en": false
  }
 }
```


3.1.3 Parameters

Member	Type	Meaning
select	JSON array	Optional. Each array item must be a string. Each string can be the key of a sub-object to include in the response. If missing, all sub-objects are returned.
metering	Bool	Enable metering data broadcast.

3.1.4 Response

Member	Type	Meaning
product	String	General product ID. This is a legacy thing and shouldn't be used anymore. Always "ISAAC" (read-only).
firmware_version	String	Firmware version identifier, usually a release version string (read-only). Note: never parse this in program code! Some firmware binaries use a git hash instead of a version number.
api_version	Integer	API revision used by device, may be different from client's version in "api_version" field (read-only). Note: older firmware versions do not have this field.
fw_date	Integer	UNIX-time of firmware build date (read-only). Note: older firmware versions do not have this field.
fw_magic	Integer	32 bit unsigned firmware magic, used to check which firmware to send on updates. Note: some older firmware builds send a signed 32 bit number, or do not have this f
device_id	String	128 bit globally unique hardware identifier in hexadecimal, for example "00313553504e52433033303436303535". This is the device's builtin UUID. It never changes, unless the hardware is physically replaced. (Read-only.)
firmware_update_error	String	Optional. If present, a human readable error description on firmware update errors. Always cleared on reboots. (Read-only.)
product_id	Integer	Product ID (see table below). (Read-only, normally.)
hw_channels	Integer	Available audio channels in this hardware. Note: older firmware versions do not have this field. Treat the value to 1 by default.
net	Object	Network settings/state (see table below).
ui	Object	Informational fields for the Network Manager GUI (see table below).
stream	Object	AES67 parameters (see table below).
streams	Object	AES67 stream list (see table below).
rtp	Object	AES67 state (see table below).
logging	Object	Logging control (see table below).
link_state	Object	Ethernet port(s) link state (see table below).

3.1.5 product_id (Product IDs)

The "product_id" field contains a fixed number that is defined by Intona and used to differentiate between different types of end devices.

Raw value	Name
0	(Never used, might happen on major device flash corruption.)
1 .. 999	Reserved for third party.

Raw value	Name
1000+	Maintained by Intona, ask them.

3.1.6 net (Network settings/state)

The response "net" field is an object with the following fields:

Member	Type	Meaning
mac	String	Current MAC address, formatted as 6 groups of 2 hexadecimal numbers separated by ":", for example "AB:CD:12:34:56:7E". (Read-only.)
ip	String	Current IP address formatted as quad-dotted IP address, for example "169.254.1.0". (Read-only.)
static_ip	String	Configured static IP address (the same format as the "ip" field), or "0.0.0.0" if the IP address should be dynamically allocated. Applied after device reboot.
igmp_hack	Boolean	If true, enable an IGMP workaround, that is supposed to help with broken IGMP-snooping switches. (Enabled by default.)

3.1.7 ui (UI fields)

The response "ui" field is an object with the following fields:

Member	Type	Meaning
order	Integer	Order ID (used for GUI device list sort order).
name	String	User-assigned device name (max. 127 bytes).
loc	String	User-assigned device location text (max. 127 bytes).
memo	String	Text field for free use (max. 127 bytes).

The strings are supposed to be encoded as UTF-8 (usually enforced by JSON), though the device will accept any encoding in JSON or these fields.

None of the values in this object are actually interpreted by the firmware. They are for use by GUI tools.

3.1.8 stream (AES67 settings)

The response "stream" field is an object with the following fields:

Member	Type	Meaning
name	String	Stream name, used to select session.
link_offset	Integer	Link offset in samples.
gain_db	Float	Gain correction in dB. Must be <= 0.

Member	Type	Meaning
output_channels	Integer Array	Deprecated, use ”ch0”...”ch*n*”. Set the source channel for each hardware output, starting from 0. Set -1 to disable the source. If a stream source channel is unavailable, it will be treated as ”-1” and muted.
ch0 .. ch*n*	Integer	Set the source channel for the first hardware output, starting from 0. Set to -1 to mute this output.
rtp	Object	Manual AES67 session settings (see table below).

Note: there are two methods to set the source channel(s). Using of ”ch*n*” is recommended. If both ”output_channels” and ”ch0” are given in the same command, the behaviour is undefined.

3.1.9 Example using output_channels

User => **Device**

```
{"command": "set_params", "stream": { "ch0": 0, "ch1": 2, "ch2": -1, "ch3": 2 }}
```

Assuming four hardware outputs, the example above sets the audio sources as follows:

Hardware Output	Stream Channel
0	0
1	2
2	muted
3	2

3.1.10 rtp (AES67 manual session)

The response ”rtp” field is an object with the following fields:

Member	Type	Meaning
en	Bool	Manual session enabled. If true, stream.name is ignored, and the settings below are used to receive the AES67 stream.
ip	String	RTP destination IP address formatted as quad-dotted IP address, usually a multicast address, for example ”239.81.83.67”.
dp	Integer	RTP UDP destination port, for example 5004. Ignored if 0.
dc	Integer	RTP UDP destination port count. Set this to 0 or 1 if unused. If larger than 1, this assumes multiple RTP streams. See remarks below for details.
sp	Integer	RTP UDP source port. Ignored if 0. This should normally be set to 0, but some AES67 sources require using this.

Member	Type	Meaning
sc	Integer	RTP UDP source port count. Set this to 0 if unused. This cannot be combined with a dc port range. See remarks below for details.
sr	Integer	RTP audio data sample rate, for example 48000.
ss	Integer	RTP audio sample size. Must be 16 or 24.
cc	Integer	RTP audio channel count, for example 2. If multiple RTP streams are used, this is the total channel count.
pt	Integer	RTP payload type. This is the PT field in the RTP header. 96 is a common value.
ci	String	PTP clock ID. This is the clockIdentity PTP header field, as little endian 64 bit number (byte swapped), as a JSON string. If this is 0, matching the clock ID is disabled (this uses 0 as magic value and assumes no real PTP clock uses this value). It is formatted as a JSON string because JSON numbers are usually not 64 bit safe. Sender/receiver may format this as decimal, or as hex by prepending ”0x” to the number.
cd	Integer	PTP domain number. This is the domainNumber PTP header field. If ”ci” is set to a value other than 0, this field is ignored. If ”ci” is set to a value of 0, the domain number is used to select the PTP clock.

3.1.11 Manual session configuration

If ”en” is set to true, manual session configuration is enabled. In this case, the selected SAP session is ignored. None of its parameters are used for streaming audio. Only the manual session configuration fields are used. Settings other than the session selection, such as link_offset or output channel selection, are still used. You should normally set the ip, dp, sr, ss, cc, pt, ci fields. Verify them before attempting to stream. Use other fields only if you know what you’re doing.

3.1.12 Multiple RTP streams with port ranges

The dc field can be used to stream from multiple RTP sub-stream. All sub-streams must use the same audio format, PTP clock, and channel count. Each sub-stream is identified by its UDP port and must use the same RTP parameters and IP destination address. The first port is given by the dp value, the last port is dp+dc-1. The dc field sets the number of RTP sub-streams. The total number of channels (set by cc) must be the RTP packet channel count multiplied with the number of ports (set by dc).

It is possible to use a source port range to identify RTP sub-streams with the sp and sc fields. This should normally not be used, but some AES67 sources use the same UDP destination port and require using the source port to distinguish streams. You cannot use both destination and source port ranges at the same time. (If a source/destination port range is used, the destination/source port must be either a single port, or unset by specifying 0. If the port is unset, RTP packets with any ports are accepted.)

3.1.13 PTP clock selection

The ci field is normally used to select the PTP clock source and the preferred method. Using the cd field is not recommended, but it is needed with some AES67 sources. the cd field value is used only if ci is set to 0.

3.1.14 streams (AES67 session list)

The response ”streams” field is an object with the following fields:

Member	Type	Meaning
list	JSON array	Actual session array, see below.

Each item in the ”list” array is an object with the following fields:

Member	Type	Meaning
n	String	Session name
i	String	Session info
c	Integer	Channel count

This is the list of discovered sessions. The session name is the ”s” field in the SDP, and is compared to stream.name to select a session. The session info is the ”i” field in the SDP. The channel count is also as indicated in the SDP, and gives the upper range of the stream.output_channels field (exclusive).

3.1.15 rtp (RTP state)

The response ”rtp” field is an object with the following fields:

Member	Type	Meaning
lock	Integer	AES67 streaming state

The lock field can have the following values:

Raw value	Meaning
0	No PTP, no RTP data
1	PTP locked, but no RTP data
2	No PTP, but RTP locked
3	PTP and RTP locked

”Locked” in this context means ”synchronized to”. Streaming is possible at state 2 or 3. State 2 may be indicated by a blinking yellow Ethernet port LED, at state 3 it glows normally. RTP is for Real-Time Transfer Protocol, the part of AES67 that transports actual audio data over network.

3.1.16 logging (Logging state)

The response ”logging” field is an object with the following fields:

Member	Type	Meaning
en	Boolean	Whether logging is enabled

Logging is a debug feature. The log is useful to firmware developers only. Technically inclined people may be able to interpret parts of the log, but it’s not recommended. Logging makes the device broadcast UDP packets to IP 255.255.255.255 on port 7055. The log state is off by default, and is not restored across firmware reboots.

Log packets have a format that is slightly different from normal protocol messages. The first part of the UDP payload consists of JSON text, like in normal messages. This is followed by a 0 byte, a binary length field, and the log text.

Position	Length	Field
0	N	JSON text of length N bytes
N + 0	1	Literal 0 byte (N found by searching for the first 0 byte)
N + 1	2	Little endian uint16_t log text length
N + 2	length	ASCII log text, using length field above

This binary length field and log payload is found by searching for a 0 byte. JSON cannot legally contain a 0 byte.

This format was chosen because we wanted to continue using JSON for its positive properties, all while reducing device MCU load by not having to turn the log message into a legal JSON string value (escaping).

The JSON text is an object with the following fields:

Member	Type	Meaning
command	String	Contains the string ”raw_log”, otherwise it’s not a log message
pos	Integer	Log byte position, can be used to detect dropped log messages
overflow	Boolean	Internal overflow flag (not always present)

The ”pos” field is the number of log bytes output so far. The log receiver can add the packet’s log text length to the ”pos” field to know the ”pos” field of the next log packet. If the values mismatch, a log packet must have been dropped or reordered. The value of this field is an unsigned 32 bit integer and rolls over.

”overflow” indicates that the device log buffer (which is of limited size) overflowed before the contents could be sent to the network. The ”pos” field is also discontinuous in this case.

3.1.17 link_state (Ethernet ports link state)

The response ”link_state” is an object with the following fields:

Member	Type	Meaning
list	JSON array	Actual integer array, see below.

Each item in the ”list” array is an integer value that reflects the link state from the indexed ethernet port in the hardware.

Link state values:

Value	Meaning
0	No link
1	100 Mbps link active
2	1000 Mbps link active
3+	Reserved for future use

Example if hardware has two Ethernet ports with the first port being inactive and the second port having a 100 Mbps link:

Device reply (prettified)

```
{ "link_state": { "list": [ 0, 1 ] } }
```

Example if hardware has a single Ethernet port with the port having a 1000 Mbps link:

Device reply (prettified)

```
{ "link_state": { "list": [ 2 ] } }
```

3.1.18 Metering mechanism

If the **`metering`** parameter is set to true, the device will broadcast separate metering packets to IP 255.255.255.255 on UDP port 7055. The device sends 10 packets per second. The request expires after 1 second. The client should send **`device_info`** packets with the **`metering`** parameter set to true periodically every 500 ms to renew the request. This mechanism is not available on the serial UART interface, because the metering packets are always sent to UDP.

The UPD packets contain JSON text like all API packets:

Member	Type	Meaning
command	String	Contains the string ”broadcast_info”, otherwise it’s a different message
device_id	String	Unique device ID, like in ”device_info” responses.
metering	Object	Metering information, if present.

The metering object is as follows:

Member	Type	Meaning
channel_peaks	Array	Array of (tbd)
channel_overdrive	Array	Array of (tbd)

3.2 set_params

Write various device settings.

3.2.1 Parameters

This command mirrors a number of fields in the device_info response. All fields are optional, and only parameters sent with the command are actually set.

See the response fields of the device_info command for type and meaning of each field, and whether it can be written.

3.2.2 Response

Success/error only. If an error happens, and the request contains multiple parameters, it's possible that the request is partially applied. In this case, it's unpredictable which parameters were applied and which not. Use the device_info command to confirm the new values.

3.2.3 Description

This command can set most fields which appear in device_info. Name, location, JSON data type, and meaning are exactly the same as the fields in device_info.

If the client needs to change multiple parameters, it's recommended to set all parameters at once, using a single "set_params" command.

3.2.4 Example

This sets the link offset to 64 samples and leaves all other parameters untouched:

User => Device

```
{"command": "set_params", "stream": { "link_offset": 64 }}
```

Device reply

```
{"seq": 0}
```

This sets multiple fields (set first and second channel sources, set link offset to 48 samples, set memo to "hello"), and leaves all other parameters untouched:

User => Device

```
{"command": "set_params", "stream": { "output_channels": [ 0, 1 ], "link_offset": 48 },
"ui": { "memo": "hello" } }
```

Device reply

```
{"seq": 0}
```

3.3 show_rtp_status

Show information about current AES67 streaming (network to device).

3.3.1 Parameters

None.

3.3.2 Response

Member	Type	Meaning
ip	String	IP address of source stream.
port	Integer	UDP port of source stream.

Member	Type	Meaning
clock_offset	Integer	SDP mediaclk parameter (0 if unset)
link_offset	Integer	Offset chosen with set_session command.
samplesize	Integer	Audio sample size in bits per channel
samplerate	Integer	Audio sample rate
channels	Integer	Number of channels sent by source stream.
output_channels	Array	Channels used, as chosen with set_session command.
packet_drops	Integer	New packet drops since last command.
packet_drop_last_ms	Integer	Absolute device time of last packet drop in ms. Can wrap around.
rtp_received_last_ms	Integer	Absolute device time of last RTP audio packet received. Can wrap around.
clock_locked	Integer	Complete lock. (device_info rtp.lock has more detailed information.)

3.3.3 Description

This shows the status of AES67 streaming, as well as some of the selected parameters.

This command is for debugging, and you should probably not rely on it being present or compatible in the next firmware version.

3.3.4 Example

User => Device

```
{"command": "show_rtp_status"}
```

Device reply

```
{
  "seq": 0,
  "ip": "239.69.214.91",
  "port": 5004,
  "clock_offset": -1479386051,
  "link_offset": 64,
  "samplesize": 24,
  "samplerate": 48000,
  "channels": 1,
  "output_channels": [
    0
  ],
  "packet_drops": 2147483647,
  "packet_drop_last_ms": 693922,
  "rtp_received_last_ms": 823747,
  "clock_locked": true,
  "ptp_sync_last_ms": 823543
}
```

3.4 sap_purge

Remove sessions from the device's session cache.

3.4.1 Parameters

Member	Type	Meaning
age	Float	Maximum session age in seconds that should be preserved (older sessions are deleted). (Default: 60)
blocktime	Float	Time in seconds during which new SAPs should be ignored. (Default: 0)

3.4.2 Response

Success/error only.

3.4.3 Description

The device caches previously discovered sessions (using the SAP protocol) and hold them in RAM and on the flash. There is currently a session timeout of 100 seconds, after which a session is automatically deleted. This command can be used to remove a session immediately. Sometimes, it can help resolving playback problems by removing broken cache entries.

The ”blocktime” parameter can be set if sap_purge is supposed to be run on all devices in the network. The problem is that the devices will never execute the command at exactly the same time, and bogus SAP packets still could be propagating through the network. In particular, devices will send their own bogus cached/replicated SAPs to all other devices. To avoid that devices immediately readd bogus sessions, this parameter can disable SAP processing temporarily.

3.5 blink_leds

Let some LEDs blink on the target for some seconds.

3.5.1 Parameters

None.

3.5.2 Response

None.

3.5.3 Description

This command is forwarded to the serial port. This can be used to make it easier for the user to locate the device. It is recommended to blink all LEDs for three seconds after receiving the command.

3.6 reset_settings

Wipe all settings.

3.6.1 Parameters

None.

3.6.2 Response

None.

3.6.3 Description

This resets all user settings, and reboots. Like with the ”reboot” command, no response is sent. The device_id is not changed.

3.7 reboot

Reboots the device.

3.7.1 Parameters

None.

3.7.2 Response

None.

3.7.3 Description

The device is reset, as if power-cycled. There is no response, because the hardware is reset before a response can be sent out.

3.7.4 Example

User => **Device**

```
{"command": "reboot"}
```

Document version: 56 / Jun 05, 2024 11:46