BlueRiver Platform

BlueRiver API 2.14 Developer Reference Guide

Semtech
Proprietary & Confidential

Revision History

Version	Date	Revision
2.14	May 17, 2018	Update documentation for API version 2.14: • Colour quantization range added to video format arguments.
2.13	April 13, 2018	Update documentation for API version 2.13: • Add **program** command for first-time programming. • Add I2C events support. • Add CEC support. • Support for programming the on-board Icron Extreme USB extender. • Video compressor control. • Color generator control. • Report the frame rate fractional part (e.g. 59.94 instead of 60). • Add **boot_status** member to device status to report when a device booted on the golden firmware image. • Add **update_in_progress** member to device status to report when a firmware update has been started on the device. • Add configurable **locate_mode** property to device configuration to control locate mode. • Add **error_code** member to device status to read error code reported by a device. • Add **firmware_details** member to firmware identity which provides the version of both firmware images (primary and golden). • Optionally enabled auto-stop of HDMI stream 1. • Allow subscription_type and subscription_index arguments to be omitted in the **leave** command.

2.12	November 30, 2017	Convert document to the Semtech format.
		Update documentation for API version 2.12:
		• Add **vendor_id**, **product_id** and **firmware_comment** members to the device identity.
		• Add **supports_production** member to firmware items in the return value of the **list firmware** command.
		• Add **api** keyword to the **switch** command which allows data to be routed to the API server without having to specify the IP address explicitly.
		• Add **mode filter_rs232_event** command to filter out **RS232_RECEIVED** events.
		• Support Pronto infrared codes up to 256 burst pairs:
		  ○ Increased maximum string length in **send** command argument.
		  ○ Add **data_length_max** member to the infrared encoder node status.
		• Update description of the start command to reflect the fact that the multicast IP address allocation range is now configurable.
		Corrections and clarifications:
		• Improved examples in the description of the switch command.
		• Remove references to the low-level multiview API which was never fully documented and is deprecated.
		• Fix typo in EDID header in the description of the **send edid** command.
		• Discourage the use of the start command without a stream type or index.

2.11	July 5, 2017	Update documentation for API version 2.11:
		• Add support for a fourth digit in release numbering:
		  ○ Updated section 1.2 Application Compatibility and API Version Numbering Scheme.
		  ○ Updated section 4.3 Command require.
		• Add DEFAULT LED function to make restoring LEDs back to their default function easier.
		• Add support for LED function control (blinking) on NT1000/NT2000 devices.
		Corrections and clarifications:
		• Move Usage Examples section closer to the beginning of the document
		• Add connection setup example to Usage Examples section.
		• Add section 1.10 Common Pitfalls.
		• Rewrote parts of the following sections for clarification:
		  ○ Section 1.3 Events

2.10 rev.1	June 7, 2017	Corrections and clarifications: • Add missing configurable properties for I2S_AUDIO_INPUT node • Clarify that the audio_details member of the HDMI_DECODER node status, not the has_audio_details member, is undefined when source_stable or has_audio_details is false. • Clarify that the number of audio channels for an I2S audio input can vary between 2 and 8, not 2 and 16. • Clarify that the member that provides sampling frequency information in the I2S_AUDIO_INPUT

Version	Date	Revision
		node configuration is named sampling_frequency, not sampling_rate.

2.10	June 7, 2017	Update documentation for API version 2.10:
		• Add I2S audio return channel support:
		  ○ I2S audio subscription can be used as source for analog audio input/output on a transmitter device
		  ○ I2S audio subscription can be used as source for I2S audio output on a transmitter device
		• Add GPIO support:
		  ○ Add GPIO node
		  ○ Add set gpio command
		  ○ Add gpio subset to get command
		• Add NULL_SOURCE node
		• Command test memory can now target transmitter device(s)
		• Add support for cost-effective devices without support for audio/video processing:
		  ○ Add audio_downmix_support member to HDMI_DECODER node status
		• Add has_audio_details and audio_details members to HDMI_DECODER nodes status.
		• Add sampling rate and number of audio channels configuration to the I2S_AUDIO_INPUT node.

2.9	March 13, 2017	Update documentation for API version 2.9:
		• New genlock scaling display mode.
		• New factory command to restore factory default settings.
		• New video member in SCALER node status
		• Add support for I2S audio transport:
		  ○ New I2S_AUDIO stream
		  ○ New I2S_AUDIO_INPUT node
		  ○ New I2S_AUDIO_OUTPUT node
		  ○ New I2S audio sources available for analog audio and HDMI audio outputs (ANALOG_AUDIO_OUTPUT and HDMI_ENCODER nodes)
		Corrections and clarifications:
		• Clarify that the output frame rate must match the source frame rate in genlock display wall mode.
		• Clarify that the HDMI_ENCODER node main input is not a configurable input. This document was previously inconsistent on this point.
		• Clarify that the switch command returns the new settings of the source, not the destination.
		• Rewrite the introductory text to section 8 Configurable Device Properties to better reflect the purpose of this section.
		• Update and correct the list of node types in the table in section 7.7 Nodes (Node).
		Others:
		• Create new section "with an example for enabling and disabling frame rate conversion.
		• Add cross references in all sub-sections of section 8 Configurable Device Properties.

2.8 rev.2	January 16, 2017	Update legal notice.
2.8 rev.1	December 19, 2016	Corrections and clarifications. Added missing IP status in NETWORK_INTERFACE node of device object.

2.8	November 15, 2016	Update documentation for API version 2.8:
		• Extend the range of usable multicast IP addresses to 224.1.1.1 to 224.1.3.255 (224.1.1.253 and 224.1.1.254 reserved).
		• Give full control over video timings and other parameters with additional video format arguments. See section 5.2 Specifying a Video Output Format.
		• Introduce zero-latency display wall mode:
		  ○ Specify that wall mode means zero-latency display wall for devices that support it in command set video arguments.
		  ○ Add WALL_GENLOCKED (i.e. zero-latency display wall) to frame buffer display modes.
		• Support for video scaling in display wall and for adding black bars to preserve aspect ratio in a display wall through the new keep and viewport video format arguments.
		• Relax the horizontal multiview window alignment constraints (width, position, offset) to 2 pixels (was previously 32 pixels).
		• Add has_video_details and video_details members to HDMI_DECODER and HDMI_ENCODER nodes status.
		• Support for frame rate converter:
		  ○ FRAME_RATE_CONVERTER node added.
		  ○ SCALER node main input now configurable to allow selection of video from frame rate converter.

Version	Date	Revision
		○ HDMI stream 0 source now configurable to allow selection of video from frame rate converter. Corrections and clarifications: • Clarify that, in the status object of HDMI_DECODER and HDMI_ENCODER nodes, the content of the video member is only valid when source_stable is true. • Clarify in the multiview limitations that the 2-pixel tile alignment constraint also apply to the horizontal read offset, not only to tile width. • Clarify in the multiview limitations that surfaces must be 32-pixel aligned. • The return value of the set edid command is same as the get edid command, not get settings. Others: • Create new section 5.2 Specifying a Video Output Format listing available video parameters.
2.6	September 14, 2016	Update documentation for API version 2.6: • New SCALER node and set scaler command to set scaling resolution on transmitters • Specify in start command that the scaler resolution must be set prior to starting a scaled video stream • Specify that the set video multiview command is equivalent to the set multiview command called with the compatibility_4k_2x2 layout • New firmware_rc member in device identity • New hdcp_22_support_disable member in HDMI decoder node configuration • New multiview command module (multiview version 1.1)
2.5	August 2, 2016	Rename document to API Developer Reference Guide Update documentation for API version 2.5: • New get temperature command • New list multicast command • New quit command • New free keyword to free multicast IP addresses in stop command • Multicast IP address can be specified in start command • New configurable option to disable HDCP support in transmitter device • New temperature member in device status • Detection of HDCP version Corrections and clarifications: • Clarify the behaviour of the independent routing of HDMI audio feature. • Clarify infrared_data member of a INFRARED_RECEIVED represents a Pronto code. Add introduction section on streams and subscriptions Add introduction section on multicast IP address management
2.4 rev. 2	May 31, 2016	Clarify which type of errors can be expected from commands join and switch in case the peer device is disconnected.
2.4 rev.1	May 10, 2016	Corrections to API 2.4 documentation: • Fix section numbering mistake, section 4.8.13 becomes 4.8.12.3 • Correct output in examples of sections 4.8.8.3, 4.8.13.3 and 4.8.14.2
2.3 rev.1	April 29, 2016	Update revision history
2.3	April 28, 2016	Update documentation for API version 2.3. Return value of command reboot.
2.2 rev.2	April 19, 2016	Corrections to API 2.2 documentation: Remove spurious spaces in Command usage for command switch
2.2	April 18, 2016	Corrections to API 2.2 documentation: • Add missing leave command • Clarify that the stop command can be called on a receiver device (to stop the RS232 and INFRARED streams). • Clarify that omitting the stream_type argument to the stop command will stop the HDMI, HDMI_AUDIO and AUDIO streams, and that this can only be done on a transmitter device.
2.0	February 15, 2016	Next generation BlueRiver NT/NT1000/NT2000 API.

Contents

List of Figures

List of Tables

1 Introduction

This document describes what a developer needs to be aware of to use the BlueRiver API commands and develop client control applications for BlueRiver NT/NT1000/NT2000 devices.

Target audience is the developers and designers that will be implementing and utilizing the BlueRiver API.

Note: The Semtech AptoVision Product Group strongly recommends developers first read the "BlueRiver Platform Technical Overview" (ug-0022). It provides the foundation for this document.

1.1 Telnet Connection and API Commands

Control software connects to the BlueRiver Control Server and issues commands using the Telnet protocol.

This allows any Telnet client to be used to configure a BlueRiver system. By default, the BlueRiver Control Server listens on TCP port 6970.

API commands are single-line text commands. For each command, the server responds with a JSON response which contains the return status (i.e. whether the command succeeded or not) and, if successful, the return value of the command.

Developers may refer to the following RFC specifications for information on how the Telnet protocol works:

Table 1 RFC Specification Links

Title	Internet Engineering Task Force (IETF) links
Telnet Protocol Specification	http://tools.ietf.org/html/rfc854
Telnet Option Specifications	http://tools.ietf.org/html/rfc855
The JavaScript Object Notation (JSON) Data Interchange Format	http://tools.ietf.org/html/rfc7159

More details about the JSON response format are provided in in section 2 Commands Overview.

1.2 Application Compatibility and API Version Numbering Scheme

Starting with API version 2.0, the API uses a version numbering scheme which expresses compatibility information. A complete version number has the format `major.minor.revision` or `major.minor.revision.hotfix`.

For example, for API version `2.12.0.1`, the major number is `2`, the minor number is `12`, the revision number is `0` and the hotfix number is `1`.

Versions with the same **major** and **minor** number that differ only in their revision and/or hotfix number are **forward and backwards compatible** with each other. For example, if you develop for API version 2.12.0.0, your application will be able to interact without problem with API versions 2.12.5.0 or 2.12.0.7.

Revisions and hotfixes introduce fixes or internal improvements that do not affect the programming interface. For this reason, this document refers to API versions only by their **major** and **minor** number (e.g. API 2.12).

Versions that differ in their **minor** number are also **backwards compatible** but are not forward compatible. This is because versions with a new minor number introduce new commands or parameters, new JSON members or other new functionality not present in earlier versions.

For example, an application developed for API version 2.10.0 will run "as-is" on API version 2.12.0.0 without any problems. However, an application developed for 2.12.0.0 might be using features not present in 2.10.0 and as such will not be able to run on API version 2.1.0.0.

When versions differ by their **major** number, effort is required to port the application from one version to another. For example, an application developed for API version 1.0.8 will not work as-is with API version 2.0.0.

To ensure compatibility of your application with future API versions, recommended to follow these simple rules:

- As new API versions will add new members to the JSON response returned by commands, it is important that your application ignore members it does not know about.
- As new API versions will add new event types, it is important that your application ignore events it does not know about (see description of the event command).
- As new API versions will add new node, subscription and stream types, it is important that your application ignore node, subscription and stream types it does not know about (see description of the device object).

The first thing an application is expected to do immediately after connecting to the BlueRiver Control Server is to load the `blueriver_api` module using the `require` command and confirm that this command succeeds.

As part of the `require` command, the application specifies the actual version of the API for which it was developed. The `require` command checks the provided version with the current API version according to the rules described above to ensure all the required features are present. It may also enable compatibility options as required to provide the expected environment.

For more details on the `require` command, refer to section 4.3 Command require.

1.3 Events

Interesting occurrences such as device state changes or reception of RS-232 data trigger events.

There are two ways an application can handle events to keep up-to date with the state of the system. It can:

1. Poll the list of latest events with the `event` command; or
2. Enable asynchronous event notifications using the `mode async` command and be notified automatically whenever new events are triggered.

Each event has an event type, and most event types have associated data. Events with data have an associated request ID that can be passed to the request command to retrieve that data.

More details about events, including a list of supported event types is available in section 5.3 Command event.

The `request` command, used to retrieve event data (among other things), is described in section 5.13 Command request. The `mode async` command, used to enable asynchronous event notifications, is described in section 5.10.1 mode async.

1.4 Immediate versus Background Commands

There are two types of commands in the BlueRiver API:

- Commands that return immediately with either `SUCCESS` or `ERROR` status (immediate commands)
- Commands that issue a request or a task that is completed in the background after the command returns (background commands).

The lifecycle of a background command is as follows:

- In case of an error (e.g. illegal argument), the command returns an `ERROR` status immediately. However, unlike immediate commands, valid background commands return with the status member set to `PROCESSING`, and with a request ID specified in the JSON response.

- To get the command status after it was issued, the request ID can be passed to the request command:
  - If the command is still being handled, the request command returns a PROCESSING status.
  - Once the command had completed, the request command returns a SUCCESS status, as well as the return value (i.e. returned data) of the command.
- Once a command has completed, the event `REQUEST_COMPLETE` is triggered. Once the `REQUEST_COMPLETE` event has been triggered for a given request ID, the request command is guaranteed to return with a SUCCESS status when called for that request ID.

More details about the `request` command can be found in 5.13 Command request.

1.5 Keeping Track of Device Status

The BlueRiver API is used to manage and control multiple BlueRiver NT/NT1000/NT2000 based devices that operate on a single 10GbE network.

There are two ways an application can keep up to date with the current state of these devices. It can either:

1. Handle device lifecycle and state change events, either by polling events with the `event` command or by enabling asynchronous event notifications using the `mode async` command.
   a. Relevant event types in this context are `DEVICE_CONNECTED`, `DEVICE_INFO_AVAILABLE` and `SETTINGS_CHANGED`; or
2. Periodically poll the status of every device with the `get` command.

Handling events is the recommended method.

Periodically polling device status is relatively efficient for a small number of devices but when the number of commands that needs to be issued and the volume of data that is exchanged quickly increases as the system (number of devices) grows response time can be impacted. The issue is less about scalability than it is about latency (responsiveness) as the device count increases.

For more details, see sections 1.3 Events and 5.5 Command get.

1.6 Device Model

At the heart of the BlueRiver API is the device model. The device model fully describes the device status, configuration, data connectivity and supported capabilities.

The API represents the device model with a JSON object referred to as the Device Object. The Device Object contains the following information:

- `Device identity` includes immutable information on the device, for example the firmware version.
- `Device configuration` includes settings which can be directly affected by API commands.
- `Device status` includes information pertaining to the status of the device, for example whether a device is active or not and whether it is connected to a switch or in point-to-point mode.
- `Device streams` describe all the data streams being sent out from a device.
- `Devices subscription` on the other hand describe all the data streams received by a device. They are the receiving endpoint of a stream.
- Finally, `device nodes` list and describe the hardware modules present on the device along with their status and configuration. This includes HDMI input and output ports, Analog Audio Ports, Network Interface, Frame Buffer hardware etc.

For more details, see section 7 Device Object Definition (DeviceObject).

1.7 Device Life Cycle

From when a device is first discovered on the network until it is disconnected, it triggers events that the control software can retrieve. By tracking these events, the control software can keep track of each device and receive notifications of changes. For example, should a device unexpectedly disappear from the network, alerting to the possibility of device or network failure.

- When a new device is discovered on the network, the event `DEVICE_CONNECTED` occurs. Simply states that a device is seen on the network but no further communication has been established with it. As this point, device is considered active (i.e. `status.active` becomes `true` in the device model).
- The `DEVICE_INFO_AVAILABLE` event occurs after the `DEVICE_CONNECTED` occurred, once the BlueRiver Control Server can retrieve all information from the device. This signals that the device is accessible and can be controlled.
- While the device is active, it can trigger the `SETTINGS_CHANGED` event, indicating that the state of the device has changed. For example, state changes that trigger a `SETTINGS_CHANGED` event are a video source resolution change or a monitor that is connected to or disconnected from a BlueRiver receiver device.
- When a device disappears from the network, the `DEVICE_DISCONNECTED` event is triggered. The device is still listed in the device list that is retrieved with the `get` command, but it is shown to be inactive (i.e. `status.active` becomes `false` in the device model).

1.8 Streams and Subscriptions

Control of **data routing** between devices uses a stream and subscription abstraction:

- Streams are the sending endpoints while subscriptions are the receiving endpoints.
- Each stream or subscription has a type, and only streams and subscriptions of the same type can be associated together.

Streams are either joinable or switchable, depending on their type.

- Joinable streams:
  - Configured to send data to a multicast channel using the `start` command.
  - Subscriptions of receivers which are to receive this data are configured to join this channel using the `join` command.
  - A subscription can unsubscribe/leave a multicast channel either by joining another one or using the `leave` command.
- Switchable streams:
  - Configured to directly send data to a target device's subscription, to the subscriptions of a group of devices or to the API server using the switch command.
  - No configuration of the receiving device is necessary for switchable stream types.

Both joinable and switchable streams can be configured to stop transmitting using the `stop` command.

For more details, refer to sections 5.7 Command join, 5.8 Command leave, 5.16 Command start, 5.17 Command stop and5.18 Command switch.

1.9 Multicast IP Address Management

Each active joinable stream uses a multicast IP address. Allocation of multicast IP addresses can be managed either by the client through the API or by the BlueRiver Control Server (API) itself. In all cases, the BlueRiver Control Server ensures multicast IP address assignments do not conflict.

The client can manage multicast IP address allocation by specifying an address whenever a stream is started with the `start` command.

If no address is specified, the BlueRiver Control Server uses the previously assigned address, if applicable or allocates an unused address. A list of currently assigned multicast IP addresses can be obtained by calling the `list multicast` command. Whether the multicast IP address associated with a stream was allocated by the client or the BlueRiver Control Server, it can be freed by calling the `stop` command.

If multicast IP addresses are managed by the client, the independent routing of HDMI audio feature should be enabled by calling the `mode split_hdmi_audio` feature. Otherwise, the BlueRiver Control Server allocates some addresses implicitly as part of pre-2.1 API backward compatibility behaviour.

For more details refer to sections 5.9.2 list multicast, 5.10.3 mode split_hdmi_audio, 5.16 Command start and5.17 Command stop.

1.10 Common Pitfalls

This section lists a few points that are not necessarily obvious and that the developer should keep in mind.

1.10.1 Always Use mode split_hdmi_audio

Applications should enable independent routing of HDMI audio at the start of any API session by using the `mode split_hdmi_audio` command:

- This command is intended to disable specific compatibility behaviours necessary for pre-2.1 API applications and not as a control for a feature that can be meaningfully turned on or off.
- Developers that expect the latter will find that it fails the "Principle of least astonishment" criteria in more than one way. Refer to section 5.10.3 mode split_hdmi_audio for details.

1.10.2 The get settings Command Can Return PROCESSING

As an optimization, the BlueRiver Control Server caches information that allows it to return a result for the `get settings` command (that is, the `get` command with the subset argument set to `settings`) immediately most of the time.

- However, applications must not rely on the assumption that this command never returns with a status of `PROCESSING`.
- Refer to section 5.5 Command get for details. Of particular interest is section 5.5.1.4 Notes, which provides information on which subsets always return immediately and which ones can return with status `PROCESSING`.

1.10.3 Don't Use the start Command Without a Stream Type or Index

The `start` command allows the stream type or index to be omitted, in which case all joinable streams or all streams of the specified type are started, respectively. The use of these forms of the `start` command is highly discouraged.

A firmware update can cause a device to support new streams 1) of which the control application is not aware and 2) which are started by these forms of the start command. This, in turn, can cause the command to fail because of unmet preconditions or the system to malfunction because of network bandwidth issues.

An application should be explicit about which streams it intends to start.

2 Commands Overview

All commands generate a JSON response like the following:

```
{
        "status" : String,
        "request_id" : Integer,
        "result" : Varies,
        "error" : Error
}
```

Member name	Since	Type	Description
status	2.0	String	The resulting status, which is one of the following: `SUCCESS`: The command succeeded. `ERROR`: The command failed. `PROCESSING`: The command is being executed as a background command, and you are free to execute other commands in the meantime. See the request command for how to retrieve the return value once the command completes. See the `event` command and the `REQUEST_COMPLETE` event type to be notified when the command completes. In addition, the `NOTIFICATION` status is never encountered in a command response but indicates an asynchronous event notification when this feature is enabled. See description of the `mode async` command for detail.
request_id	2.0	Integer	`Request ID` assigned to your request. Set when the status is `PROCESSING` or when calling the request command, null otherwise.
result	2.0	Varies	Return value of command. Set when status is `SUCCESS`, `null` otherwise. The return value is documented in the command reference section of each individual command.
error	2.0	Error	`Error` object describing reason for command failure. Set when the status is `ERROR`, `null` otherwise.

An Error object has the following structure:

```
{
      "reason" : String,
      "message" : String
}
```

Member name	Since	Type	Description
reason	2.0	String	A machine-readable string identifying the error, which is one of the following: `ILLEGAL_ARGUMENT`: An invalid argument was specified for the command. `INVALID_COMMAND`: The entered command (or sub-command) does not exist. `INVALID_STATE`: The specified command cannot be executed given the current state of the system. The same command may succeed later if the condition is resolved. `IO_ERROR`: The operation failed due to an input/output error (for example, disk read/write failure). `OUT_OF_MEMORY`: The operation failed because new memory could not be allocated. `UNSUPPORTED_FEATURE`: The requested feature/module is not supported by the API server (see the require command). `VERSION`: The application is not supported by this version of

Member name	Since	Type	Description
			the API server (see the require command). The following errors indicate incorrect behavior on the part of the API (please contact AptoVision support): • INTERNAL_ERROR • NOT_IMPLEMENTED
message	2.0	String	A human-readable error message describing the error.

In addition to the error conditions documented for each command individually, the following errors apply to most commands:

INVALID_COMMAND	Entered command does not exist.
ILLEGAL_ARGUMENT	Command syntax or the values provided as arguments do not conform to the description in this user guide

Some commands include an array of device error objects as part of their return value (see individual command descriptions). This allow the API server to report that the command was successfully understood and succeeded for some but not all devices. A device error object (DeviceError) has the following structure:

```
{
      "device_id" : String,
      "reason" : String,
      "message" : String
}
```

Member name	Since	Type	Description
device_id	2.0	String	Device ID
reason	2.0	String	Machine-readable name of the error type: DEVICE_DISCONNECTED The device is disconnected from the network. DEVICE_TYPE The operation cannot be performed on this device because it does not support it. INVALID_OPERATION The operation cannot be performed on this device for a reason other than the device type or state. INVALID_STATE The operation cannot be performed on the device in its current state. The same command may succeed later if the condition is resolved. IO_ERROR The operation failed due to an input/output error (for example, disk read/write failure). REQUEST_TIMEOUT The device did not respond in time. RESOURCE_NOT_FOUND A resource (e.g. a firmware file) needed for this operation was not found.
message	2.0	String	Human-readable message describing the error

In addition to the device-specific error conditions documented for each individual command, the following errors apply to most commands:

DEVICE_DISCONNECTED	Occurs when the command issued targets a device which is currently disconnected from the network.
INVALID_STATE	Occurs when the command issued targets a device for which a firmware update is in currently progress.
REQUEST_TIMEOUT	Indicates that the device did not respond in time to the request. This can happen, for example, if network packets are dropped because of transient

network issues.

3 Usage Examples

3.1 Connection Setup

There are some commands that are typically used when initiating a connection to the BlueRiver Control Server when setting up the API session. Their use is demonstrated below.

First the application uses the `require` command to specify the API version it is expecting.

```
require blueriver_api 2.12.0.0
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

The version of the blueriver_api command module always matches the release version of the BlueRiver Control Server (API server).

In the above example, the application is declaring that it has been developed for the API server release 2.14.0.0. This also means that it requires API 2.14 or later to run. Refer section 1.2 Application Compatibility and API Version Numbering Scheme for details.

Then, if the application will be using the optional "multiview" functionality, it needs to use the `require` command again to load the multiview command module. Example of this command shown below.

```
require multiview 1.1.0
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

The multiview command module is described in section 6 Module *multiview* Version 1.1 Command Reference.

Then, for the application to enable independent routing of HDMI audio it issues the command `mode split_hdmi_audio`. This will acknowledge that application is a post-API 2.0 application that supports independent routing of HDMI audio. Refer to section 5.10.3 mode split_hdmi_audio for details.

```
mode split_hdmi_audio on
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

Finally, if the application will be handling events through asynchronous notifications, instead of polling for events using the `event` command (optional), it issues the `mode async` command.

A few event notifications appear immediately to inform the application about devices present on the network (typically `DEVICE_CONNECTED` and one `DEVICE_INFO_AVAILABLE` per device), as shown below.

```
mode async on
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
{
   "status" : "NOTIFICATION",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec08de23c",
            "event_id" : 4178,
            "event_type" : "DEVICE_CONNECTED",
            "timestamp" : 1499282658,
            "request_id" : 4281
         },
         {
            "device_id" : "d88039634377",
            "event_id" : 4179,
            "event_type" : "DEVICE_CONNECTED",
            "timestamp" : 1499282658,
            "request_id" : 4283
         },
         {
            "device_id" : "d88039634377",
            "event_id" : 4200,
            "event_type" : "DEVICE_INFO_AVAILABLE",
            "timestamp" : 1499282658,
            "request_id" : 4284
         },
         {
            "device_id" : "001ec08de23c",
            "event_id" : 4201,
            "event_type" : "DEVICE_INFO_AVAILABLE",
            "timestamp" : 1499282658,
            "request_id" : 4303
         }
      ]
   },
   "error" : null
}
```

3.2 Enabling and Disabling the Frame Rate Converter

3.2.1 Connection Setup

When connecting to BlueRiver Control Server (API) we ask for API 2.8 or later. This is the minimum version that supports the frame rate converter and enables asynchronous event notifications.

```
require blueriver_api 2.8.0
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
```

```
   "error" : null
}
mode async
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

3.2.2 Device Discovery and Feature Detection

Asynchronous event notifications are received that indicate two devices are present on the network (`DEVICE_CONNECTED` events) and that detailed information about these devices is available (`DEVICE_INFO_AVAILABLE` events).

```
{
   "status" : "NOTIFICATION",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "d8803962e132",
            "event_id" : 5,
            "event_type" : "DEVICE_CONNECTED",
            "timestamp" : 1489164040,
            "request_id" : 5
         },
         {
            "device_id" : "001ec0f03668",
            "event_id" : 6,
            "event_type" : "DEVICE_CONNECTED",
            "timestamp" : 1489164040,
            "request_id" : 7
         },
         {
            "device_id" : "d8803962e132",
            "event_id" : 7,
            "event_type" : "DEVICE_INFO_AVAILABLE",
            "timestamp" : 1489164040,
            "request_id" : 6
         },
         {
            "device_id" : "001ec0f03668",
            "event_id" : 8,
            "event_type" : "DEVICE_INFO_AVAILABLE",
            "timestamp" : 1489164040,
            "request_id" : 8
         }
      ]
   },
   "error" : null
}
```

BlueRiver API retrieves detailed information about the first device. This information associated with the device's `DEVICE_INFO_AVAILABLE` event and is obtained using the `request` command. This same information can also be retrieved later using the `get` device command.

```
request 6
{
   "status" : "SUCCESS",
   "request_id" : 6,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "identity" : {
               "engine" : "PLETHORA",
               "firmware_version" : "3.2.1",
               "firmware_rc" : "6",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "configuration" : {
               "device_name" : "FIRST_DEVICE"
            },
            "status" : {
               "active" : true,
               "point_to_point" : false,
               "temperature" : 79
            },
            "streams" : [
               {
                  "type" : "HDMI",
                  "index" : 0,
                  "configuration" : {
                     "address" : "224.1.2.42",
                     "enable" : false,
                     "source" : {
                        "value" : 0,
                        "choices" : [
                           {
                              "value" : 0,
                              "description" : "HDMI video input",
                              "details" : {
                                 "ref_class" : "NODE",
                                 "ref_type" : "HDMI_DECODER",
                                 "ref_index" : 0
                              }
                           },
                           {
                              "value" : 1,
                              "description" : "Frame rate converter",
                              "details" : {
                                 "ref_class" : "NODE",
                                 "ref_type" : "FRAME_RATE_CONVERTER",
                                 "ref_index" : 0
                              }
                           }
                        ]
                     }
```

```
            },
            "status" : {
               "state" : "STOPPED"
            },
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "HDMI_DECODER",
               "ref_index" : 0
            }
         },
         {
            "type" : "HDMI",
            "index" : 1,
            "configuration" : {
               "address" : "224.1.2.44",
               "enable" : false
            },
            "status" : {
               "state" : "STOPPED"
            },
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "SCALER",
               "ref_index" : 0
            }
         },
         (...)
      ],
      "subscriptions" : [
         (...)
      ],
      "nodes" : [
         (...)
         {
            "type" : "FRAME_RATE_CONVERTER",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "divide_factor" : 2
            },
            "inputs" : [
               {
                  "name" : "main",
                  "index" : 0,
                  "configuration" : {},
                  "status" : {
                     "source" : {
                        "ref_class" : "NODE",
                        "ref_type" : "HDMI_DECODER",
                        "ref_index" : 0
                     }
                  }
               }
            ]
         },
```

```
            {
               "type" : "HDMI_DECODER",
               "index" : 0,
               "configuration" : {
                  "edid" :
"00ffffffffffff004c2d5706333244590a140103801009782aee91a3544c99260f5054230
800a9408180814081009500b30001010101023a801871382d40582c4500a05a0000001e011
d007251d01e206e285500a05a0000001e000000fd00323c1b5111000a20202020202000000
0fc0053796e634d61737465720a2020013b02031cf34890041f05141303122309070783010
00066030c00100080011d80d0721c1620102c2580a05a0000009e011d8018711c1620582c2
500a05a0000009e011d00bc52d01e20b8285540a05a0000001e011d007251d01e206e28550
0a05a0000001e8c0ad090204031200c405500a05a000000180000000000000000000075",
                  "hdcp_support_enable" : true,
                  "hdcp_22_support_disable" : false
               },
               "status" : {
                  "video" : {
                     "width" : 0,
                     "height" : 0,
                     "frames_per_second" : 0,
                     "color_space" : "RGB",
                     "bits_per_pixel" : 8,
                     "scan_mode" : "PROGRESSIVE"
                  },
                  "has_video_details" : true,
                  "video_details" : {
                     "pixel_clock" : 0,
                     "total_width" : 0,
                     "total_height" : 0,
                     "hsync_width" : 0,
                     "hsync_front_porch" : 0,
                     "hsync_negative" : false,
                     "vsync_width" : 0,
                     "vsync_front_porch" : 0,
                     "vsync_negative" : false,
                     "vic" : 0,
                     "has_hdmi_vic" : false,
                     "hdmi_vic" : 0,
                     "picture_aspect" : "ASPECT_NO_DATA",
                     "has_active_format" : false,
                     "active_format" : 0,
                     "it_content_type" : "NO_DATA",
                     "scan_information" : "NO_DATA",
                     "colorimetry" : "RGB",
                     "rgb_range" : "DEFAULT",
                     "ycc_range" : "LIMITED"
                  },
                  "hdcp_protected" : false,
                  "hdcp_version" : "NONE",
                  "hdmi_2_0_support" : true,
                  "source_stable" : false
               },
               "inputs" : [
                  {
```

```
                        "name" : "main",
                        "index" : 0,
                        "configuration" : {
                           "source" : {
                              "value" : 0,
                              "choices" : [
                                 {
                                    "value" : 0,
                                    "description" : "HDMI Input 1",
                                    "details" : {
                                       "ref_class" : "NODE",
                                       "ref_type" : "VIDEO_INPUT",
                                       "ref_index" : 0
                                    }
                                 },
                                 {
                                    "value" : 1,
                                    "description" : "Display Port Input
1",
                                    "details" : {
                                       "ref_class" : "NODE",
                                       "ref_type" : "VIDEO_INPUT",
                                       "ref_index" : 1
                                    }
                                 },
                                 {
                                    "value" : 32,
                                    "description" : "Automatic Input
Selection (HDMI Input 1)",
                                    "details" : {
                                       "ref_class" : "AUTOMATIC",
                                       "ref_type" : null,
                                       "ref_index" : 0
                                    }
                                 }
                              ]
                           }
                        },
                        "status" : {
                           "source" : {
                              "ref_class" : "NODE",
                              "ref_type" : "VIDEO_INPUT",
                              "ref_index" : 0
                           }
                        }
                     }
                  ]
               },
               (...)
               {
                  "type" : "SCALER",
                  "index" : 0,
                  "configuration" : {
                     "width" : 0,
                     "height" : 0
```

```
                    },
                    "status" : {
                       "video" : {
                          "width" : 0,
                          "height" : 0,
                          "frames_per_second" : 0,
                          "color_space" : "RGB",
                          "bits_per_pixel" : 8,
                          "scan_mode" : "PROGRESSIVE"
                       }
                    },
                    "inputs" : [
                       {
                          "name" : "main",
                          "index" : 0,
                          "configuration" : {
                             "source" : {
                                "value" : 0,
                                "choices" : [
                                   {
                                      "value" : 0,
                                      "description" : "HDMI video input",
                                      "details" : {
                                         "ref_class" : "NODE",
                                         "ref_type" : "HDMI_DECODER",
                                         "ref_index" : 0
                                      }
                                   },
                                   {
                                      "value" : 1,
                                      "description" : "Frame rate
converter",
                                      "details" : {
                                         "ref_class" : "NODE",
                                         "ref_type" :
"FRAME_RATE_CONVERTER",
                                         "ref_index" : 0
                                      }
                                   }
                                ]
                             }
                          },
                          "status" : {
                             "source" : {
                                "ref_class" : "NODE",
                                "ref_type" : "HDMI_DECODER",
                                "ref_index" : 0
                             }
                          }
                       }
                    ]
                 },
                 (...)
              ]
           }
        ],
```

```
      "error" : []
   },
   "error" : null
}
```

The information provided includes the following data of interest:

通过is_transmitter字段的真假判断设备是否是发送器

- Device type is confirmed as a transmitter device (`is_transmitter` is `true` in the identity section and the device has at least one HDMI stream).
- The device has two HDMI streams:
  - HDMI 0 (native resolution stream); and
  - HDMI 1 (scaled resolution stream).
- The configuration section of HDMI 0 stream shows two possible choices for the source of this stream:
  - The choice with value 0 "HDMI Video Input" for video coming directly from the HDMI decoder (`node HDMI_DECODER 0`).
  - The choice with value 1 "Frame Rate Converter" for video that comes from the frame rate converter (`node FRAME_RATE_CONVERTER 0`).
- The configuration section of the HDMI 0 stream also shows the currently selected value is `0`, indicating the device is currently sending the original video. This is confirmed by the source section of the stream which shows the video source is the HDMI decoder (`node HDMI_DECODER 0`).
- The source section of the HDMI 1 stream indicates that the video comes from the video scaler (`SCALER node 0`). The configuration section of this same stream shows no option to configure this.
- The video scaler node (`SCALER 0`) has a single input (`main 0`). Like stream HDMI 0, the configuration section of this input shows two possible source choices:
  - The choice with value 0 "HDMI Video Input" for video coming directly from the HDMI decoder (`node HDMI_DECODER 0`).
  - The choice with value 1 "Frame Rate Converter" for video that comes from the frame rate converter (`node FRAME_RATE_CONVERTER 0`).
- Again, like stream HDMI 0, the configuration and status sections of this input show the original video to be the current source.

Information retrieved for the second device.

```
request 8
{
   "status" : "SUCCESS",
   "request_id" : 8,
   "result" : {
      "devices" : [
         {
            "device_id" : "001ec0f03668",
            "identity" : {
               "engine" : "BLUERIVER_NT",
               "firmware_version" : "2.12.1",
               "firmware_rc" : "1",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "configuration" : {
               "device_name" : "SECOND_DEVICE"
            },
            "status" : {
               "active" : true,
               "point_to_point" : false,
               "temperature" : 71
```

```
            },
            "streams" : [
               {
                  "type" : "HDMI",
                  "index" : 0,
                  "configuration" : {
                     "address" : "224.1.2.43",
                     "enable" : true
                  },
                  "status" : {
                     "state" : "STREAMING"
                  },
                  "source" : {
                     "ref_class" : "NODE",
                     "ref_type" : "HDMI_DECODER",
                     "ref_index" : 0
                  }
               },
               (...)
            ],
            "subscriptions" : [
               (...)
            ],
            "nodes" : [
               (...)
               {
                  "type" : "HDMI_DECODER",
                  "index" : 0,
                  "configuration" : {
                     "edid" :
"00ffffffffffff004c2d5706333244590a140103801009782aee91a3544c99260f5054230
800a9408180814081009500b30001010101023a801871382d40582c4500a05a0000001e011
d007251d01e206e285500a05a0000001e000000fd00323c1b5111000a20202020202000000
0fc0053796e634d61737465720a2020013b02031cf34890041f05141303122309070783010
00066030c00100080011d80d0721c1620102c2580a05a0000009e011d8018711c1620582c2
500a05a0000009e011d00bc52d01e20b8285540a05a0000001e011d007251d01e206e28550
0a05a0000001e8c0ad090204031200c405500a05a000000180000000000000000000075",
                     "hdcp_support_enable" : true,
                     "hdcp_22_support_disable" : false
                  },
                  "status" : {
                     "video" : {
                        "width" : 1920,
                        "height" : 1080,
                        "frames_per_second" : 60,
                        "color_space" : "RGB",
                        "bits_per_pixel" : 8,
                        "scan_mode" : "PROGRESSIVE"
                     },
                     "has_video_details" : false,
                     "video_details" : {},
                     "hdcp_protected" : false,
                     "hdcp_version" : "NONE",
                     "hdmi_2_0_support" : false,
                     "source_stable" : true
                  },
```

```
                "inputs" : [
                   {
                      "name" : "main",
                      "index" : 0,
                      "configuration" : {},
                      "status" : {
                         "source" : {
                            "ref_class" : "NODE",
                            "ref_type" : "VIDEO_INPUT",
                            "ref_index" : 0
                         }
                      }
                   }
                ]
             },
             (...)
          ]
       }
    ],
    "error" : []
 },
 "error" : null
}
```

This device is also a transmitter device. However, it has more limited capabilities than the first device example. The first thing to note is that it does not have a second HDMI stream (or a scaler node), meaning the scaled resolution stream is not supported on this device.

The second thing to note is that the configuration section of the HDMI 0 stream does not have a source member, nor does the device have a frame rate converter node. This indicates that the device does not support frame rate conversion.

In this specific case, the reason the frame rate converter is not supported is that the device is a BlueRiver NT device (instead of a BlueRiver NT2000 device).

3.2.3 Enabling Frame Rate Conversion on Original Resolution Video Stream

The `set property` command is used to select the frame rate converted video instead of the original video on the original resolution stream (HDMI 0). Example uses the group name `ALL_TX` as the target, meaning it is requesting the configuration change be applied to both devices. The command runs asynchronously, `PROCESSING` status is returned and has request ID `9`.

```
set ALL_TX property streams[HDMI:0].configuration.source.value 1
{
   "status" : "PROCESSING",
   "request_id" : 9,
   "result" : null,
   "error" : null
}
```

A `SETTINGS_CHANGED` event is received from the API to indicate that the first device's settings have been modified. If other API clients are connected to the BlueRiver Control Server (API), they will also receive this event, which allows them to stay up to date with the device's settings.

```
{
   "status" : "NOTIFICATION",
```

```
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "d8803962e132",
            "event_id" : 9,
            "event_type" : "SETTINGS_CHANGED",
            "timestamp" : 1489164095,
            "request_id" : 10
         }
      ]
   },
   "error" : null
}
```

Once the execution of the command is completed, a `REQUEST_COMPLETE` event with request ID `9` (request ID associated with above command) is received. This ID is used with the `request` command to retrieve the result.

```
request 9
{
   "status" : "SUCCESS",
   "request_id" : 9,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "identity" : {
               "engine" : "PLETHORA",
               "firmware_version" : "3.2.1",
               "firmware_rc" : "6",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "configuration" : {
               "device_name" : "FIRST_DEVICE"
            },
            "status" : {
               "active" : true,
               "point_to_point" : false
            },
            "streams" : [
               {
                  "type" : "HDMI",
                  "index" : 0,
                  "configuration" : {
                     "address" : "224.1.2.42",
                     "enable" : false,
                     "source" : {
                        "value" : 1
                     }
                  },
                  "status" : {
                     "state" : "STOPPED"
                  },
                  "source" : {
                     "ref_class" : "NODE",
                     "ref_type" : "FRAME_RATE_CONVERTER",
```

```
                        "ref_index" : 0
                     }
                  },
                  (...)
               ],
               "subscriptions" : [
                  (...)
               ],
               "nodes" : [
                  (...)
               ]
            }
         ],
         "error" : [
            {
               "device_id" : "001ec0f03668",
               "reason" : "DEVICE_TYPE",
               "message" : "Operation cannot be performed on this
device type"
            }
         ]
      },
      "error" : null
}
```

The new settings of the first device are in the devices array of the result which indicates the request succeeded for this device. The value in the configuration section of stream HDMI 0 as well as the source section of that same stream both show that the video source has been modified as expected.

For the second device which does not support frame rate conversion, there is an entry in the error array of the result with a reason of DEVICE_TYPE which indicates that the request failed for the second device because it does not support this feature.

3.2.4 Disabling Frame Rate Conversion on Original Resolution Video Stream

Frame rate conversion can be disabled using the `set property` command again.

```
set d8803962e132 property streams[HDMI:0].configuration.source.value 0
      {
         "status" : "PROCESSING",
         "request_id" : 37,
         "result" : null,
         "error" : null
      }
```

3.2.5 Enabling Frame Rate Conversion on Scaled Resolution Video Stream

In a similar way, the `set property` command can be used to select the frame rate converted video as the source of the video scaler.

```
      set d8803962e132 property
      nodes[SCALER:0].inputs[main:0].configuration.source.value 1
      {
         "status" : "PROCESSING",
         "request_id" : 39,
         "result" : null,
```

```
   "error" : null
}
{
   "status" : "NOTIFICATION",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : null,
            "event_id" : 40,
            "event_type" : "REQUEST_COMPLETE",
            "timestamp" : 1489090702,
            "request_id" : 39
         }
      ]
   },
   "error" : null
}
```

Again, retrieving the result with the request command shows the changes have been applied as expected.

```
request 39
{
   "status" : "SUCCESS",
   "request_id" : 39,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "identity" : {
               "engine" : "PLETHORA",
               "firmware_version" : "3.2.1",
               "firmware_rc" : "6",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "configuration" : {
               "device_name" : "FIRST_DEVICE"
            },
            "status" : {
               "active" : true,
               "point_to_point" : false
            },
            "streams" : [
               (...)
            ],
            "subscriptions" : [
               (...)
            ],
            "nodes" : [
               (...)
               {
                  "type" : "SCALER",
                  "index" : 0,
                  "configuration" : {
                     "width" : 0,
                     "height" : 0
                  },
```

```
                        "status" : {
                           "video" : {
                              "width" : 0,
                              "height" : 0,
                              "frames_per_second" : 0,
                              "color_space" : "RGB",
                              "bits_per_pixel" : 8,
                              "scan_mode" : "PROGRESSIVE"
                           }
                        },
                        "inputs" : [
                           {
                              "name" : "main",
                              "index" : 0,
                              "configuration" : {
                                 "source" : {
                                    "value" : 1
                                 }
                              },
                              "status" : {
                                 "source" : {
                                    "ref_class" : "NODE",
                                    "ref_type" : "FRAME_RATE_CONVERTER",
                                    "ref_index" : 0
                                 }
                              }
                           }
                        ]
                     },
                     (...)
                  ]
               }
            ],
            "error" : []
         },
         "error" : null
      }
```

3.2.6 Disabling Frame Rate Conversion on Scaled Resolution Video Stream

Once again, the original video can be selected back by using the `set property` command.

```
set d8803962e132 property
nodes[SCALER:0].inputs[main:0].configuration.source.value 0
{
   "status" : "PROCESSING",
   "request_id" : 41,
   "result" : null,
   "error" : null
}
```

4 Core Command Reference

4.1 Command mode

The `mode` command provides sub-commands which are used to modify the behaviour of the API. Its effect is limited to a single API session.

4.1.1 mode human

4.1.1.1 Command usage

```
mode human [(on|off)]
```

4.1.1.2 Description

Enable or disable human-readable output mode.

When human-readable mode is enabled, JSON responses returned by the API are indented and formatted to be visually inspected with ease.

When it is disabled, JSON output contains no newline, space or other white space character.

At the start of an API session, human-readable mode is disabled by default.

4.1.1.3 Arguments

If neither `on` or `off` is specified, the default is `on`.

4.1.1.4 Return Value

```
null
```

4.1.1.5 Errors

`ILLEGAL_ARGUMENT` occurs when a value other than on or off is specified.

4.1.1.6 Example

```
mode human
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}
```

4.2 Command quit

4.2.1 quit

4.2.1.1 Command usage

```
quit
```

4.2.1.2 Description

End the current API session and close the connection.

4.2.1.3 Arguments

None

4.2.1.4 Notes

This command was introduced in version 2.5 of the API.

Use of this command is optional: the client can simply close the TCP connection instead.

4.2.1.5 Return Value

None

4.2.1.6 Errors

`INVALID_COMMAND` Indicates the API server supports only a version of the BlueRiver API prior to 2.5.

4.2.1.7 Example

```
quit
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

4.3 Command require

4.3.1 require

4.3.1.1 Command usage

```
require module_name version
```

4.3.1.2 Description

设置使用的命令模块和版本

Determines, whether the requested version of a given command module is available in the current BlueRiver Control Server and loads the specified module if it is.

Compatibility of the requested version with the version included in the API server is determined according to the rules specified in section 1.2 Application Compatibility and API Version Numbering Scheme.

This reference guide specifies two command modules:

blueriver_api 2.14:	Refer to section 5 Module *blueriver_api* Version 2.14 Command Reference
multiview 1.1:	Refer to section 6 Module *multiview* Version 1.1 Command Reference.

4.3.1.3 Arguments

The ***module_name*** argument is the name of the requested module. Currently, the only supported module name is blueriver_api. Other feature names are reserved for future extensions.

The ***version*** argument is the requested version number.

4.3.1.4 Return Value

```
null
```

4.3.1.5 Errors

`VERSION` returned when the API version is not compatible with the given version of requested module.

`UNSUPPORTED_FEATURE` returned when the requested module is not supported by the API version.

4.3.1.6 Example

The following examples assumes current version is 2.11.0.0

```
require blueriver_api 2.10.0
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}

require blueriver_api 2.11.0.0
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}

require blueriver_api 2.11.4.0
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}

require blueriver_api 2.13.0.0
{
      "status" : "ERROR",
      "request_id" : null,
      "result" : null,
      "error" : {
            "reason" : "VERSION",
            "message" : "Incompatible API version"
      }
}
```

4.4 Command version

4.4.1 version

4.4.1.1 Command usage

```
version
```

4.4.1.2 Description

得到控制服务器支持的模块和版本

Get version and other information about the BlueRiver Control Server (API) and available command modules.

4.4.1.3 Arguments

None

4.4.1.4 Notes

This command was introduced in version 2.4 of the API.

Support for a fourth digit in the version number was added starting with API version 2.11.

4.4.1.5 Return Value

```
{
      "server" : String,
      "vendor" : String,
      "version" : String,
      "modules" : [ModuleDescriptionObject, ...]
}
```

Member name	Since	Type	Description
server	2.4	String	The name of the API server software
vendor	2.4	String	The vendor of the API server software
version	2.4	String	The release version of the API server software
modules	2.4	Array of ModuleDescriptionObject	A list of available command modules with their version (refer to section 4.3 Command require).

The `ModuleDescriptionObject` JSON object has the following structure:

```
{
      "name" : String,
      "version" : String
}
```

Member name	Since	Type	Description
Name	2.4	String	The name of the command module
Version	2.4	String	The version of the command module

4.4.1.6 Errors

`INVALID_COMMAND` Indicates that the API server supports a version of the API before 2.4.

4.4.1.7 Example

```
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "server" : "BlueRiver Control Server",
      "vendor" : "AptoVision",
      "version" : "2.4.0",
      "modules" : [
         {
            "name" : "blueriver_api",
            "version" : "2.4.0"
         }
      ]
   },
   "error" : null
}
```

5 Module *blueriver_api* Version 2.14 Command Reference

5.1 Introduction

This section describes the blueriver_api command module version 2.14. The blueriver_api command module contains most of the commands for controlling and querying the status of transmitter and receiver devices.

Note: Before the commands described in this section can be called, the command module must be loaded with the `require` command:

```
require blueriver_api 2.14.0.0
```

Refer to section 4.3 Command require for detail.

5.2 Specifying a Video Output Format

The video output format of a receiver device can be set in one of the following ways:

- Using the **`set video`** command, which is generally the recommended command to use (refer to section5.15.6 set video).
- Using the **`join`** command. When joining a video stream, the video output format can optionally be specified, which allows both operations (joining and setting video output format) to be performed with a single command (refer to section 5.7.1 join).
- Using the **`set multiview`** command to set the device in multiview mode with a specified multiview layout (see section 6.6.1 set multiview).

5.2.1 Video Display Mode Argument

The `set` **`video`** and **`join`** commands (when a video format is specified, which is optional for the `join` command) require a display mode argument.

This argument is one of the following keywords:

- **genlock** for genlocked (i.e. zero-frame latency) full screen 1:1 display mode.
- **genlock_scaling** for genlocked (i.e. zero-frame latency) full screen mode with scaling.
- **fastswitch** for fast switched full screen mode.
- **wall** for display wall mode (genlocked display wall for devices that support it and legacy fast switched wall otherwise).
- **multiview** for multiview mode.

In fast switched full screen mode, if the aspect ratio of the video source and the output resolution are not the same, the default behaviour is to add black bars on the sides (i.e. pillarbox) or at the top and bottom (i.e. letterbox) of the image to preserve the aspect ratio.

An alternate behaviour can be specified by adding one of the following keywords after the **fastswitch** keyword:

- If the **crop** keyword is specified, the image is cropped to preserve the aspect ratio.
- If the **stretch** keyword is specified, the image is stretched to cover the entire screen without regard for the aspect ratio.

Comment: The **set multiview** command implicitly puts the device in multiview mode and as such does not require a display mode argument.

5.2.2 Video Format Arguments

The video format arguments are a list of arguments introduced by keywords. As arguments are recognized by the keyword that precedes them, the ordering is not important. Each argument must be specified at most once.

Unless otherwise stated, arguments apply to all display modes except genlock, which does not allow any video format arguments. Also unless otherwise stated, all arguments are individually optional, but a minimum set of arguments must be specified to uniquely identify a video format (refer to section 5.2.3 Uniquely Identifying a Video Format).

5.2.2.1 Picture Aspect Ratio图像行宽比

`aspect` *`aspect_ratio`*

The **aspect_ratio** argument specifies the picture aspect ratio. There are two valid formats for this argument: either the aspect ratio can be specified directly as a ratio (e.g. 4:3 or 16:9) or one of the following may be used:

可设置值
- `ASPECT_NO_DATA` if the aspect ratio is to be left unspecified.
- `ASPECT_4_3` for a 4:3 aspect ratio.
- `ASPECT_16_9` for a 16:9 aspect ratio.
- `ASPECT_256_135` for a 256:135 aspect ratio.
- `ASPECT_64_27` for a 64:27 aspect ratio.

5.2.2.2 Frame Rate 帧率

`fps` *`fps`*

The `fps` argument specifies the frame rate in frames per second. It can be either an integer (e.g. 50) or a real number (e.g. 50.000) and can be optionally followed by the letter "`m`" (e.g. 60m or 30.00m) to indicate that the frame rate must be divided by 1.001 (i.e. multiplied by 1000/1001).

In genlock scaling display mode and in genlock wall mode (for devices that support it), the specified frame rate must match the frame rate of the video source.

5.2.2.3 Video Sync Front Porch

`front_porch` *`hsync_front_porch vsync_front_porch`*

The `hsync_front_porch` and `vsync_front_porch` specify respectively the horizontal and vertical front porch in pixels. If the front porch is specified, the pulse width must also be specified (refer to section 5.2.2.8Video Sync Pulse Width below).

5.2.2.4 HDMI Video Identification Code (HDMI_VIC)

`hdmi_vic` *`hdmi_vic`*

The `hdmi_vic` argument specifies the HDMI Video Identification Code (HDMI_VIC).

The following values are supported:

HDMI Video Identification Code	Video Format
1	3840x2160 (2160p) 29.97Hz or 30Hz
2	3840x2160 (2160p) 25Hz
3	3840x2160 (2160p) 24Hz or 23.98Hz
4	4096x2160 24Hz or 23.98Hz (256:135)

Other values will be inserted in the HDMI Vendor-Specific InfoFrame as requested but cannot be used to uniquely identify a video format (refer to section 5.2.3 Uniquely Identifying a Video Format for details).

5.2.2.5 Keep Region Size

`keep keep_width keep_height`

The `keep_width` and `keep_height` specify respectively the width and the height of the portion of the source video that is displayed (i.e. "kept") in pixels.

The `keep_width` argument must be even (i.e. a multiple of two).

The keep argument is specific to the wall display mode, and is typically specified along with the read offset argument (see section 5.2.2.6 Read Offset for more information). This argument must be specified for the wall display mode and must not be specified for other display modes.

5.2.2.6 Read Offset

`offset horiz_offset vert_offset`

The `horiz_offset` and `vert_offset` arguments specify respectively the horizontal and vertical offset of the displayed video within the full video source, in pixels.

The offset argument is specific to the wall display mode, and is typically specified along with the keep region size argument (refer to section 5.2.2.5 Keep Region Size for more information). This argument must be specified for the wall display mode and must not be specified for other display modes.

5.2.2.7 Video Sync Polarity

`polarity horiz_polarity vert_polarity`

The `horiz_polarity` and `vert_polarity` arguments specify respectively the polarity of the horizontal and vertical synchronization pulse, respectively.

Each may be one of the following:

`POSITIVE` for a positive sync polarity.
`NEGATIVE` for a negative sync polarity.

5.2.2.8 Video Sync Pulse Width

`pulse hsync_pulse vsync_pulse`

The `hsync_pulse` and `vsync_pulse` arguments specify respectively the width of the horizontal and vertical synchronization pulse, respectively. If the pulse width is specified, the front porch must also be specified (refer to section 5.2.2.3 Video Sync Front Porch for more information).

5.2.2.9 Video Resolution

`size width height`

The `width` and `height` arguments specify respectively the width and the height of the displayed video, in pixels.

The `width` argument must be even (i.e. a multiple of two).

This argument is optional in the multiview display mode. If left unspecified, the video resolution is set to be identical to the multiview layout size.

5.2.2.10 Total Size

`total total_width total_height`

The `total_width` and `total_height` arguments specify respectively the total width and height in pixels, including the blanking's.

5.2.2.11 Video Identification Code (VIC)

`vic vic`

The `vic` argument specifies the Video Identification Code (VIC). The following values are supported:

Video Identification Code	Video Format
1	640x480 60Hz (4:3)
2	720x480 59.94Hz or 60 Hz (4:3)
3	720x480 59.94Hz or 60 Hz (16:9)
4	1280x720 (720p) 59.94Hz or 60 Hz
16	1920x1080 (1080p) 59.94Hz or 60 Hz
17	720x576 50Hz (4:3)
18	720x576 50Hz (16:9)
19	1280x720 (720p) 50Hz
31	1920x1080 (1080p) 50Hz
32	1920x1080 (1080p) 24Hz or 23.98Hz
33	1920x1080 (1080p) 25Hz
34	1920x1080 (1080p) 29.97Hz or 30Hz
60	1280x720 (720p) 24Hz or 23.98Hz
61	1280x720 (720p) 25Hz
62	1280x720 (720p) 29.97Hz or 30Hz
93*	3840x2160 (2160p) 24Hz or 23.98Hz
94*	3840x2160 (2160p) 25Hz
95*	3840x2160 (2160p) 29.97Hz or 30Hz
96	3840x2160 (2160p) 50Hz
97	3840x2160 (2160p) 59.94Hz or 60 Hz
98*	4096x2160 24Hz or 23.98Hz (256:135)
99	4096x2160 25Hz (256:135)
100	4096x2160 29.97Hz or 30Hz (256:135)
101	4096x2160 50Hz (256:135)
102	4096x2160 59.94Hz or 60 Hz (256:135)

* Video Identification Codes (VIC) with an asterisk can alternatively be specified by their HDMI Video Identification Codes (HDMI_VIC). Refer to section 5.2.2.4 HDMI Video Identification Code (HDMI_VIC) for more information.

Values not in the table above will be inserted in the Auxiliary Video Information (AVI) InfoFrame as requested but cannot be used to uniquely identify a video format (refer to section5.2.3 Uniquely Identifying a Video Format).

5.2.2.12 Viewport

`viewport` *`viewport_horiz viewport_vert viewport_width viewport_height`*

The viewport is the rectangular region on screen where video is rendered. Areas on screen outside this region are black.

The `viewport_horiz` and `viewport_vert` arguments specify respectively the horizontal and vertical position of the viewport's top-left corner on the screen, in pixels.

The `viewport_width` and `viewport_height` arguments specify respectively the width and height of the viewport, in pixels.

The `viewport_ horiz` and `viewport_width` arguments must be even (i.e. a multiple of two). The viewport must be completely contained within the screen size (refer to section 5.2.2.9 Video Resolution for more information).

This argument is specific to the wall display mode. If left unspecified, the viewport covers the whole screen. This argument must not be specified for other display modes other than wall.

5.2.2.13 Colour Quantization Range

`quantization` *`quantization_range`*

The `quantization_range` argument specifies the colour quantization range of the output and is one of the following:

- STANDARD Appropriate quantization range for the video format as defined by the CEA-861-F standard.
- AUTO Same as STANDARD, but explicitly state FULL or LIMITED as appropriate instead of DEFAULT in the AVI InfoFrame. This can be used to work around some monitor issues.
- LIMITED Force limited quantization range.
- FULL Force full quantization range.

If this argument is omitted, the behaviour is the same as if STANDARD had been specified.

5.2.3 Uniquely Identifying a Video Format

A minimum set of video format arguments are required to uniquely identify a video format. This section describes the various options for doing so.

5.2.3.1 By Video Identification Code (VIC or HDMI_VIC)

If the desired video format has a supported Video Identification Code (VIC) or HDMI Video Identification Code (HDMI_VIC), it is sufficient to only specify this information. Other parameters (such as video timings, etc.) are automatically retrieved from a table.

For some video formats, there is an ambiguity regarding the frame rate, which is resolved as follow:

- For formats that have a frame rate of 24Hz or 23.98Hz (24Hz/1.001), the 24Hz frame rate is selected.
- For formats that have a frame rate of 30Hz or 29.97Hz (30Hz/1.001), the 29.97Hz frame rate is selected.
- For formats that have a frame rate of 60Hz or 59.94Hz (60Hz/1.001), the 59.94Hz frame rate is selected.

Refer to sections 5.2.2.11 Video Identification Code (VIC) and 5.2.2.4 HDMI Video Identification Code (HDMI_VIC) for a list of supported Video Identification Codes.

5.2.3.2 By Video Identification Code (VIC or HDMI_VIC) and Frame Rate

Same as by Video Identification Code (VIC or HDMI_VIC) as above but the frame rate ambiguity is resolved by specifying the frame rate explicitly.

5.2.3.3 By Video Resolution and Frame Rate

If the desired video format has a supported Video Identification Code (VIC) or HDMI Video Identification Code (HDMI_VIC), it is sufficient to only specify the video resolution and frame rate.

5.2.3.4 Completely Specified Video Formats

Finally, if the video format does **not** have a supported Video Identification Code (VIC) or HDMI Video Identification Code (HDMI_VIC), all video arguments must be specified to completely describe the video format, this includes the following:

- Picture aspect ratio if applicable (section 5.2.2.1).
- Frame Rate (section 5.2.2.2).
- Video Sync Front Porch (section 5.2.2.3).
- Video Sync Polarity (section 5.2.2.7).
- Video Sync Pulse Width (section 5.2.2.8).
- Video Resolution (section 5.2.2.9).
- Total Size (section 5.2.2.10).

Video Identification Code (VIC or HDMI_VIC) if applicable. Refer to sections5.2.2.4 HDMI Video Identification Code (HDMI_VIC) and 5.2.2.11 Video Identification Code (VIC) for further information.

5.3 Command event

5.3.1 event

5.3.1.1 Command usage

`event [`*`last_event_id`*`]`

5.3.1.2 Description

Get a list of events that occurred recently.

5.3.1.3 Arguments

Event ID of the last event you already retrieved. The command lists all events that occurred after that event. If left blank, all unexpired events since the beginning of the API session are returned.

5.3.1.4 Notes

As an alternative to using the `event` command, a client can be notified asynchronously of new events by enabling asynchronous event notification using the `mode async` command.

Refer to section 5.10.1 mode async for details.

When a new client connects to the BlueRiver Control Server (API), the following events are triggered (for the new client only) to update the client on the current state of the network:

A `DEVICE_CONNECTED` event for any device present on the network.
A `DEVICE_DISCONNECTED` event for every device that has previously been seen by the API server but that is no longer present on the network.
A `FIRMWARE_UPDATE_START` event for any device that is being updated.

Comment: Events expire (are removed from the event list) 10 minutes after they occur.

5.3.1.5 Return Value

```
{
"events" : [EventObject ...]
```

```
}
```

i.e. an object with a single member named "events" which is an array of event objects. An event object (EventObject) has the following structure:

```
{
        "device_id" : String
        "event_id" : Integer,
        "event_type" : String,
        "timestamp" : Integer,
        "request_id" : Integer
}
```

Member name	Since	Type	Description
device_id	2.0	String	The ID of the device for which an event occurred. **Note:** This member is null for **REQUEST_COMPLETE** events since background command requests can target multiple devices.
event_id	2.0	Integer	The ID of the event.
event_type	2.0	String	The type of event, which is one of the following: `DEVICE_CONNECTED` A device has joined the network. `DEVICE_DISCONNECTED` A device has left the network. `DEVICE_INFO_AVAILABLE` Initial device information is available for a newly connected device. `FIRMWARE_UPDATE_COMPLETE` A firmware update of a device has completed. `FIRMWARE_UPDATE_PROGRESS` Firmware update of a device has progressed. `FIRMWARE_UPDATE_START` A firmware update of a device has been started. No operation can be performed on that device until the firmware update completes. `I2C` An I2C bus master (e.g. a MCU) has generated an event through the I2C interface. `INFRARED_RECEIVED` Infrared data has been received. `REQUEST_COMPLETE` A background command request has completed. `RS232_RECEIVED` RS-232 data has been received. `SETTINGS_CHANGED` The settings of a device changed. As new versions of the API can introduce new event types, your application should accept and ignore other event types to ensure compatibility with future versions.
timestamp	2.0	Integer	Time at which the event occurred. The precision of this time stamp is in seconds and its origin (time 0) is unspecified.
request_id	2.0	Integer	Request ID associated with the event, or null if not applicable. See section 5.13 Command request. `DEVICE_CONNECTED` and `DEVICE_DISCONNECTED`: Request ID that can be used to retrieve basic device information. `DEVICE_INFO_AVAILABLE` and `SETTINGS_CHANGED`:

Member name	Since	Type	Description
			Request ID that can be used to retrieve available data `I2C`: Request ID that can be used identify the source of the event `INFRARED_RECEIVED` and `RS232_RECEIVED`: Request ID that can be used to retrieve the received data. `REQUEST_COMPLETE`: Request ID of the request that just completed.

5.3.1.6 Errors

None

5.3.1.7 Example

```
event 49
{
	"status" : "SUCCESS",
	"request_id" : null,
	"result" : {
		"events" : [{
			"device_id" : "d8803962abcd",
			"event_id" : 50,
			"event_type" : "SETTINGS_CHANGED",
			"timestamp" : 1450386633,
			"request_id" : 29
		},
		{
			"device_id" : "d8803962b2d9",
			"event_id" : 51,
			"event_type" : "DEVICE_CONNECTED",
			"timestamp" : 1450386633,
			"request_id" : null
		},
		{
			"device_id" : "d8803962b2d9",
			"event_id" : 52,
			"event_type" : "DEVICE_INFO_AVAILABLE",
			"timestamp" : 1450386633,
			"request_id" : 30
		}]
	},
	"error" : null
}
```

5.4 Command factory

5.4.1 factory

5.4.1.1 Command usage

`factory` ***`target`***

5.4.1.2 Description

Restore the factory default settings of a single device or a group of devices.

5.4.1.3 Arguments

The **`target`** argument must be either the device ID of a single device or the name of a device group.

The allowed device groups are the following:

`ALL`: All devices
`ALL_TX`: All transmitter devices
`ALL_RX`: All receiver devices

5.4.1.4 Return Value

The return value is the same as if the `get settings` command had been called on the same target right after the factory default settings were restored.

Refer to section 5.5 Command get for details.

5.4.1.5 Errors

`ILLEGAL_ARGUMENT` occurs if the target argument is not a valid device ID or group name.

5.4.1.6 Example

```
factory 001ec0f03668
{
   "status" : "PROCESSING",
   "request_id" : 18772,
   "result" : null,
   "error" : null
}
```

5.5 Command get

5.5.1 get

5.5.1.1 Command usage

`get` ***`target subset`***

5.5.1.2 Description

The `get` command fetches information from a single device or from a group of devices.

For each device, the retrieved information is a subset of the device object, which contains all configuration, status, supported features and other information about the device.

5.5.1.3 Arguments

The **target** argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`: All devices
`ALL_TX`: All transmitter devices
`ALL_RX`: All receiver devices

The valid values for the **subset** argument and their meaning are given in the table below.

Subset name	Since	Description
device	2.0	Retrieves all available information regarding target device(s).
edid	2.0	Retrieves the EDID from target device(s).
gpio	2.10	Retrieves the GPIO pins configuration and status from target device(s).
hello	2.2	Retrieves basic regarding target device(s) that are available when the device is first detected on the network, such as device type and firmware version.
identity	2.0	Retrieves basic identity information regarding target device(s), such as device name, device type and firmware version.
list	2.0	Lists target devices with no information other than device ID. Most useful with a device group as the target argument.
settings	2.0	Return detailed information regarding the current configuration and state of the target device(s). The return is like the return from `get device`. Most of the device object is returned, excluding the EDID, temperature, GPIO pins configuration and status, as well as the enumeration of available choices for configuration members.
temperature	2.5	Retrieves the temperature of target device(s).

5.5.1.4 Notes

When the **subset** argument is either `list` or `hello`, this command is guaranteed to return immediately. For all other subsets, the command will or may return with status `PROCESSING`.

Specifically, as an optimization, the BlueRiver Control Server caches information that allows it to return the requested information immediately for subset `settings` most of the time. However, it will sometimes return with status `PROCESSING` (mostly when it is busy processing background requests that affect the queried device). Client software must be able to handle the case where the `get settings` command returns with status `PROCESSING`.

5.5.1.5 Return Value

For each device targeted by the `get` command, there is either an entry in the devices array if the information was fetched successfully or a device error object in the error array if the information could not be fetched for some reason.

```
{
      "devices" : [DeviceObject, ...],
      "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
devices	2.0	Array of DeviceObject	An array of objects which are a subset of DeviceObject.
error	2.0	Array of DeviceError	An array of DeviceError objects indicating the devices for which the command failed and the reason why it failed

Refer to section 7 Device Object Definition (DeviceObject) for a description of the device object (DeviceObject) and section 2 Commands Overview for a description of the device error object (DeviceError).

5.5.1.6 Errors

`ILLEGAL_ARGUMENT` occurs if the target argument is not a valid device ID or group name.
`ILLEGAL_ARGUMENT` also occurs if the subset argument is not a valid subset name.

5.5.1.7 Example

See examples in Appendix A.

5.6 Command icron

The icron command provides sub-commands to control the optional on-board Icron Extreme USB extender.

5.6.1 icron program

5.6.1.1 Command usage

`icron` *`target`* `program firmware rs232` *`port_index`*

`icron` *`target`* `program eeprom rs232` *`port_index`*

`icron` *`target`* `program none`

5.6.1.2 Description

Enable or disable programming of the on-board Icron Extreme USB extender through one of the BlueRiver chipset's RS-232 port.

The first form of the command (with the **firmware** keyword) enables programming of the extender's firmware. The second form of the command (with the **eeprom** keyword) enables programming of the extender's non-volatile configuration memory. The third form of the command (with the **none** keyword) disables programming of the extender and restores normal operation of the RS-232 port.

For the complete programming procedure, please refer to document UG011 BlueRiver Icron Extreme USB Programming Procedure.

5.6.1.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **port_index** argument specifies the (zero-based) number of the RS-232 port through which the Icron USB extender is to be programmed.

5.6.1.4 Notes

This command was introduced in API version 2.13.

5.6.1.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the source right after the new settings were applied.

Refer to section 5.5 Command get for details.

5.6.1.6 Example

```
icron 001ec0f04d9c program firmware rs232 0
{
   "status" : "PROCESSING",
   "request_id" : 3929,
   "result" : null,
   "error" : null
}
icron 001ec0f04d9c program eeprom rs232 0
{
   "status" : "PROCESSING",
   "request_id" : 3930,
   "result" : null,
   "error" : null
}
icron 001ec0f04d9c program none
{
   "status" : "PROCESSING",
   "request_id" : 3931,
   "result" : null,
   "error" : null
}
```

5.7 Command join

5.7.1 join

5.7.1.1 Command usage

`join source:stream_type:index_tx destination[:subscription_type]:index_rx [display_mode video_args]`

5.7.1.2 Description

Subscribe one or more receiver device(s) to a transmitter device stream.

5.7.1.3 Arguments

The **source** argument must be the device ID of a single transmitter device.

The **stream_type** argument must be a stream type. The following types are supported:

`AUDIO` for an analog audio stream
`HDMI` for an HDMI video stream
`HDMI_AUDIO` for an HDMI audio stream
`I2S_AUDIO` for an I2S audio stream

The **index_tx** argument must the valid index of a stream of the specified type on the source device.

For the command to succeed, the source stream must have been previously started with the start command.

The **destination** argument must be either the device ID of a single receiver device or name of a device group.

The allowed device groups are the following:

`ALL`: all devices
`ALL_RX`: all receiver devices

The **subscription_type** argument is optional. If specified, it must be identical to the **stream_type** argument

(a subscription can only join a stream of the same type).

The **index_rx** argument must be a valid index of a subscription of the specified type on the destination device.

The optional **display_mode** and **video_args** arguments specify the display mode and video format arguments, respectively.

Refer to section 5.2 Specifying a Video Output Format for details.

These arguments can only be specified when joining a stream of type HDMI (i.e. the **stream_type** argument is HDMI). When they are, the command has the effect of a combined `join/set video` command.

5.7.1.4 Notes

Before joining HDMI video (i.e. stream type **HDMI**), the display mode and video output parameters must have been set on the receiver device(s). This can be accomplished either using a separate `set video` command, or by specifying the display mode and video output parameters directly in the `join` command as described above.

When joining a stream to **HDMI** subscription `0`, **HDMI_AUDIO** stream `0` from the same transmitter is also implicitly joined unless the receiver is in multiview mode or independent routing of HDMI audio is enabled.

Refer to section 5.10.3 mode split_hdmi_audio for details.

5.7.1.5 Return Value

The return value is the same as if the `get settings` command had been called on the destination right after the new settings were applied.

Refer to section 5.5 Command get for details.

5.7.1.6 Errors

`ILLEGAL_ARGUMENT`	Indicates the arguments entered are invalid.
`INVALID_STATE`	Occurs if the source device is disconnected.

The request may also fail for individual devices for the following reasons:

`INVALID_STATE`	Indicates that the source stream is stopped
`DEVICE_DISCONNECTED`	if the source or destination device is disconnected

If the source device is not connected when the command is issued, the command fails with `INVALID_STATE`.

If the source device is disconnected while the command is being processed, the command may fail for some destination devices with `DEVICE_DISCONNECTED`.

5.7.1.7 Example

```
join 001ec0f03668:AUDIO:0 001ec0f04d9c:0
{
      "status" : "PROCESSING",
      "request_id" : 61,
      "result" : null,
      "error" : null
}

join 001ec0f03668:HDMI:0 001ec0f04d9c:0 genlock
{
      "status" : "PROCESSING",
      "request_id" : 62,
      "result" : null,
      "error" : null
}
```

```
join 001ec0f03668:HDMI:0 001ec0f04d9c:0 fastswitch size 1920 1080 fps 60
{
      "status" : "PROCESSING",
      "request_id" : 63,
      "result" : null,
      "error" : null
}

join 001ec0f03668:HDMI:0 001ec0f04d9c:0 wall size 1920 1080 fps 30 offset
0 1080
{
      "status" : "PROCESSING",
      "request_id" : 64,
      "result" : null,
      "error" : null
}

join 001ec0f03668:HDMI:0 001ec0f04d9c:2 multiview fps 30
{
      "status" : "PROCESSING",
      "request_id" : 65,
      "result" : null,
      "error" : null
}
```

5.8 Command leave

5.8.1 leave

5.8.1.1 Command usage

`leave` ***`target:`*****`[`*****`subscription_type:`*****`[`*****`index`*****`]]`**

5.8.1.2 Description

Unsubscribe one or more receiver device(s) from a transmitter device stream.

5.8.1.3 Arguments

The **target** argument must be either the device ID of a single receiver device, or the name of a device group.

The allowed device groups are the following:

`ALL`: All devices.
`ALL_RX`: All receiver devices.

The **subscription_type** argument must be a subscription type. The following types are supported:

`AUDIO` for an analog audio stream
`HDMI` for an HDMI video stream
`HDMI_AUDIO` for an HDMI audio stream
`I2S_AUDIO` for an I2S audio stream

The **index** argument must the valid index of a subscription of the specified type on the target device.

If **index** is omitted, all subscriptions of the given type are affected. If **subscription _type** is omitted, all subscriptions of all the types listed above are affected.

5.8.1.4 Notes

The `leave` command was introduced in API version 2.1.

This command applies only to receiver devices.

5.8.1.5 Return Value

The return value is the same as if the `get settings` command had been called on the target device right after the new settings were applied.

Refer to section 5.5Command get for details.

5.8.1.6 Errors

`ILLEGAL_ARGUMENT` Occurs if the arguments entered are invalid.

The request may also fail for individual devices for the following reason:

`DEVICE_TYPE` Occurs if the device receiving the command is not a receiver device.

5.8.1.7 Example

```
leave 001ec0f04d9c:HDMI:0
{
   "status" : "PROCESSING",
   "request_id" : 3929,
   "result" : null,
   "error" : null
}
```

5.9 Command list

The list command provides sub-commands to query lists from the API server.

5.9.1 list firmware

5.9.1.1 Command usage

```
list firmware
```

5.9.1.2 Description

List all firmware update files available to the API server.

5.9.1.3 Notes

This command was introduced in API version 2.2.

5.9.1.4 Return Value

```
{
      "firmware" : [FirmwareItemObject, ...]
}
```

Member name	Since	Type	Description
firmware	2.2	Array of FirmwareItemObject	An array of objects which represent the firmware file

Each item in the firmware array (FirmwareItemObject) has the following structure:

```
{
```

```
        "file_name" : String,
        "supports_production" : String,
}
```

Member name	Since	Type	Description
file_name	2.2	String	File name of the firmware file
supports_production	2.12	Boolean	Whether the firmware file format supports first-time programming (i.e. with the **program** command). If this member is false, then the file can only be used for field upgrade (i.e. with the **update** command).

5.9.1.5 Example

```
list firmware
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "firmware" : [
         {
            "file_name" : "plethora_v3_4_0_0_eval_units.zip",
            "supports_production" : false
         },
         {
            "file_name" : "semtech_blueriver_3.5.0.0-
evaluation.apz",
            "supports_production" : true
         }
      ]
   },
   "error" : null
}
```

5.9.2 list multicast

5.9.2.1 Command usage

```
list multicast
```

5.9.2.2 Description

List information regarding the multicast IP addresses currently in use.

5.9.2.3 Notes

This command was introduced in API version 2.5.

5.9.2.4 Return Value

```
{
        "multicast" : [MulticastItemObject, ...]
}
```

Member name	Since	Type	Description
multicast	2.5	Array of	An array of objects which each represent a multicast IP

		MulticastItemObject	address

Each item in the multicast array (MulticastItemObject) has the following structure:

```
{
        "address" : String,
        "state" : String,
        "device_id" : String,
        "stream_type" : String,
        "stream_index" : Integer
}
```

Member name	Since	Type	Description
address	2.5	String	Multicast IP address in dotted decimal notation
state	2.5	String	Current state of the stream, which is one of the following: • **STREAMING** the device is actively transmitting of this multicast address • **STOPPED** the stream is stopped
device_id	2.5	String	Device ID of the device which is using the multicast address
stream_type	2.5	String	Type of the stream for which the multicast address is being used
stream_index	2.5	Integer	Index of the stream for which the multicast address is being used

5.9.2.5 Example

```
list multicast
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "multicast" : [
         {
            "address" : "224.1.1.1",
            "state" : "STREAMING",
            "device_id" : "d8803962feae",
            "stream_type" : "HDMI",
            "stream_index" : 0
         },
         {
            "address" : "224.1.1.2",
            "state" : "STREAMING",
            "device_id" : "d8803962dbf3",
            "stream_type" : "HDMI",
            "stream_index" : 0
         },
         {
            "address" : "224.1.1.3",
            "state" : "STREAMING",
            "device_id" : "001ec0f01818",
            "stream_type" : "AUDIO",
```

```
            "stream_index" : 0
         }
      ]
   },
   "error" : null
}
```

5.10 Command mode

The mode command provides sub-commands which modify the behaviour of the API.

Its effect is limited to a single API session.

5.10.1 mode async

5.10.1.1 Command usage

`mode async [(on|off)]`

5.10.1.2 Description

`Enable` or `disable` asynchronous event notification.

When asynchronous event notification is enabled, the API server sends an asynchronous notification to the client whenever new events are triggered, removing the need to poll events using the `event` command.

A notification is a JSON object with the same format as a command response, except that the status member is **`NOTIFICATION`**, which is not a valid status for a command response.

The result member of an asynchronous notification has the same format as the return value of the event command (i.e. a list of events, see description of the event command in section 5.3 Command event.

When an API session is launched, the asynchronous event notification status is `disabled`.

5.10.1.3 Arguments

When command entered, if neither on nor off is specified, the default is `on`.

5.10.1.4 Notes

This command was introduced in API version 2.1.

5.10.1.5 Return Value

`null`

5.10.1.6 Errors

`ILLEGAL_ARGUMENT` if any other value than on or off is specified.

5.10.1.7 Example

```
mode async
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}
{
   "status" : "NOTIFICATION",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "d8803962e664",
            "event_id" : 178,
            "event_type" : "SETTINGS_CHANGED",
            "timestamp" : 1460660784,
            "request_id" : 178
```

```
            },
            {
               "device_id" : "d8803962e664",
               "event_id" : 179,
               "event_type" : "SETTINGS_CHANGED",
               "timestamp" : 1460660784,
               "request_id" : 179
            }
         ]
      },
      "error" : null
   }
```

5.10.2 mode filter_rs232_event

5.10.2.1 Command usage

`mode filter_rs232_event [(on|off)]`

5.10.2.2 Description

`Enable` or `disable` RS-232 data event filtering.

While RS-232 data event filtering is enabled, client software does not receive RS232_RECEIVED events when a device sends RS-232 data to the BlueRiver Control server. This command can be used by client software that is not interested by RS-232 data to reduce the number of events it must handle.

When an API session is launched, RS-232 event filtering is `disabled`.

5.10.2.3 Arguments

When the command is entered, if neither on nor off is specified, the default is `on`.

5.10.2.4 Notes

This command was introduced in API version 2.12.

5.10.2.5 Return Value

`null`

5.10.2.6 Errors

`ILLEGAL_ARGUMENT` if any other value than on or off is specified.

5.10.2.7 Example

```
mode filter_rs232_event
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}
```

5.10.3 mode split_hdmi_audio

5.10.3.1 Command usage

`mode split_hdmi_audio [(on|off)]`

5.10.3.2 Description

`Enable` or `disable` independent routing of HDMI audio.

Prior to API 2.1, HDMI video and audio were routed together. It was not possible to route them separately.

By default, the API server enables a set of compatibility behaviours to provide an environment that is backwards compatible with pre-2.1 API applications. New applications **must** use this command to indicate they support independent routing of HDMI audio, which disables the compatibility behaviours.

Specifically, the following compatibility behaviours are **disabled** when enabling independent routing of HDMI audio with this command:

- When joining `HDMI subscription 0` of a receiver device to any `HDMI` stream of a transmitter device, `HDMI_AUDIO subscription 0` from the same receiver device is implicitly joined to `HDMI_AUDIO` stream 0 of the same transmitter device, unless the receiver device is in multiview mode (refer to section 5.7 Command join).
- When starting `HDMI` stream 0 of a transmitter device, `HDMI_AUDIO` stream 0 of the same device is also implicitly started (refer to section 5.16 Command start).
- When stopping `HDMI` stream 0 of a transmitter device, `HDMI_AUDIO` stream 0 of the same device is also implicitly stopped (see section 5.17 Command stop).

Note: The `leave` command, which was introduced in API 2.1, is not affected by any of the items listed above.

TIP: It is highly recommended that new applications always enable independent routing of HDMI audio. This is especially important in the following case:

- For Receiver devices that are configured to receive HDMI video and audio from **different** transmitter devices.

The client application manages the allocation of multicast IP addresses (see section5.16 Command start).

Reminder: At the beginning of an API session, independent routing of HDMI audio is disabled. It must be enabled explicitly using this command. This may change in a future version of this API (based on the API version requested with the `require` command).

5.10.3.3 Arguments

If neither on or off is specified, the default is `on`.

5.10.3.4 Notes

This command was introduced in API version 2.1.

5.10.3.5 Return Value

`null`

5.10.3.6 Errors

`ILLEGAL_ARGUMENT` if some value other than on or off is specified

5.10.3.7 Example

```
mode split_hdmi_audio
{
      "status" : "SUCCESS",
      "request_id" : null,
      "result" : null,
      "error" : null
}
```

5.11 Command program

5.11.1 program

5.11.1.1 Command usage

program ***target vendor_id product_id file_name***

5.11.1.2 Description

Perform first-time programming of the BlueRiver chipset firmware on one or more devices.

This command is mostly useful in a R&D laboratory environment or for low-volume manufacturing. To perform a firmware update, use the **update** command instead. Refer to section 5.20 "Command update" for a complete description.

Unless explicitly stated otherwise, this command behaves like the **update** command. The main differences between this command and the **update** command are:

- The vendor ID and product ID to program into the device are specified explicitly as part of this command, whereas the **update** command reads these IDs from a device that has already been programmed at least once to choose the correct firmware configuration.
- This command programs the golden firmware image in addition to the primary firmware image while the **update** command leaves it untouched.
- This command reverts device configuration to factory defaults while the **update** command leaves it untouched.

Warning: Targeting devices in a customer installation with this command should not be needed and is not recommended. Mistakenly using this command with a vendor ID and product ID that do not match the actual hardware of the target device(s) can render these devices incapable of communicating over the network (i.e. "bricked"), a situation which can be resolved with means typically only available in a manufacturing or R&D laboratory environment (i.e. individually for each device through the JTAG interface) if at all.

5.11.1.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **vendor_id** and **product_id** arguments must be the vendor ID and product ID of the device(s), respectively.

The **file_name** argument is the file name of a firmware file. The available firmware files can be listed with the list firmware command (refer to section 5.9.1 "list firmware"). Only the firmware files for which the list firmware command reports they support production can be used with this command.

5.11.1.4 Notes

This command was introduced in API version 2.13.

5.11.1.5 Return Value

The return value of this command is identical to the **update** command. Please see section 5.20.1.5 “Return Value”.

5.11.1.6 Errors

`ILLEGAL_ARGUMENT`	Occurs if the target argument entered is not a valid device ID or group name.
`ILLEGAL_ARGUMENT`	Occurs if the specified firmware file does not support production.
`IO_ERROR`	Occurs if the firmware file could not be opened.

The request may also fail for individual devices for the following reasons:

`IO_ERROR`	Occurs if the firmware file could not be read.
`RESOURCE_NOT_FOUND`	Occurs if specified firmware file does not contain a firmware for the device.

5.11.1.7 Example

```
program 1 1 001ec0f04d9c semtech_blueriver_3.5.0.0-evaluation.apz
{
   "status" : "PROCESSING",
   "request_id" : 32223,
   "result" : null,
   "error" : null
}

request 32223
{
   "status" : "SUCCESS",
   "request_id" : 32223,
   "result" : {
      "update" : [
         {
            "device_id" : "001ec0f04d9c"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.12 Command reboot

5.12.1 reboot

5.12.1.1 Command usage

`reboot` ***`target`***

5.12.1.2 Description

Causes the target device(s) to restart.

Note: The target device(s) will become unavailable for a short period of time while the restart is completed.

5.12.1.3 Arguments

The target argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

5.12.1.4 Return Value

For each device targeted by the reboot command, there is either an entry in the reboot array if the data was sent successfully or a device error object in the error array if the operation failed for some reason.

```
{
        "reboot" : [RebootObject, ...],
        "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
reboot	2.2	Array of RebootObject	An array of objects which represent the devices for which the request succeeded.
error	2.2	Array of DeviceError	An array of DeviceError objects indicating the devices for which the command failed and the reason why it failed

The RebootObject object has the following structure:

```
{
        "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.3	String	Device ID

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

Comment: Prior to API 2.3, the return value for this command was `null`.

5.12.1.5 Errors

`ILLEGAL_ARGUMENT` if the target argument is not a valid device ID or group name

5.12.1.6 Example

```
reboot 001ec0f03668
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "reboot" : [
         {
            "device_id" : "001ec0f03668"
         }
      ],
      "error" : []
   },
   "error" : null
```

```
}
```

5.13 Command request

5.13.1 request

5.13.1.1 Command usage

`request request_id`

5.13.1.2 Description

This function retrieves either the return value of a background command (i.e. a command that responded with the `PROCESSING` status and a request ID) or the data associated with an event.

5.13.1.3 Arguments

The `request_id` is a request identification number included in either in the `PROCESSING` command response or the associated event object.

5.13.1.4 Notes

Request numbers expire 10 minutes after they are created.

Calling the request command with an expired request number results in an `ILLEGAL_ARGUMENT` error.

5.13.1.5 Return Value

The return value is dependant on the request type. Refer to the sections below for details.

5.13.1.6 Errors

`ILLEGAL_ARGUMENT` if the given request ID does not exist.

5.13.1.7 Example

Refer to examples provided in the subsections of this section for details.

5.13.2 Requests associated with background commands

5.13.2.1 Description

The `REQUEST_COMPLETE` event indicates the completion of a background command (i.e. a command that responded with status `PROCESSING` and a request ID instead of status `SUCCESS`).

5.13.2.2 Return Value

Once the request completes, the return type is the one documented for a background command.

If request is called before the request completes (i.e. without checking first for an event of type `REQUEST_COMPLETE`), the request command responds with status `PROCESSING` and the same request ID as the one specified when calling the request.

5.13.2.3 Example

The example below illustrates how the `request` command is used to confirm successful execution of a `stop` command sent to a specific device.

```
stop 001ec0f03668:HDMI:0
{
   "status" : "PROCESSING",
   "request_id" : 85521,
   "result" : null,
   "error" : null
```

```
}
request 85521
{
   "status" : "PROCESSING",
   "request_id" : 85521,
   "result" : null,
   "error" : null
}
event 85528
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : null,
            "event_id" : 85529,
            "event_type" : "REQUEST_COMPLETE",
            "timestamp" : 1462902262,
            "request_id" : 85521
         }
      ]
   },
   "error" : null
}
request 85521
{
   "status" : "SUCCESS",
   "request_id" : 85521,
   "result" : {
      "devices" : [
         {
            "device_id" : "001ec0f03668",
            ...
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.13.3 Requests associated with event CEC_RECEIVED

5.13.3.1 Description

The `CEC_RECEIVED` event indicates that Consumer Electronics Control (CEC) data has been received by a specific device.

5.13.3.2 Return Value

```
{
   "cec" : [CEC_DataObject, ...]
}
```

Member name	Since	Type	Description
cec	2.13	Array of CECDataObject	Array of CEC data objects, described below

The `CECDataObject` includes two strings:

1. Representing the device that the CEC data was received from.
2. The CEC data itself.

```
{
   "device_id" : String,
   "cec_data" : String
}
```

Member name	Since	Type	Description
device_id	2.13	String	Device ID of the device from which the data was received
cec_data	2.13	String	Hexadecimal string which represent the received CEC data.

5.13.3.3 Example

```
event 85750
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec0f04d9c",
            "event_id" : 85752,
            "event_type" : "CEC_RECEIVED",
            "timestamp" : 1462903915,
            "request_id" : 85744
         }
      ]
   },
   "error" : null
}
request 85744
{
   "status" : "SUCCESS",
   "request_id" : 85744,
   "result" : {
      "cec" : [
         {
            "device_id" : "001ec0f04d9c",
            "cec_data" : "4f36"
         }
      ],
      "error" : []
   },
   "error" : null

}
```

5.13.4 Requests associated with event DEVICE_CONNECTED

5.13.4.1 Description

The `DEVICE_CONNECTED` event indicates that a new device has joined the network.

5.13.4.2 Return Value

Basic information on the device is returned. The format is identical to the output of the "`get hello`" command that would have been called on the device associated with the `DEVICE_CONNECTED` event.

5.13.4.3 Example

See Appendix A, specifically A.6 Command "get hello".

5.13.5 Requests associated with event DEVICE_DISCONNECTED

5.13.5.1 Description

The `DEVICE_DISCONNECTED` event indicates that a device has left the network.

5.13.5.2 Return Value

Basic information on the device is returned. The format is identical to the output of the "`get hello`" command that would have been called on the device associated with the `DEVICE_ DISCONNECTED` event.

5.13.5.3 Example

See Appendix A, specifically A.6 Command "get hello".

5.13.6 Requests associated with event DEVICE_INFO_AVAILABLE

5.13.6.1 Description

The `DEVICE_INFO_AVAILABLE` event indicates that device information is available for a newly discovered device.

5.13.6.2 Return Value

The device information of a newly connected device is returned. The format is identical to the output of the "`get device`" command that would have been called on the device associated with the `DEVICE_INFO_AVAILABLE` event.

5.13.6.3 Example

Refer to Appendix A, specifically A.1 Command "get device" for Transmitter Device and A.2 Command "get device" for Receiver Device.

5.13.7 Requests associated with event FIRMWARE_UPDATE_PROGRESS

5.13.7.1 Description

The `FIRMWARE_UPDATE_PROGRESS` event indicates that a firmware update is in progress.

Information associated with this event can be used to report progress information to the user, for example by displaying a progress bar.

5.13.7.2 Return Value

```
{
        "update_progress" : [UpdateProgressObject, ...]
}
```

Member name	Since	Type	Description
update_progress	2.4	Array of UpdateProgressObject	Array of firmware update progress objects, described below.

The `UpdateProgressObject` includes the following:

```
{
        "device_id" : String,
        "progress" : Integer,
        "total" : Integer
}
```

Member name	Since	Type	Description
device_id	2.4	String	Device ID of the device from which the data was received
progress	2.4	Integer	The number of progress units accomplished so far. These units are arbitrary and subject to change: they are only meaningful relative to the value of the **total** member.
total	2.4	Integer	The number of progress units for the whole operation

5.13.7.3 Example

```
event 22128
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec0f04d9c",
            "event_id" : 22129,
            "event_type" : "FIRMWARE_UPDATE_PROGRESS",
            "timestamp" : 1462471715,
            "request_id" : 22150
         }
      ]
   },
   "error" : null
}
request 22150
{
   "status" : "SUCCESS",
   "request_id" : 22150,
   "result" : {
      "update_progress" : [
         {
            "device_id" : "001ec0f04d9c",
            "progress" : 3603552,
            "total" : 8643204
         }
      ],
      "error" : []
   },
   "error" : null
```

```
}
```

5.13.8 Requests associated with event I2C

5.13.8.1 Description

The `I2C` event indicates that an I2C bus master (e.g. a MCU) has generated an event through the I2C interface.

5.13.8.2 Return Value

```
{
        "i2c_event" : [I2CEventObject, ...]
}
```

Member name	Since	Type	Description
i2c_event	2.13	Array of I2CEventObject	Array of I2C event objects, described below.

The `IRDataObject` includes two strings:

1. Representing the device that the IR data was received from.
2. The IR data itself.

```
{
        "device_id" : String,
        "event_source" : Integer
}
```

Member name	Since	Type	Description
device_id	2.13	String	Device ID of the device from which the data was received
event_source	2.13	Integer	Number of the event source. This number is specified by the I2C bus master that generates the event. Event sources 0 to 7 are currently supported. It is recommended that event source 0 be reserved for a feature that allows a device to be identified (e.g. user physically presses a button on the device which has the effect of highlighting the device in the software's user interface in some way). The semantic of other event sources is product-specific. Recommended practice: • If your product implements an "identify device" feature as described above, use event source 0 for that purpose. • If your product does not implement an "identify device" feature, do not use event source 0. • Except for event source 0, software should not interpret I2C events unless it is for a device of a type it recognizes, basing that decision on the device's vendor ID and product ID.

5.13.8.3 Example

```
event 1233
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec0f03668",
            "event_id" : 1234,
            "event_type" : "I2C",
            "timestamp" : 1462903093,
            "request_id" : 1300
         }
      ]
   },
   "error" : null
}
request 1300
{
   "status" : "SUCCESS",
   "request_id" : 1300,
   "result" : {
      "i2c_event" : [
         {
            "device_id" : "001ec0f03668",
            "event_source" : 4
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.13.9 Requests associated with event INFRARED_RECEIVED

5.13.9.1 Description

The `INFRARED_RECEIVED` event indicates that Infrared data has been received by a specific device.

5.13.9.2 Return Value

```
{
      "infrared" : [IRDataObject, ...]
}
```

Member name	Since	Type	Description
Infrared	2.0	Array of IRDataObject	Array of infrared data objects, described below

The `IRDataObject` includes two strings:

1. Representing the device that the IR data was received from.
2. The IR data itself.

```
{
        "device_id" : String,
        "infrared_data" : String
}
```

Member name	Since	Type	Description
device_id	2.0	String	Device ID of the device from which the data was received
infrared_data	2.0	String	Hexadecimal string which represent the received Pronto code.

5.13.9.3 Example

```
event 85609
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec0f03668",
            "event_id" : 85611,
            "event_type" : "INFRARED_RECEIVED",
            "timestamp" : 1462903093,
            "request_id" : 85604
         }
      ]
   },
   "error" : null
}
request 85604
{
   "status" : "SUCCESS",
   "request_id" : 85604,
   "result" : {
      "infrared" : [
         {
            "device_id" : "001ec0f03668",
            "infrared_data" :
"0000006d0000002700ac00a90016000f0016000f00150010001500100014001100
4003700150010001400110015001000150010001500100015001000150010001500
0001400110015001000140 0a900150036001500360014003700150010001500360
0150036001500100015003600150010001 6000f00150010001 6000f00150010001500
0001500360016000f00150036001500360015003600140037001501 7c"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.13.10 Requests associated with event RS232_RECEIVED

5.13.10.1 Return Value

```
{
```

```
   "rs232" : [RS232DataObject, ...]
}
```

Member name	Since	Type	Description
rs232	2.0	Array of RS232DataObject	Array of RS-232 data objects, described below

The `RS232DataObject` includes two strings:

3. Representing the device that the RS-232 data was received from.
4. The RS-232 data itself.

```
{
   "device_id" : String,
   "rs232_data" : String,
   "stream_idx" : Integer
}
```

Member name	Since	Type	Description
device_id	2.0	String	Device ID of the device from which the data was received
rs232_data	2.0	String	ASCII string representation of RS-232 data. The binary data is escaped using the following rules: ➔ \b replaces the ASCII backspace character (hex. 08) ➔ \f replaces the ASCII form feed (hex. 0c) ➔ \n replaces the ASCII line feed (hex. 0a) ➔ \r replaces the ASCII carriage return (hex. 0d) ➔ \t replaces the ASCII tab character (hex. 09) ➔ \\ replaces a single backslash (\) character ➔ \x*nn*, where *nn* is the hexadecimal value (two hexadecimal digits) of the character, replaces: • all ASCII control characters (hex. 1f and below) not already handled above; • the ASCII DEL character (hex. 7f); and • all non-ASCII characters (hex. 80 and above). This is in addition to the JSON escaping rules. The relevant JSON rules are as follow: ➔ \" replaces a double quote ➔ \\ replaces a single backslash (\) character A JSON parser will handle JSON escaping rules, leaving only the first set of rules to be handled.
stream_idx	2.1	Integer	RS-232 port number from which the data was sent. This is also the **RS232** stream/subscription index.

5.13.10.2 Example

```
event 85750
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "events" : [
         {
            "device_id" : "001ec0f04d9c",
            "event_id" : 85752,
```

```
            "event_type" : "RS232_RECEIVED",
            "timestamp" : 1462903915,
            "request_id" : 85744
         }
      ]
   },
   "error" : null
}
request 85744
{
   "status" : "SUCCESS",
   "request_id" : 85744,
   "result" : {
      "rs232" : [
         {
            "device_id" : "001ec0f04d9c",
            "rs232_data" : "Hello",
            "stream_idx" : 0
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.13.11 Requests associated with event SETTINGS_CHANGED

5.13.11.1 Return Value

The new settings are returned.

The format is identical to the output of the "`get settings`" command that would have been called on the device associated with the `SETTINGS_CHANGED` event.

5.13.11.2 Example

Refer to Appendix A, specifically A.9 Command "get settings" for Transmitter Device and A.10 Command "get settings" for Receiver Device.

5.14 Command send

The `send` command provides sub-commands which can be used to send data to one or more device(s).

5.14.1 send cec

5.14.1.1 Command usage

`send` ***`target`*** **`cec`** ***`data_hex`***

5.14.1.2 Description

Send Consumer Electronics Control (CEC) data to one or more device(s).

5.14.1.3 Arguments

The `target` argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL` All devices

`ALL_TX` All transmitter devices
`ALL_RX` All receiver devices

The **`data_hex`** argument is a hexadecimal string which represents the CEC data.

5.14.1.4 Return Value

For each device targeted by the send infrared command, there is either an entry in the send_cec array if the data was sent successfully or a device error object in the error array if the operation failed for some reason.

```
{
      "send_cec" : [SendCECObject, ...],
      "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
send_cec	2.13	Array of SendInfraredObject	An array of objects which represent devices for which the request succeeded.
error	2.13	Array of DeviceError	An array of DeviceError objects indicating devices for which the command failed and the reason why it failed

The `SendCECObject` object has the following structure:

```
{
      "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.13	String	Device ID

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

5.14.1.5 Errors

`ILLEGAL_ARGUMENT` returned if the target argument is not a valid device ID or group name.
`ILLEGAL_ARGUMENT` also returned if the data is not in the format described above.

The request may also fail for individual devices for the following reasons:

`DEVICE_TYPE` if the device does not support CEC.

5.14.1.6 Example

```
send 001ec0f03668 cec 0f36
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "send_cec" : [
         {
            "device_id" : "001ec0f03668"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.14.2 send infrared

5.14.2.1 Command usage

`send target infrared data_hex`

5.14.2.2 Description

Send infrared remote-control data to one or more device(s).

5.14.2.3 Arguments

The `target` argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **`data_hex`** argument is a hexadecimal string which represents the Pronto infrared code to be sent. The length of this string must be a multiple of four bytes (i.e. eight hexadecimal characters) and at most 1032 bytes (i.e. 2064 hexadecimal characters or 256 Pronto burst pairs).

Note: Depending on the firmware version, some devices only support a code length of up to 512 bytes – refer to the **data_length_max** member in section 7.7.17.2 Status to detect whether this is the case.

5.14.2.4 Return Value

For each device targeted by the send infrared command, there is either an entry in the send_infrared array if the data was sent successfully or a device error object in the error array if the operation failed for some reason.

```
{
        "send_infrared" : [SendInfraredObject, ...],
        "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
send_infrared	2.2	Array of SendInfraredObject	An array of objects which represent devices for which the request succeeded.
error	2.2	Array of DeviceError	An array of DeviceError objects indicating devices for which the command failed and the reason why it failed

The `SendInfraredObject` object has the following structure:

```
{
        "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.2	String	Device ID

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

Note: Prior to API 2.2, the return value for this command was null.

5.14.2.5 Errors

`ILLEGAL_ARGUMENT` returned if the target argument is not a valid device ID or group name.
`ILLEGAL_ARGUMENT` also returned if the data is not in the format described above.

The request may also fail for individual devices for the following reasons:

`DEVICE_TYPE` if the device does not have an infrared output.

5.14.2.6 Example

```
send 001ec0f03668 infrared
0000006d00220002015700ab00160015001600150016003f00160015001600150016
00150016001500160015001600 3f0016003f00160015001600150016001500160015
0016003f0016003f0016003f0016003f00160015001600150016001500160015001600
00150016001500160015001600150016003f0016003f0016003f0016003f0016003f
0016003f001605f30156005500160e4d
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "send_infrared" : [
         {
            "device_id" : "001ec0f03668"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.14.3 send rs232

5.14.3.1 Command usage

`send` *`target`* `rs232[:`*`port`*`]` *`data_string`*

5.14.3.2 Description

Send RS-232 data to one or more device(s).

5.14.3.3 Arguments

The **target** argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The optional `port` argument is the RS-232 port number. If omitted, the default value for port is `0`. Not all devices support multiple RS-232 ports.

The `data_string` argument is a printable ASCII text string.

Note: This string must be escaped if it contains ASCII control characters or non-ASCII byte values. Space characters must also be escaped.

The following escape sequences are recognized:

`\0`	nul character
`\n`	line feed
`\r`	carriage return
`\t`	horizontal tab
`\xnn`	(where nn are two hexadecimal characters) hexadecimal representation of byte
`\\`	single backslash

`\(space)` space character

5.14.3.4 Return Value

For each device targeted by the send rs232 command, there is either an entry in the send_rs232 array if the data was sent successfully or a device error object in the error array if the operation failed for some reason.

```
{
        "send_rs232" : [SendRs232Object, ...],
        "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
send_rs232	2.2	Array of SendRs232Object	An array of objects which represent the devices for which the request succeeded.
error	2.2	Array of DeviceError	An array of DeviceError objects indicating the devices for which the command failed and the reason why it failed

The `SendRs232Object` object has the following structure:

```
{
        "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.2	String	Device ID

Refer to section2 Commands Overview for a description of the device error object (DeviceError).

Note: Prior to API 2.2, the return value for this command was null.

5.14.3.5 Errors

`ILLEGAL_ARGUMENT` returned when the target argument is not a valid device ID or group name.
`ILLEGAL_ARGUMENT` also returned if the data is not in the format described above.

The request may also fail for individual devices for the following reasons:

`DEVICE_TYPE` returned if the device does not have a UART or the RS-232 port number specified

5.14.3.6 Example

```
send 001ec0f03668 rs232 Hello\ World!
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "send_rs232" : [
         {
            "device_id" : "001ec0f03668"
         }
      ],
      "error" : []
   },
   "error" : null
}
send 001ec0f03668 rs232:2 Hello\ World!\r\n
```

```
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "send_rs232" : [
         {
            "device_id" : "001ec0f03668"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.15 Command set

5.15.1 set edid

5.15.1.1 Command usage

set *target* edid *data*

5.15.1.2 Description

Set the EDID of one or more transmitter device(s).

5.15.1.3 Arguments

The **target** argument must be either the device ID of a single transmitter device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices

The **data** argument must be a hexadecimal string which represents the binary data to set. Its length must be 512 hexadecimal characters (i.e. data length must be 256 bytes). Since the data represents an EDID, the data string must start with "`00ffffffffffff00`" (that is, bytes `00 ff ff ff ff ff ff 00`, which is the EDID header).

5.15.1.4 Notes

This command applies only to transmitter devices.

5.15.1.5 Return Value

The return value is the same as if the "`get edid`" command had been called on the same target right after the new settings were applied. Refer to section 5.5 Command get for details.

5.15.1.6 Errors

`ILLEGAL_ARGUMENT`	Returned when the target argument is not a valid device ID or group name.
`ILLEGAL_ARGUMENT`	Also returned if the data argument is not the proper length.
`ILLEGAL_ARGUMENT`	in addition, this error is also returned if the data argument does not begin with the EDID header.

5.15.1.7 Example

```
set 001ec0f03668 edid
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250
542f6f00714f8180818a9500950fb3000101d1c0023a801871382d40582c4500dc0c
1100001e000000ff00415535313031323031313731373137000000fd00384c1e5315000a
2020202020200000fc005068696c697073202323231450a01b602031ef04b100501
020304061213141f2309070783010000065030c001100023a801871382d40582c4500
dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e206e28
5500dc0c1100001e8c0ad090204031200c405500dc0c1100001800000000000000000
00000000000000000000000000000000000ac
{
   "status" : "PROCESSING",
   "request_id" : 57,
   "result" : null,
   "error" : null
}
```

5.15.2 set gpio

5.15.2.1 Command usage

set *target* gpio *index* mode INPUT

set *target* gpio *index* mode OUTPUT value (0|1)

5.15.2.2 Description

Set the direction and output value of a general-purpose input/output (GPIO) pin of one or more device(s).

5.15.2.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **index** argument specifies the index of the GPIO pin. It must match the index of a GPIO node on the device.

The **mode** argument specifies the direction of the GPIO pin:

`INPUT`	to set the GPIO pin as an input.
`OUTPUT`	to set the GPIO pin as an output.

The **value** argument specifies the output value of the GPIO pin:

- Must be present if the GPIO pin is set as an output (i.e. if the **mode** argument is `OUTPUT`).
- Must not be present if the GPIO pin is set as an input (i.e. if the **mode** argument is `INPUT`).

5.15.2.4 Notes

This command was introduced in API version 2.10.

5.15.2.5 Return Value

For each device targeted by the **set gpio** command, there is either an entry in the GPIO array if the operation was performed successfully on the device or a device error object in the error array if the operation failed for some reason.

```
{
        "gpio" : [GpioResultObject, ...],
        "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
gpio	2.10	Array of GpioResultObject	An array of objects which represent the result of the operation for each device.
error	2.10	Array of DeviceError	An array of DeviceError objects indicating the devices for which the operation failed and the reason why it failed

The `GpioResultObject` object has the following structure:

```
{
        "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.10	String	Device ID

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

5.15.2.6 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid.

5.15.2.7 Example

```
set d88039631d77 gpio 7 mode output value 1
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "gpio" : [
         {
            "device_id" : "d88039631d77"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

5.15.3 set ip

5.15.3.1 Command usage

`set target ip mode mode [address address mask mask gateway gateway]`

5.15.3.2 Description

Set the Internet Protocol (IP) configuration of the primary network interface of a target device.

5.15.3.3 Arguments

The **target** argument must be device ID of a single device.

The **mode** argument must be one of the following:

`MANUAL`	For manual IP configuration.
`DHCP`	For automatic configuration using a DHCP server (with fallback on auto-IP).

If mode is `MANUAL`, the **address**, **mask** and **gateway** arguments must be specified.

- The **address** argument is the IP address.
- The **mask** argument is the subnet mask.
- The **gateway** argument is the IP address of the default network gateway.

If mode is `DHCP`, these arguments must not be specified.

5.15.3.4 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied. Refer to section 5.5 Command get for details.

5.15.3.5 Errors

`ILLEGAL_ARGUMENT`	if the target argument entered is not a valid device ID (including if it is a group name).

5.15.3.6 Example

```
set 001ec0f03668 ip mode dhcp
{
   "status" : "PROCESSING",
   "request_id" : 58,
   "result" : null,
   "error" : null
}

set 001ec0f03668 ip mode manual address 10.1.1.5 mask 255.255.255.0
gateway 10.1.1.1
{
   "status" : "PROCESSING",
   "request_id" : 59,
   "result" : null,
   "error" : null
}
```

5.15.4 set property

5.15.4.1 Command usage

`set` *`target`* `property` *`key value`*

5.15.4.2 Description

Set a configuration property for one or more device(s).

For more information on configuration properties, refer to section 8 Configurable Device Properties.

5.15.4.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **key** argument identifies a configurable property of a device and the **value** argument is a value of a type appropriate for that property.

The **key** argument is a path that refers to an object member inside the device object, starting at the root of the device object (refer to section 7Device Object Definition (DeviceObject)).

Syntax rules for a key are as follows:

- The key starts with the name of one of the members of the device object.
- Object members are accessed by appending a period (.) followed by the member name.
- Array elements are accessed by appending an array selector between square brackets ([and]).
- An array selector is a name and an index separated by a colon (e.g. [name:3]).
    - The array must be an array of objects (i.e. each element is an object).
    - The matched element is the one that has a member named "type" or "name" (checked in that order) which matches the provided name and a member named "index" which matches the provided index.

For a list of all configurable device properties, refer to section 8 Configurable Device Properties.

5.15.4.4 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied. Refer to section 5.5 Command get for details.

5.15.4.5 Errors

`ILLEGAL_ARGUMENT`	If the target argument entered is not a valid device ID or group name.

The request may also fail for individual devices for the following reasons:

`DEVICE_TYPE`	Returned if the property being set or the chosen value for that property is not supported by the device.

5.15.4.6 Example

```
set 001ec0f04d9c property
nodes[ANALOG_AUDIO_OUTPUT:0].inputs[main:0].configuration.source.value 3
{
   "status" : "PROCESSING",
   "request_id" : 60,
   "result" : null,
   "error" : null
}
```

5.15.5 set scaler

5.15.5.1 Command usage

`set` *`target`* `scaler size` *`width height`*

5.15.5.2 Description

Set the scaler video resolution of one or more transmitter device(s).

5.15.5.3 Arguments

The **target** argument must be either the device ID of a single transmitter device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices

The **width** and **height** arguments specify respectively the width and the height of the scaled video, in pixels.

5.15.5.4 Notes

This command was introduced in API version 2.6.

The scaler resolution must be set using this command before a scaled video stream can be started.

5.15.5.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied. Refer to section 5.5 Command get for details.

5.15.5.6 Errors

`ILLEGAL_ARGUMENT` Occurs if the arguments entered are invalid.

5.15.5.7 Example

```
set d88039634e85 scaler size 800 600
{
   "status" : "PROCESSING",
   "request_id" : 13133,
   "result" : null,
   "error" : null
}
```

5.15.6 set video

5.15.6.1 Command usage

set target video **[*display_mode video_args*]**

5.15.6.2 Description

Set the display mode and video output parameters of one or more receiver device(s).

5.15.6.3 Arguments

The **target** argument must be either the device ID of a single receiver device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_RX`	All receiver devices

The `display_mode` and `video_args` arguments specify the display mode and video format arguments, respectively. Refer to section 5.2 Specifying a Video Output Format for details.

5.15.6.4 Notes

This command was introduced in API version 2.1.

To set a device in multiview display mode and apply a specific multiview layout, refer to 6.6.1 set multiview.

TIP: For backward compatibility, the command:

set *target* video multiview *video_args*

is equivalent to

set target multiview compatibility_4k_2x2 video_args

that is, to calling the `set multiview` command with the predefined **compatibility_4k_2x2** layout. Refer to section 6.2.4 Predefined Compatibility Layout for details.

5.15.6.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied.

Refer to 5.5 Command get for details.

5.15.6.6 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid

5.15.6.7 Example

This first example places the receiver in genlock display mode:

```
set 001ec0f04d9c video genlock
{
      "status" : "PROCESSING",
      "request_id" : 61,
      "result" : null,
      "error" : null
}
```

Supported video format 1080p60 specified by video resolution and frame rate in fast switch full screen display mode:

```
set 001ec0f04d9c video fastswitch size 1920 1080 fps 60
{
      "status" : "PROCESSING",
      "request_id" : 62,
      "result" : null,
      "error" : null
}
```

Slightly different example here, supported video format 1080p60 specified by Video Identification Code (VIC) in fast switch full screen display mode:

```
set 001ec0f04d9c video fastswitch vic 16
{
      "status" : "PROCESSING",
      "request_id" : 62,
      "result" : null,
      "error" : null
}
```

In this example, the supported video format is 3840x2160 (2160p) 59.94Hz in stretched fast switch full screen display mode:

```
set d88039631d77 video fastswitch stretch size 3480 2160 fps 60m
{
      "status" : "PROCESSING",
      "request_id" : 1349,
      "result" : null,
      "error" : null
}
```

Fully specified 800x600 video format in fast switch full screen display mode:

```
set d88039631d77 video fastswitch size 800 600 fps 60.316541 total
1056 628 pulse 128 4 front_porch 40 1 polarity positive positive
{
      "status" : "PROCESSING",
      "request_id" : 1350,
      "result" : null,
      "error" : null
}
```

video wall设置

Display wall (wall display mode) with a 1080p 59.94Hz output video format and a bottom black bar:

```
set d88039631d77 video wall size 1920 1080 fps 60m offset 0 810 keep
960 270 viewport 0 0 1920 540
{
      "status" : "PROCESSING",
      "request_id" : 1351,
      "result" : null,
      "error" : null
}
```

5.16 Command start

5.16.1 start

5.16.1.1 Command usage

`start target:[stream_type[:index [multicast_addr]]]`

5.16.1.2 Description

Starts one or more stream(s) on one or more transmitter device(s).

5.16.1.3 Arguments

The **target** argument must be either the device ID of a single transmitter device, or the name of a device group.

Note: If the **multicast_addr** argument is specified the **target** must be the device ID of a single transmitter device.

The following device groups are allowed:

`ALL`	All devices
`ALL_TX`	All transmitter devices

The **stream_type** argument must be a stream type. The following types are supported:

`AUDIO`	for an analog audio stream
`HDMI`	for an HDMI video stream
`HDMI_AUDIO`	for an HDMI audio stream
`I2S_AUDIO`	for an I2S audio stream

Comment: For all other stream `types`, use the `switch` command. For details refer to section5.18 Command switch.

The **index** argument must the valid index of a stream of the specified type.

If an **index** argument omitted (deprecated - see note below), then all streams of the given type are started.

If **stream_type** argument omitted (deprecated - see note below), all streams of all types listed above are started.

Note: Omitting the **stream_type** and/or **index** argument(s) is highly discouraged. A firmware update can cause a device to support new streams 1) of which the control application is not aware and 2) which are started by these forms of the start command. This, in turn, can cause the command to fail because of unmet preconditions or the system to malfunction because of network bandwidth issues. An application should be explicit about which streams it intends to start.

The **multicast_addr** argument, if present, must be an available (i.e. unused) multicast IP address within the allocation range configured in the configuration file of the BlueRiver Control server (defaults to [224.1.1.1, 224.1.3.255]). Furthermore, the following multicast IP addresses are reserved and must not be used:

- The "`all TX`" multicast IP address (also specified in the configuration file, defaults to 224.1.1.253).
- The legacy "`all TX`" multicast IP address 224.1.1.253 (which may or may not be identical to the above depending on BlueRiver configuration file content).
- The "`all RX`" multicast IP address (also specified in the configuration file, defaults to 224.1.1.254).
- The legacy "`all RX`" multicast IP address 224.1.1.254 (which might or might not be identical to the above depending on configuration file content).
- Multicast IP address 225.225.225.225.

If **multicast_addr** is not specified, the BlueRiver Control Server uses the address previously assigned to each stream when applicable or otherwise it allocates an unused address.

5.16.1.4 Notes

The **multicast_addr** argument was added in API 2.5. The multicast allocation range became configurable starting with API 2.12.

When starting `HDMI` stream `0`, `HDMI_AUDIO` stream `0` from the same transmitter is also implicitly started unless independent routing of HDMI audio is enabled. Refer to Section 5.10.3 mode split_hdmi_audio for details.

When the client manually manages the allocation of multicast IP addresses using the **multicast_addr** argument, it is recommended to enable independent routing of HDMI audio so that the multicast IP address of the `HDMI` and `HDMI_AUDIO` streams can be specified independently.

Otherwise, the HDMI stream uses the specified address and the BlueRiver Control Server automatically allocates an address for the `HDMI_AUDIO` stream.

Note: This command applies only to transmitter devices.

5.16.1.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied.

Refer to section 5.5 Command get for details.

5.16.1.6 Errors

ILLEGAL_ARGUMENT	Occurs when the target argument is not a valid device ID or group name.
ILLEGAL_ARGUMENT	Or if the target argument is not a valid device ID and a multicast IP address was specified.
ILLEGAL_ARGUMENT	In addition, occurs if no stream type or index was specified but a multicast IP address was specified.
ILLEGAL_ARGUMENT	Also, if a multicast IP address outside the configurable allocation range was specified.
ILLEGAL_ARGUMENT	Occurs also if a reserved multicast IP address was specified.

The operation may also fail for devices individually for the following reasons:

DEVICE_TYPE	if the device does not have the specified stream type or index
DEVICE_TYPE	if the device is a receiver

INVALID_STATE	if the specified multicast IP address is in use by another stream
INVALID_STATE	if no more multicast IP addresses are available
INVALID_STATE	if starting a scaled video stream but the scaler resolution has not been set

5.16.1.7 Example

```
start 001ec0f04d9c:HDMI:0
{
   "status" : "PROCESSING",
   "request_id" : 61,
   "result" : null,
   "error" : null
}
```

5.17 Command stop

5.17.1 stop

5.17.1.1 Command usage

`stop` *`target:`*`[`*`stream_type`*`[`*`:index`*`]] [free]`

5.17.1.2 Description

Stops one or more stream(s) on one or more device(s), and optionally free the multicast address associated with each stopped stream so it can be re-used for another stream.

5.17.1.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **stream_type** argument must be a stream type. The following types are supported for all device types:

RS232	For an RS-232 data stream.
INFRARED	For an infrared data stream.

Furthermore, the following types are supported on **transmitter** devices only, target argument cannot be a receiver device or the ALL_RX group:

`AUDIO`	For an analog audio stream.
`HDMI`	For an HDMI video stream.
`HDMI_AUDIO`	For an HDMI audio stream.
`I2S_AUDIO`	For an I2S audio stream.

The **index** argument must the valid index of a stream of the specified type.

If **index** is omitted, all streams of the given type are stopped. If **stream_type** is omitted, all streams of the following types are stopped:

`HDMI`	For an HDMI video stream.
`HDMI_AUDIO`	For an HDMI audio stream.
`AUDIO`	For an analog audio stream.
`I2S_AUDIO`	For an I2S audio stream.

If **stream_type** is omitted, the **target** argument cannot be a receiver device or the ALL_RX group.

If the **free** keyword is added to the command, the multicast IP address associated with each target stream is freed. For streams with type `RS232` or `INFRARED` where no multicast IP address is associated with the stream, the **free** keyword is allowed but ignored.

5.17.1.4 Notes

When stopping `HDMI` stream `0`, `HDMI_AUDIO` stream `0` from the same transmitter is also implicitly stopped unless independent routing of HDMI audio is enabled.

Refer to section 5.10.3 mode split_hdmi_audio for details.

The **free** keyword has been added in API 2.5.

5.17.1.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied.

Refer to section 5.5 Command get for details.

5.17.1.6 Errors

ILLEGAL_ARGUMENT Occurs when the target argument is not a valid device ID or group name.

The operation may also fail for devices individually for the following reasons:

DEVICE_TYPE Occurs when the device does not have the specified stream type or index.
DEVICE_TYPE Occurs if the stream_type argument is omitted and the device is a receiver.

5.17.1.7 Example

```
stop 001ec0f04d9c:HDMI:0
{
   "status" : "PROCESSING",
   "request_id" : 61,
   "result" : null,
   "error" : null
}
```

5.18 Command switch

5.18.1 switch

5.18.1.1 Command usage

`switch source:stream_type:index_tx destination[:subscription_type]:index_rx`

`switch source:stream_type:index_tx api`

源地址 流类型 流索引号

`switch source:stream_type:index_tx dest_ip_address`

5.18.1.2 Description

设置设备的目的流

Set stream destination for a device.

建立 单向

The `switch` command is used to establish a one-way connection between source and destination devices. To establish a two-way connection between any two devices, the command must be issued twice with source and destination reversed.

颠倒

The destination is specified in one of three ways, hence the three forms of the command:

使用设备的ID号和流的索引号当目的是一个设备的时候

- Using a device ID and a stream index (first form) when the destination is a device.
- Using the keyword **api** (second form) when the destination is the BlueRiver Control server.

当目的是一个BlueRiver控制服务器的时候使用api

使用IP地址

- Using an IP address (third form).

5.18.1.3 Arguments

The **source** argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **stream_type** argument must be a stream type. The following types are supported:

`CEC`	For a CEC data stream.
`RS232`	For an RS-232 data stream.
`INFRARED`	For an infrared data stream.

Note: For other stream types, refer to the `start` command (section 5.16).

The **index_tx** argument must the valid index of a stream of the specified type on the source device.

In the first form of the command, the **destination** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **subscription_type** argument is optional. If specified, it must be identical to the **stream_type** argument (a stream can only be switched to a subscription of the same type).

The **index_rx** argument must be the valid index of a subscription of the specified type on the destination device. For RS-232 data stream (stream type `RS232`), the **index_rx** argument must match the **index_tx** argument.

In the third form of the command, the **dest_ip_address** must be the IP address to which the data is to be sent.

5.18.1.4 Notes

A device cannot send data to itself.

The second form of the command with the **api** keyword was introduced in API 2.12. It its use requires firmware support (3.4.0.0 or later).

5.18.1.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the source right after the new settings were applied.

Refer to section 5.5 Command get for details.

5.18.1.6 Errors

`ILLEGAL_ARGUMENT`	Occurs when the arguments entered are invalid.
`INVALID_STATE`	This occurs if the destination device is disconnected.

The request may also fail for individual devices for the following reasons:

`INVALID_OPERATION`	Occurs if the device is being configured to send data to itself.
`DEVICE_TYPE`	If the second form of the command is used and the firmware is not recent enough to support it (3.4.0.0 or later).

5.18.1.7 Example

Routing the third RS-232 port from a device to another:

```
switch 001ec0f03668:RS232:2 001ec0f04d9c:2
{
   "status" : "PROCESSING",
   "request_id" : 61,
   "result" : null,
   "error" : null
}
```

Routing the first RS-232 port from a device to the BlueRiver Control Server:

```
switch 001ec0f04d9c:RS232:0 169.254.1.0
{
   "status" : "PROCESSING",
   "request_id" : 62,
   "result" : null,
   "error" : null
}
```

Routing infrared data between two devices:

```
switch 001ec0f03668:INFRARED:0 001ec0f04d9c:0
{
   "status" : "PROCESSING",
   "request_id" : 63,
   "result" : null,
   "error" : null
}
```

5.19 Command test

The test command provides sub-commands to test the proper operation of one or more device(s).

5.19.1 test memory

5.19.1.1 Command usage

`test` *`target`* `memory`

5.19.1.2 Description

Tests the framebuffer memory of one or more devices.

5.19.1.3 Arguments

The **target** argument must be either the device ID of a single device, or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

5.19.1.4 Notes

This command was introduced in API version 2.1.

It is possible to run a memory test on a transmitter device(s) (including the ALL_TX group) since API 2.10.

5.19.1.5 Return Value

For each device targeted by the **test memory** command, there is either an entry in the `device_test` array indicating the test was performed on the device or a device error object in the error array if the test could not be successfully performed.

```
{
        "device_test" : [TestResultObject, ...],
        "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
device_test	2.2	Array of TestResultObject	An array of objects which represent the result of the tests.
error	2.2	Array of DeviceError	An array of DeviceError objects indicating the devices for which the command failed and the reason why it failed

The `TestResultObject` object has the following structure:

```
{
        "device_id" : String,
        "memory" : String
}
```

Member name	Since	Type	Description
device_id	2.2	String	Device ID
memory	2.2	String	A string which represents result of memory test. One of the following: `PASS` The memory test was successful `FAIL` The memory test failed The following is reserved for future use: `SKIP`

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

5.19.1.6 Errors

ILLEGAL_ARGUMENT if the arguments are invalid

The operation may also fail for devices individually for the following reason:

DEVICE_TYPE	if the device does not have a framebuffer
DEVICE_TYPE	if the device firmware does not support the memory test (transmitter devices prior to 3.3.0.0)

5.19.1.7 Example

```
test 001ec0f04d9c memory
{
   "status" : "PROCESSING",
   "request_id" : 32170,
   "result" : null,
   "error" : null
}
request 32170
{
   "status" : "SUCCESS",
   "request_id" : 32170,
```

```
        "result" : {
           "device_test" : [
              {
                 "device_id" : "001ec0f04d9c",
                 "memory" : "PASS"
              }
           ],
           "error" : []
        },
        "error" : null
     }
```

5.20 Command update

5.20.1 update

5.20.1.1 Command usage

update ***target file_name***

5.20.1.2 Description

Update the BlueRiver chipset firmware of one or more devices.

While a firmware update is in progress, no other operation can be performed on the device. For this reason, any command sent to the BlueRiver Control Server that targets the device will fail with `INVALID_STATE` for this device until the firmware update completes.

All connected clients are notified of the start and end of a firmware update with a `FIRMWARE_UPDATE_START` event and a `FIRMWARE_UPDATE_COMPLETE` event, respectively.

A reboot command must be issued to the device once the firmware update is complete for the changes to take effect. The device is in an undefined state until that reboot.

5.20.1.3 Arguments

The **target** argument must be either the device ID of a single device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices
`ALL_TX`	All transmitter devices
`ALL_RX`	All receiver devices

The **file_name** argument is the file name of a firmware file. The available firmware files can be listed with the list firmware command (refer to section 5.9.1 list firmware.

5.20.1.4 Notes

This command was introduced in API version 2.2.

5.20.1.5 Return Value

For each device targeted by the update command, there is either an entry in the update array if the device was updated successfully or a device error object in the error array if the operation failed for some reason.

```
{
      "update" : [UpdateResultObject, ...],
      "error" : [DeviceError, ...]
}
```

Member name	Since	Type	Description
update	2.2	Array of UpdateResultObject	An array of objects which represent the devices for which the firmware update succeeded.
error	2.2	Array of DeviceError	An array of DeviceError objects indicating the devices for which the command failed and the reason why it failed

The UpdateResultObject object has the following structure:

```
{
      "device_id" : String
}
```

Member name	Since	Type	Description
device_id	2.2	String	Device ID

Refer to section 2 Commands Overview for a description of the device error object (DeviceError).

5.20.1.6 Errors

ILLEGAL_ARGUMENT	Occurs if the target argument entered is not a valid device ID or group name.
IO_ERROR	Occurs if the firmware file could not be opened.

The request may also fail for individual devices for the following reasons:

IO_ERROR	Occurs if the firmware file could not be read.
RESOURCE_NOT_FOUND	Occurs if specified firmware file does not contain a firmware for the device.

5.20.1.7 Example

```
update 001ec0f04d9c semtech_blueriver_3.5.0.0-evaluation.apz
{
   "status" : "PROCESSING",
   "request_id" : 32223,
   "result" : null,
   "error" : null
}

request 32223
{
   "status" : "SUCCESS",
   "request_id" : 32223,
   "result" : {
      "update" : [
         {
            "device_id" : "001ec0f04d9c"
         }
      ],
      "error" : []
   },
   "error" : null
}
```

6 Module *multiview* Version 1.1 Command Reference

6.1 Introduction

This section describes the multiview command module version 1.1 which contains a set of commands that allows the use of the multiview capabilities of devices that support them.

Before the commands described in this section can be called, the command module must be loaded with the `require` command:

```
require multiview 1.1.0
```

Refer to section 4.3Command require for details.

6.2 Multiview Model

6.2.1 Overview

In multiview display mode, the screen is divided in a grid of at most 16×16 rectangular tiles. Each tile displays video from a rectangular area of the receiver device's frame buffer. Alternatively, a tile can also be black.

To use the multiview feature, a layout is first created and configured. The layout is then applied to one or more receiver device(s).

6.2.2 Concepts and Terminology

A **surface** is a rectangular area in a receiver device's frame buffer that is allocated for a specific video source. Each BlueRiver NT2000 receiver device supports up to 32 surfaces and each surface is associated with the HDMI subscription that has the same index (e.g. the video from HDMI subscription 14 goes into surface 14).

The screen is split into a grid of **tiles**, where a tile is the smallest unit on the screen to which content can be assigned. Each tile either contains a rectangular area of a surface or is black.

A **window**, is a rectangular area on the screen that either maps to a rectangular area inside a surface or is black. Windows can overlap, in which case the content of the window with the smallest index is shown (e.g. window 5 is in front of window 6). Multiple tiles may be required to represent a single window (when windows overlap) and the black background that is not part of any window.

A **layout** represents a full set of a multiview configuration that can be applied to a receiver device. A layout has a name that is used to refer to it by the various commands and an output resolution. Each layout also has a set of surfaces and a set of windows.

The example figure provided below illustrates some of the above concepts using a layout on a 5×5 grid of tiles. The layout shown has two windows:

1. One with the car; and
2. One with the skier.

Four tiles form the car window (tiles 7, 8, 12 and 13), three tiles form the visible portion of the skier window (tiles 14, 18 and 19) and the others are part of the black background.

The car window has a lower index than the skier window because it is on top (e.g. the car window could be window 0 and the skier window could be window 1). The car window maps to a surface with the car video and the skier window maps to a surface that has the skier video.

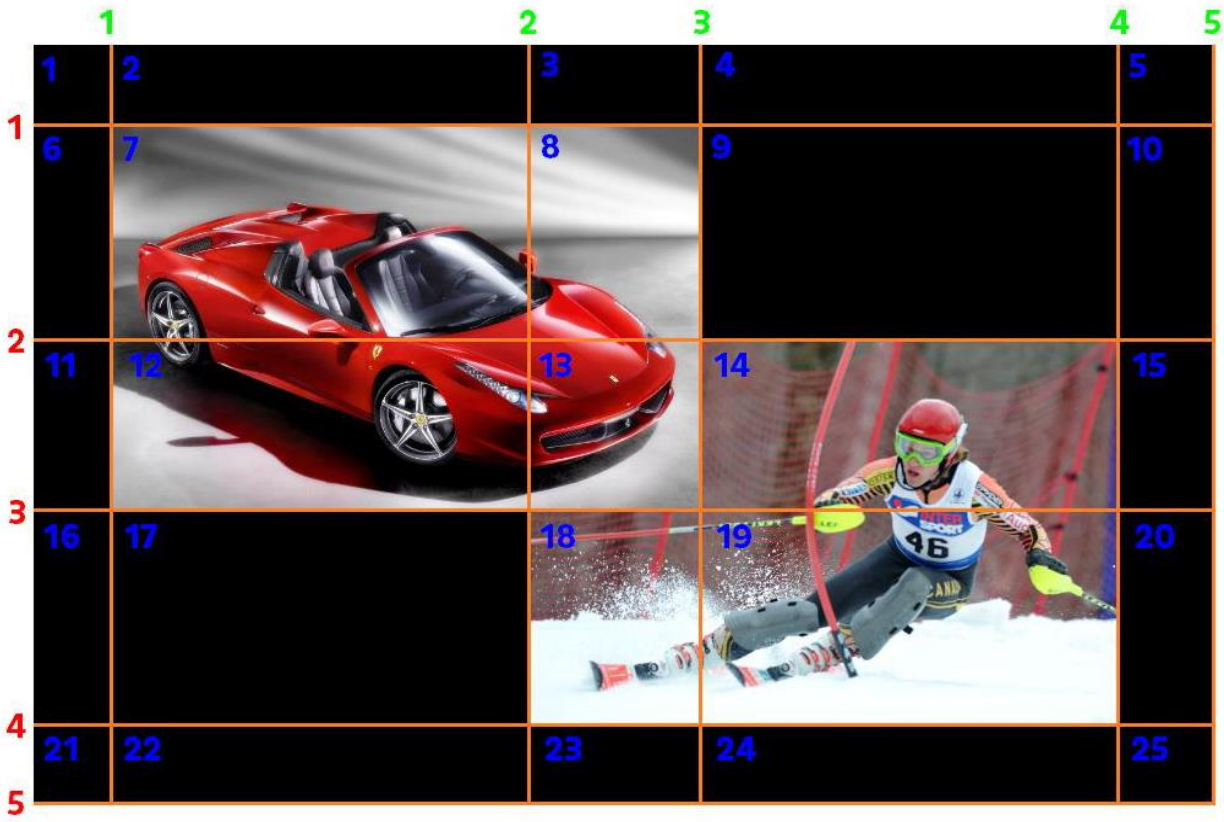

Figure 1: Example of a 5x5 multiview grid

6.2.3 Limitations

The limitations that apply to multiview layouts are as follow:

- The number of windows must not exceed 32.
- The number of distinct video sources (i.e. surfaces) cannot exceed 32.
- The number of tiles cannot exceed 15 in either the horizontal or vertical direction, and 16 in the other direction (i.e. a grid of 15×16 or 16×15 tiles are both acceptable but 16×16 is not).
- Tile widths and horizontal read offsets (equivalently window widths, horizontal positions and horizontal offsets) must be even.
- The total bandwidth sent by an individual transmitter and received by an individual receiver must not exceed 9Gbit/s.
- Source video must be progressive 4:4:4.
- Surfaces must be horizontally aligned on a 32-pixel boundary and separated by a 32-pixel margin.
- The total size of the frame buffer is 10880×4096.

6.2.4 Predefined Compatibility Layout

The API provides a predefined layout named **compatibility_4k_2x2**.

As the name indicates, this layout is a 4K (3840×2160) layout with four 1080p sources arranged in a 2×2 grid. This layout emulates the legacy multiview behaviour of BlueRiver NT devices on the BlueRiver NT2000 devices.

The **compatibility_4k_2x2** layout can be applied to legacy BlueRiver NT and BlueRiver NT2000 devices using the `set multiview` command (refer to section 6.6.1 set multiview) and is the only layout that can be applied to BlueRiver NT devices.

Comment: The **compatibility_4k_2x2** layout cannot be modified. Trying to do so with the `layout` command fails with a `MULTIVIEW_LAYOUT_STATE` error.

6.2.5 Default Surface Configuration

For a specific multiview layout, the allocation of surfaces in a device's frame buffer can be customized with the `layout surface` command.

However, when a layout is first created (or reset) using the `layout create` command, a default surface configuration is applied. This default configuration, which is sufficient for most purposes, is as follows:

Surface 0 can receive video up to a resolution of 4096×2160.

Surfaces 1 to 11 can receive video up to a resolution of 2048×1080.

Surfaces 12 to 16 can receive video up to a resolution of 2048×856.

Surfaces 17 to 20 can receive video up to a resolution of 2048×1080.

- However, surfaces 17 to 20 use the same area of the frame buffer as surface 0.
- This means surfaces 17 to 20 can be used *instead* of surface 0 if the multiview layout does not contain a source with a resolution over 2048×1080, but they cannot be used at the same time as surface 0.

6.3 Multiview Example

Load the **blueriver_api** and the **multiview** command modules:

```
require blueriver_api 2.6.0
require multiview 1.1.0
```

Create a picture-in-picture multiview layout with a full-screen 4K window and a smaller second window overlaid over the first:

```
layout picture_in_picture create size 3840 2160
layout picture_in_picture window 0 position 2752 128 size 960 540 target 1
layout picture_in_picture window 1 position 0 0 size 3840 2160 target 0
```

Start the full-size video stream on the first source device:

```
start d88039634e85:HDMI:0
```

Set the scaler video resolution and start the scaled video stream on the second source device:

```
set d8803962e132 scaler size 960 540
start d8803962e132:HDMI:1
```

Apply the multiview layout to the receiver device:

```
set d88039631d77 multiview picture_in_picture fps 30
```

Join both source streams to the receiver devices' subscriptions:

```
join d88039634e85:HDMI:0 d88039631d77:0
join d8803962e132:HDMI:1 d88039631d77:1
```

6.4 Command layout

The list command provides sub-commands to create and modify multiview layouts.

6.4.1 layout create

6.4.1.1 Command usage

`layout` *`name`* `create size` *`width height`*

6.4.1.2 Description

Create a new multiview layout or reset an existing layout with the specified name.

6.4.1.3 Arguments

The **name** argument is the name of the multiview layout.

The **width** and **height** arguments specify respectively the total width and height of the displayed video in pixels.

Note: The **width** argument must be a multiple of 32.

6.4.1.4 Return Value

None

6.4.1.5 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid.
`MULTIVIEW_LAYOUT_STATE` Occurs when the specified layout exists and is read-only.

6.4.1.6 Example

```
layout some_layout create size 1920 1080
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

6.4.2 layout delete

6.4.2.1 Command usage

`layout` *`name`* `delete`

6.4.2.2 Description

Delete the multiview layout with the specified name if it exists.

6.4.2.3 Arguments

The **name** argument is the name of the multiview layout to be deleted.

6.4.2.4 Return Value

None

6.4.2.5 Errors

`ILLEGAL_ARGUMENT` Occurs when if the arguments entered are invalid.
`MULTIVIEW_LAYOUT_STATE` Occurs if the specified layout is read-only.

6.4.2.6 Example

```
layout some_layout delete
{
```

```
    "status" : "SUCCESS",
    "request_id" : null,
    "result" : null,
    "error" : null
}
```

6.4.3 layout describe

6.4.3.1 Command usage

`layout name describe`

6.4.3.2 Description

Provide a complete description of the multiview layout with the specified name.

6.4.3.3 Arguments

The **name** argument is the name of the multiview layout.

6.4.3.4 Return Value

```
{
        "layout_description" : LayoutDescription
}
```

Member name	Since	Type	Description
layout_description	1.1	LayoutDescription	A layout description object, described below

A layout description object (`LayoutDescription`) has the following structure:

```
{
        "name" : String
        "width" : Integer,
        "height" : Integer,
        "read_only" : Boolean,
        "surfaces" : [SurfaceDescription, ...],
        "windows" : [WindowDescription, ...]
}
```

Member name	Since	Type	Description
name	1.1	String	The layout name
width	1.1	Integer	Display width in pixels
height	1.1	Integer	Display height in pixels
read_only	1.1	Boolean	Whether a layout is protected against modifications. Currently, only predefined **compatibility_4k_2x2** layout is read-only.
surfaces	1.1	Array of SurfaceDescription	Description of all surfaces in this layout.
windows	1.1	Array of WindowDescription	Description of all windows in this surface.

A surface description object (`SurfaceDescription`) has the following structure:

```
{
        "index" : Integer,
        "horizontal_position" : Integer,
        "vertical_position" : Integer,
        "width" : Integer,
```

```
        "height" : Integer
    }
```

Member name	Since	Type	Description
index	1.1	Integer	The surface index
horizontal_position	1.1	Integer	Horizontal start position of surface in pixels
vertical_position	1.1	Integer	Vertical start position of surface in pixels
width	1.1	Integer	Surface width in pixels
height	1.1	Integer	Surface height in pixels

A window description object (`WindowDescription`) has the following structure:

```
{
        "index" : Integer,
        "horizontal_position" : Integer,
        "vertical_position" : Integer,
        "width" : Integer,
        "height" : Integer,
        "horizontal_offset" : Integer,
        "vertical_offset" : Integer,
        "content" : String,
        "target_surface" : Integer,
}
```

Member name	Since	Type	Description
index	1.1	Integer	The surface index
horizontal_position	1.1	Integer	Horizontal start position of window on screen, in pixels
vertical_position	1.1	Integer	Vertical start position of window on screen, in pixels
width	1.1	Integer	Window width in pixels
height	1.1	Integer	Window height in pixels
horizontal_offset	1.1	Integer	Horizontal offset from the start of the surface
vertical_offset	1.1	Integer	Vertical offset from the start of the surface
content	1.1	String	Type of content of this window, either: • `VIDEO` window has video content • `BLACK` window is black
target_surface	1.1	Integer	Index of the surface to which this window refers. This member is only meaningful if the content member is VIDEO.

6.4.3.5 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid.
`RESOURCE_NOT_FOUND` Occurs when the specified layout does not exist.

6.4.3.6 Example

See B.1 Command “layout describe” for a complete example.

6.4.4 layout surface

6.4.4.1 Command usage

layout *name* surface *index* (position *horiz_position vert_position* size *width height*|*delete*)

6.4.4.2 Description

Create, modify or delete a surface in the specified layout.

6.4.4.3 Arguments

The **name** argument is the name of the multiview layout.

The **index** argument is the index of the surface, which must be between 0 and 31.

The **horiz_position** and **vert_position** arguments are respectively the horizontal and vertical start positions of the surface in the frame buffer, in pixels. The **horiz_position** argument must be a multiple of 32 pixels.

The **width** and **height** arguments are respectively the width and height of the surface in pixels.

Note: The **width** argument must be a multiple of 32 pixels.

If the **delete** keyword is present, the surface is deleted if it exists.

6.4.4.4 Return Value

None

6.4.4.5 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid.
`MULTIVIEW_LAYOUT_STATE` Occurs when the specified layout exists but is read-only.
`RESOURCE_NOT_FOUND` Occurs when the specified layout does not exist.

6.4.4.6 Examples

```
layout some_layout surface 23 position 3872 0 size 1920 1080
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
layout some_layout surface 23 delete
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}
```

6.4.5 layout window

6.4.5.1 Command usage

layout *name* window *index* ([position *horiz_position vert_position* size *width height*] [offset *horiz_offset vert_offset*] [target (*surface_index*|black)]|*delete*)

6.4.5.2 Description

Create, modify or delete a window in the specified layout.

6.4.5.3 Arguments

The **name** argument is the name of the multiview layout.

The **index** argument is the index of the window, which must be between 0 and 31.

The **horiz_position** and **vert_position** arguments are respectively the horizontal and vertical start positions of the window on the screen, in pixels. The **horiz_position** argument must be a multiple of 32 pixels.

The **width** and **height** arguments are respectively the width and height of the window in pixels.

Note: The **width** argument must be a multiple of 32 pixels.

The **horiz_offset** and **vert_ offset** arguments are respectively the horizontal and vertical start positions of the window relative to the start of the target surface, in pixels. The **horiz_ offset** argument must be a multiple of 32 pixels.

The **surface_index** argument is the index of the surface to which the window refers. If the **black** keyword is used instead of a surface index, the window is black.

Parameters associated with omitted arguments are left unchanged, which is useful to selectively modify parameters of an existing window. All arguments should be present when creating a new window, with the exception of the offset argument which can be assumed to be zero (horizontal and vertical) when a window is first created.

If the **delete** keyword is present, the surface is deleted if it exists.

6.4.5.4 Return Value

None

6.4.5.5 Errors

`ILLEGAL_ARGUMENT` Occurs when the arguments entered are invalid.
`MULTIVIEW_LAYOUT_STATE` Occurs when the specified layout exists but is read-only.
`RESOURCE_NOT_FOUND` Occurs if the specified layout does not exist.

6.4.5.6 Examples

```
layout some_layout window 0 position 480 270 size 1920 1080 target 3
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}

layout some_layout window 0 target black
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
}

layout some_layout window 0 delete
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : null,
   "error" : null
```

```
}
```

6.5 Command list

The list command provides sub-commands to query lists from the BlueRiver Control Server (API).

6.5.1 list layout

6.5.1.1 Command usage

```
list layout
```

6.5.1.2 Description

List all multiview layouts available on the BlueRiver Control Server (API server).

6.5.1.3 Return Value

```
{
        "layout" : [LayoutItemObject, ...]
}
```

Member name	Since	Type	Description
layout	1.1	Array of LayoutItemObject	An array of objects which each represent a multiview layout

Each item in the layout array (LayoutItemObject) has the following structure:

```
{
        "name" : String
}
```

Member name	Since	Type	Description
name	1.1	String	The layout name

6.5.1.4 Example

```
list layout
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "layout" : [
         {
            "name" : "compatibility_4k_2x2"
         },
         {
            "name" : "example_4k_single_source_2x2_1080p"
         },
         {
            "name" :
"example_4k_three_sources_1080p_half_1080p_640x480"
         }
      ]
   }
}
```

6.6 Command set

6.6.1 set multiview

6.6.1.1 Command usage

`set` *`target`* `multiview` *`layout_name`* `[`*`video_args`*`]`

6.6.1.2 Description

Set one or more receiver device(s) in multiview mode and apply a specified multiview layout.

6.6.1.3 Arguments

The **target** argument must be either the device ID of a single receiver device or the name of a device group. The allowed device groups are the following:

`ALL`	All devices.
`ALL_RX`	All transmitter devices.

The **layout_name** argument specifies the multiview layout to apply.

The **video_args** argument specifies video format arguments. See section 5.2 Specifying a Video Output Format for details. The display mode is implicitly set to multiview mode by this command.

6.6.1.4 Notes

This command applies only to receiver devices.

BlueRiver NT devices have only limited multiview capabilities: the predefined **compatibility_4k_2x2** layout is the only one that is supported by these devices. Refer to section 6.2.4 Predefined Compatibility Layout for details.

6.6.1.5 Return Value

The return value is the same as if the "`get settings`" command had been called on the same target right after the new settings were applied.

Refer to section 5.5 Command get for details.

6.6.1.6 Errors

`ILLEGAL_ARGUMENT`	Occurs when target argument entered is not valid device ID or group name.
`RESOURCE_NOT_FOUND`	Occurs when the specified layout does not exist.
`MULTIVIEW_LAYOUT_STATE`	Occurs if the layout does not satisfy hardware or API limitations such as tile size or number of tiles (see section 6.2.3 Limitations).

The request may also fail for individual devices for the following reasons:

`DEVICE_TYPE`	This error occurs if the device is not a receiver device.
`DEVICE_TYPE`	Also occurs when the device does not support the full multiview capabilities (i.e. BlueRiver NT device) and the specified layout is not compatibility_4k_2x2.

6.6.1.7 Example

```
set d88039631d77 multiview compatibility_4k_2x2 fps 60
{
   "status" : "PROCESSING",
   "request_id" : 13465,
   "result" : null,
   "error" : null
```

```
}
```

7 Device Object Definition (DeviceObject)

The device object completely describes a device, including its configuration, state and supported features. A subset of this object is returned as part of the response to the `get` command (refer to the `get` command description for details in section 5.5 Command get).

```
{
   "device_id" : String,
   "identity" : DeviceIdentity,
   "configuration" : DeviceConfig,
   "status" : DeviceStatus
   "streams" : [StreamObject, ...],
   "subscriptions" : [SubscriptionObject, ...]
   "nodes" : [Node, ...]
}
```

Member name	Since	Type	Included in	Description
device_id	2.0	String	(always)	Device ID
identity	2.0	DeviceIdentity	get identity, get settings	Provides information regarding the identity of the device.
configuration	2.0	DeviceConfig	get identity, get settings	Provides information regarding configurable settings of the device.
status	2.0	DeviceStatus	get settings	Provides information regarding the current status of the device.
streams	2.0	Array of StreamObject	get settings	Describes the streams of data sent on the network.
subscriptions	2.0	Array of SubscriptionObject	get settings	Describes the streams of data received from the network.

nodes	2.0	Array of Node	get settings, get edid, get gpio	Describes the hardware modules present on the device along with their status and configuration.
				Note: This array is included in the response to the **get settings**, **get edid** and **get gpio** command.
				However, in the response to a **get edid** command, only nodes of the following type are included:
				`HDMI_DECODER`
				`HDMI_MONITOR`
				This is because those are the nodes that contain EDID information.
				Furthermore, in the response to a **get gpio** command, only the GPIO nodes are included, while these same GPIO nodes are not included in the response to a **get settings** command.

7.1 Generic Types

These generic JSON types are re-used in different parts of the device object.

7.1.1 Video Format Description (VideoFormat)

An object describing a video format including resolution, frame rate, color space etc.

```
{
   "bits_per_pixel" : Integer,
   "color_space" : String,
   "frames_per_second" : Integer,
   "height" : Integer,
   "scan_mode" : String,
   "width" : Integer
}
```

Member name	Since	Type	Description
bits_per_pixel	2.0	Integer	Number of bits per video component, one of: • `8 bits` • `10 bits` • `12 bits`
color_space	2.0	String	Color space, which is one of the following: • `RGB` for RGB graphics • `YCBCR_444` for YcbCr 4:4:4 video • `YCBCR_422` for YcbCr 4:2:2 video • `YCBCR_420` for YcbCr 4:2:0 video
frames_per_second	2.0	Integer	Frame rate Note that the value of this member is always an integer, so e.g. 59.94 frames/second is reported as 60. For more precision, consider using the **frame_rate** member of the VideoFormatDetails object instead.
Height	2.0	Integer	Video height
scan_mode	2.2	String	Video scan mode, which is one of the following: • `PROGRESSIVE` for progressive video • `INTERLACED` for interlaced video
Width	2.0	Integer	Video width

7.1.2 Video Format Detailed Information (VideoFormatDetails)

An object containing detailed information about a video format. Some information is taken from the Auxiliary Video Information (AVI) InfoFrame: for more information regarding the meaning of these members refer to CEA-861-F in section 6.4 “Format of Version 2 & 3 AVI InfoFrames”.

```
{
    "frame_rate" : Real
    "pixel_clock" : Integer,
    "total_width" : Integer,
    "total_height" : Integer,
    "hsync_width" : Integer,
    "hsync_front_porch" : Integer,
```

```
        "hsync_negative" : Boolean,
        "vsync_width" : Integer,
        "vsync_front_porch" : Integer,
        "vsync_negative" : Boolean,
        "vic" : Integer,
        "has_hdmi_vic" : Boolean,
        "hdmi_vic" : Integer,
        "picture_aspect" : String,
        "has_active_format" : Boolean,
        "active_format" : Integer,
        "it_content_type" : String,
        "scan_information" : String,
        "colorimetry" : String,
        "rgb_range" : String,
        "ycc_range" : String,
}
```

Member name	Since	Type	Description
frame_rate	2.13	Real	Frame rate. Includes the fractional part (e.g. 59.54).
pixel_clock	2.8	Integer	Pixel clock in Hz
total_width	2.8	Integer	Total horizontal resolution including blanking
total_height	2.8	Integer	Total vertical resolution including blanking
hsync_width	2.8	Integer	Horizontal sync pulse width in pixels
hsync_front_porch	2.8	Integer	Horizontal sync front porch in pixels
hsync_negative	2.8	Boolean	Whether the horizontal sync has negative polarity
vsync_width	2.8	Integer	Vertical sync pulse width in pixels
vsync_front_porch	2.8	Integer	Vertical sync front porch in pixels
vsync_negative	2.8	Boolean	Whether the vertical sync has negative polarity
vic	2.8	Integer	AVI InfoFrame Video Information Code (VIC)
has_hdmi_vic	2.8	Boolean	Whether there is an HDMI Vendor-Specific InfoFrame with an HDMI_VIC.
hdmi_vic	2.8	Integer	HDMI Vendor-Specific Infoframe HDMI_VIC. Undefined if **has_hdmi_vic** is false.
picture_aspect	2.8	String	AVI InfoFrame picture aspect ratio: • `ASPECT_NO_DATA` for No Data • `ASPECT_4_3` for a 4:3 aspect ratio • `ASPECT_16_9` for a 16:9 aspect ratio
has_active_format	2.8	Boolean	Whether the AVI InfoFrame specifies active format information.
active_format	2.8	Integer	AVI InfoFrame Active Format Description (AFD) code. See CEA-861-F Annex H.
it_content_type	2.8	String	AVI InfoFrame IT content type: • `NO_DATA` for No Data • `GRAPHICS` for Graphics type • `PHOTO` for Photo type • `CINEMA` for Cinema type • `GAME` for Game type

Member name	Since	Type	Description
scan_information	2.8	String	AVI InfoFrame scan information: • `NO_DATA` for No Data • `OVERSCAN` for overscan • `UNDERSCAN` for underscan
colorimetry	2.8	String	AVI InfoFrame colorimetry: • `NO_DATA` for No Data • `RGB` for RGB • `BT601` for SMPTE 170M/ITU-R BT.601 • `BT709` for ITU-R BT.709 • `XVYCC601` for $xvYCC_{601}$ • `XVYCC709` for $xvYCC_{709}$ • `SYCC601` for $sYCC_{601}$ • `ADOBE_YCC601` for $Adobe_{YCC601}$ • `ADOBE_RGB` for $Adobe_{RGB}$ • `BT2020_YCC_CONST` for ITU-R BT.2020 $Y'C'_BC'_{RC}$ • `BT2020_YCC` for ITU-R BT.2020 $Y'C'_BC'_R$ • `BT2020_RGB` for ITU-R BT.2020 R'G'B' This member combines information from the "RGB or YC_BC_R" (Y), Colorimetry (C) and Extended Colorimetry (EC) fields of the AVI InfoFrame. See CEA-861-F Table 13.
rgb_range	2.8	String	AVI InfoFrame RGB quantization range: • `DEFAULT` • `LIMITED` • `FULL`
ycc_range	2.8	String	AVI InfoFrame YCbCr quantization range: • `LIMITED` • `FULL`

7.1.3 Audio Format Detailed Information (AudioFormatDetails)

An object containing information about an audio format.

```
{
   "sampling_frequency" : Integer,
   "number_of_channels" : Integer
}
```

Member name	Since	Type	Description
sampling_frequency	2.10	Integer	The sampling frequency in Hz
number_of_channels	2.10	Integer	The number of audio channels

7.1.4 Source Reference Specification (SourceReference)

This object type is used in both streams and nodes to specify the origin of the data.

```
{
   "ref_class" : String,
   "ref_type" : String,
   "ref_index" : Integer
}
```

Member name	Since	Type	Description
ref_class	2.0	String	The reference class, which is one of the following: • NODE The origin of the data is a node. • SUBSCRIPTION The origin of the data is a subscription. In addition, the following value might be present in configuration choices of a node input: • AUTOMATIC The actual source is automatically selected, for example depending on the presence of a valid signal
ref_type	2.0	String	Reference type, matches the **type** member of a node or subscription. Undefined if reference class is AUTOMATIC.
ref_index	2.0	Integer	Reference index, matches the index member of a node or subscription. Undefined if reference class is AUTOMATIC.

7.1.5 Source Selection (SourceSelection)

This object type is used in both streams and nodes to specify the origin of the data.

```
{
   "value" : Integer,
   "choices" : [SourceSelectionChoice, ...]
}
```

Member name	Since	Type	Description
value	2.0	Integer	Selected source, must match the **value** member of one of the enumerated choices.
choices	2.0	Array of SourceSelectionChoice	Enumeration of the allowed source selections. **Note:** This member is not included in the response of a `get settings` command.

7.1.6 Source Selection Choice (SourceSelectionChoice)

This object type is used in both streams and nodes to specify the origin of the data.

```
{
   "value" : Integer,
   "description" : String
   "details" : SourceReference
}
```

Member name	Since	Type	Description
value	2.0	Integer	Value of this choice.
description	2.0	String	Human-readable description of this choice.
details	2.0	SourceReference	Source reference associated with this choice.

7.2 Device Identity (DeviceIdentity)

This object type holds information that can be used for device identification.

```
{
   "engine" : String,
   "vendor_id" : Integer,
   "product_id" : Integer,
   "firmware_comment" : String,
```

```
    "firmware_version" : String,
    "firmware_rc" : String,
    "is_receiver" : Boolean,
    "is_transmitter" : Boolean,
    "firmware_details" : [FirmwareDetails]
}
```

Member name	Since	Type	Included in	Description
engine	2.2	String	get identity, get settings	Identifies the type of the device chipset: • `BLUERIVER_NT` for BlueRiver NT chipset. • `PLETHORA` for the chipset of NT1000/NT2000 devices.
vendor_id	2.12	Integer	get identity, get settings	Device vendor ID. This feature requires firmware support. If such support is not available, the reserved value 0 is reported.
product_id	2.12	Integer	get identity, get settings	Device product ID. This feature requires firmware support. If such support is not available, the reserved value 0 is reported.
firmware_comment	2.12	String	get identity, get settings	Comment string embedded in the firmware file that was programmed in the device. This comment typically contains a version string but the exact format and content is device manufacturer-specific. May contain an empty string if no comment is set.
firmware_version	2.0	String	get hello, get identity, get settings	Device firmware version
firmware_rc	2.6	String	get hello, get identity, get settings	Device firmware release candidate **Note:** For Semtech AptoVision Product Group internal use only.
is_receiver	2.2	Boolean	get hello, get identity, get settings	True is the device is a receiver device
is_transmitter	2.2	Boolean	get hello, get identity, get settings	True is the device is a transmitter device
firmware_details	2.13	Array of FirmwareDetails	get identity, get settings	Provide information regarding the various firmware images of the device.

A FirmwareDetails object has the following structure:

```
{
    "type" : String,
```

```
    "details_available" : Boolean,
    "firmware_version" : String,
    "firmware_rc" : String
}
```

Member name	Since	Type	Description
type	2.13	String	The type of the firmware image being described. One of the following: • PRIMARY for the primary firmware image. • GOLDEN for the golden firmware image.
details_available	2.13	Boolean	Whether information is available for this firmware image.
firmware_version	2.13	String	Version of this firmware image. If the value of the **details_available** member is false, this value is undefined.
firmware_rc	2.13	String	Release candidate number of this firmware image. If the value of the **details_available** member is false, this value is undefined. **Note:** For Semtech AptoVision Product Group internal use only.

7.3 Device configuration (DeviceConfig)

This object holds device information that are configurable, i.e. device name.

```
{
    "device_name" : String,
    "locate_mode" : Boolean
}
```

Member name	Since	Type	Included in	Description
device_name	2.0	String	get identity, get settings	Name of the device
locate_mode	2.13	Boolean	get settings	Whether the device is in locate mode. When a device is in locate mode, all its LEDs flash so it can be located more easily.

7.4 Device Status (DeviceStatus)

The device status object holds information about device states such as if it is active on the network and connection state.

```
{
    "active" : Boolean,
    "boot_status" : String,
    "error_status" : DeviceErrorStatus,
    "point_to_point" : Boolean,
    "temperature" : Integer
}
```

Member name	Since	Type	Description
active	2.0	Boolean	Whether device is currently connected to the network.
boot_status	2.13	String	Firmware image on which the device booted. One of the following: • PRIMARY for the primary image • GOLDEN for the golden (i.e. fallback) image • UNKNOWN if it cannot be determined because the firmware does not support this feature If the value is GOLDEN, it indicates that the primary firmware image is corrupted and should be re-programmed.
error_status	2.13	DeviceErrorStatus	Indicates whether the device is reporting an error code.
point_to_point	2.0	Boolean	Whether device is operating in point-to-point mode.
update_in_progress	2.13	Boolean	Whether a firmware update (**update** command) or first-time programming (**program** command) has been started on this device. Once the firmware programming procedure is complete, the device must be rebooted, at which point the value of this member will be reset to false.
temperature	2.5	Integer	Current chipset junction temperature in degrees Celsius **Note:** This member is only included in the response to a **get temperature** or **get device** command.

A DeviceErrorStatus object has the following structure:

```
{
   "has_error_code" : Boolean,
   "error_code " : Integer
}
```

Member name	Since	Type	Description
has_error_code	2.13	Boolean	Whether the device is reporting an error code.
error_code	2.13	Integer	If the **has_error_code** member is true, this member contains the error code reported by the device. Otherwise, the value of this member is undefined. The meaning of an error code is device-dependent.

7.5 Streams (StreamObject)

Each stream has a stream object that specifies the stream type, its configuration and the stream origin.

```
{
   "type" : String,
   "index" : Integer,
   "configuration" : StreamConfig,
```

```
    "status" : StreamStatus,
    "source" : SourceReference
}
```

Member name	Since	Type	Description
type	2.0	String	Type of data contained in the stream, one of: `AUDIO` -- audio from an analog source `CEC` -- CEC data `HDMI` -- HDMI video `HDMI_AUDIO HDMI` -- audio `I2S_AUDIO` -- I2S audio `INFRARED` -- infrared remote-control data `RS232` -- RS-232 data
index	2.0	Integer	Index of the stream of the given type (streams are indexed individually by type, not necessarily contiguous)
configuration	2.0	StreamConfig	Stream configuration
source	2.0	SourceReference	Identifies the origin of the streamed data

7.5.1 Stream Configuration (StreamConfig)

Stream Configuration object is a sub object inside the stream object.

```
{
    "address" : String,
    "enable" :  Boolean,
    "auto_stop" :  Boolean,
    "source" :  SourceSelection
}
```

Member name	Since	Type	Description
address	2.0	String	IP address to which the data is to be sent.
enable	2.0	Boolean	Whether the stream is enabled or not
auto_stop	2.13	Boolean	When true, this stream stops automatically whenever the video source is lost or changes resolution. It must then be started again with a **start** command. This feature is provided to help with network bandwidth management. A video resolution change leads to a change in the bandwidth being used by the video and it might be appropriate to take some actions in response (e.g. enabling/disabling compression) before allowing the stream to start again. Always false for streams other than the second HDMI stream (HDMI:1).
source	2.8	SourceSelection	(Optional) Selection of the input source. This member is only present if the input source is configurable on this specific stream.

7.5.2 Stream Status (StreamStatus)

Sub-object within the stream object holding the stream status.

```
{
   "state" : String,
}
```

Member name	Since	Type	Description
state	2.0	String	State of the subscription: `STREAMING` Subscription is running `STOPPED` Subscription is stopped `LINK_DOWN` The 10-Gigabit network interface is down

7.6 Subscriptions (SubscriptionObject)

The subscription object holds information about stream subscriptions (which devices have subscribed to receive which streams).

```
{
   "type" : String,
   "index" : Integer,
   "configuration" : SubscriptionConfig
   "status" : SubscriptionStatus
}
```

Member name	Since	Type	Description
type	2.0	String	Type of data, same values as stream type
index	2.0	Integer	Index of the subscription of the given type (subscriptions are indexed individually by type, not necessarily contiguous).
configuration	2.0	SubscriptionConfig	Subscription configuration
source	2.0	SubscriptionStatus	Subscription status

7.6.1 Subscription Configuration (SubscriptionConfig)

Sub-Object inside the subscription object

```
{
   "address" : String,
   "enable" :  Boolean
}
```

Member name	Since	Type	Description
address	2.0	String	IP address from which to receive the data.
enable	2.0	Boolean	Whether the subscription is enabled or not

7.6.2 Subscription Status (SubscriptionStatus)

Sub-Object inside the subscription object

```
{
   "state" : String,
```

```
}
```

Member name	Since	Type	Description
state	2.0	String	State of the subscription: `STREAMING` -- Subscription is running `STOPPED` -- Subscription is stopped `LINK_DOWN` -- 10-Gigabit network interface is down Other values than the above will also be present. These are transient states which occur right after a configuration change and should be ignored.

7.7 Nodes (Node)

Describes the hardware modules present on the device along with their status and configuration.

```
{
   "type" : String,
   "index" : Integer,
   "configuration" : Object,
   "status" : Object,
   "inputs" : [NodeInput, ...]
}
```

Member name	Since	Type	Description
type	2.0	String	Type of node, which is one of the following: • ANALOG_AUDIO_INPUT • ANALOG_AUDIO_INPUT_OUTPUT • ANALOG_AUDIO_OUTPUT • BUTTON • FRAME_BUFFER • FRAME_RATE_CONVERTER • GPIO • HDMI_DECODER • HDMI_ENCODER • HDMI_MONITOR • I2S_AUDIO_INPUT • I2S_AUDIO_OUTPUT • INFRARED_DECODER • INFRARED_ENCODER • LED • NETWORK_INTERFACE • NETWORK_SWITCH • NULL_SOURCE • SCALER • UART • USB_ICRON_CHIP_LOCAL • USB_ICRON_CHIP_REMOTE • VIDEO_INPUT Other types might be present and should be ignored.
index	2.0	Integer	Index of the node of the given type (nodes are indexed independently for each type, not necessarily contiguous)
configuration	2.0	Object	Node configuration. The structure of this object varies

Member name	Since	Type	Description
			according to the node type. See the node type-specific sections below.
status	2.0	Object	Node status. The structure of this object varies according to the node type. See the node type-specific sections below.
inputs	2.0	Array of NodeInput	Array describing how the node is connected to upstream nodes and/or subscriptions

7.7.1 Node Inputs (NodeInput)

Sub-object inside a node object that describes how it is connected to other nodes.

```
{
   "name" : String
   "index" : Integer,
   "configuration" : NodeInputConfig,
   "status" : NodeInputStatus,
}
```

Member name	Since	Type	Description
name	2.0	String	Input name, allowable values depend on node type
index	2.0	Integer	Index of the input with the given name (inputs are indexed independently for each name)
configuration	2.0	NodeInputConfig	Input configuration
source	2.0	NodeInputStatus	Input status

7.7.1.1 Node Input Configuration (NodeInputConfig)

Description of node inputs.

```
{
   "source" : SourceSelection
}
```

Member name	Since	Type	Description
source	2.0	SourceSelection	(Optional) Selection of the input source. This member is only present if the input source is configurable on this specific input.

7.7.1.2 Node Input Status (NodeInputStatus)

```
{
   "source" : SourceReference
}
```

Member name	Since	Type	Description
source	2.0	SourceReference	Current input source

7.7.2 Analog Audio Input Node (Type ANALOG_AUDIO_INPUT)

This section describes a Node object with **type** member set to `ANALOG_AUDIO_INPUT`.

Refer to section 7.7 Nodes (Node) for details.

7.7.2.1 Configuration

None

7.7.2.2 Status

None

7.7.2.3 Inputs

None

7.7.2.4 Example

The example below shows an analog audio input node JSON object as returned as part of the output of the **get device** command.

Refer to section 5.5 Command get for details.

```
{
        "type" : "ANALOG_AUDIO_INPUT",
        "index" : 0,
        "configuration" : {},
        "status" : {},
        "inputs" : []
}
```

7.7.3 Analog Audio Input/Output Node (Type ANALOG_AUDIO_INPUT_OUTPUT)

This section describes a Node object with **type** member set to ANALOG_AUDIO_INPUT_OUTPUT.

Refer to section 7.7 Nodes (Node) for details. This node represents an analog audio port that can serve both as an input and an output port.

7.7.3.1 Configuration (AnalogAudioInputOutputNodeConfig)

```
{
   "direction" : String,
}
```

Member name	Since	Type	Description
direction	2.1	String	Port direction, which is one of the following: `INPUT` The port is set to be an audio input `OUTPUT` The port is set to be an audio output

7.7.3.2 Status

None

7.7.3.3 Inputs

Input name	Since	Configurable	Description
main	2.1	Yes (Since 2.10)	Audio input source. A device may support none, some or all of the following choices:

Input name	Since	Configurable	Description
			2 -- local HDMI audio (stereo downmix) 8 -- I2S audio subscription (audio return channel) If a device does not support any of the choices above, then the following choice is listed: 0 -- No audio source available Trying to set this last value as the selected input using a set property command yields an ILLEGAL_ARGUMENT error.

7.7.3.4 Example

The example below shows an analog audio input/output node JSON object as returned as part of the output of the `get device` command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "ANALOG_AUDIO_INPUT_OUTPUT",
  "index" : 0,
  "configuration" : {
     "direction" : "INPUT"
  },
  "status" : {},
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {},
        "status" : {
           "source" : {
              "ref_class" : "NODE",
              "ref_type" : "HDMI_DECODER",
              "ref_index" : 0
           }
        }
     }
  ]
}
```

7.7.4 Analog Audio Output Node (Type ANALOG_AUDIO_OUTPUT)

This section describes a Node object with **type** member set to `ANALOG_AUDIO_OUTPUT`.

Refer to section 7.7 Nodes (Node) for details.

7.7.4.1 Configuration

None

7.7.4.2 Status

None

7.7.4.3 Inputs

Input name	Since	Configurable	Description
Main	2.0	Yes	Audio input source. A device may support none, some or all of the following choices: 2 -- HDMI audio (stereo downmix) 3 -- Analog audio 8 -- I2S audio subscription 9 -- I2S audio local loop out

7.7.4.4 Example

The example below shows an analog audio output node JSON object as returned as part of the output of the `get device` command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "ANALOG_AUDIO_OUTPUT",
  "index" : 0,
  "configuration" : {},
  "status" : {},
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {
           "source" : {
              "value" : 3,
              "choices" : [
                 {
                    "value" : 2,
                    "description" : "HDMI Audio (Downmix)",
                    "details" : {
                       "ref_class" : "SUBSCRIPTION",
                       "ref_type" : "HDMI_AUDIO",
                       "ref_index" : 0
                    }
                 },
                 {
                    "value" : 3,
                    "description" : "Analog Audio",
                    "details" : {
                       "ref_class" : "SUBSCRIPTION",
                       "ref_type" : "AUDIO",
                       "ref_index" : 0
                    }
                 },
                 {
                    "value" : 8,
                    "description" : "I2S Audio",
                    "details" : {
                       "ref_class" : "SUBSCRIPTION",
                       "ref_type" : "I2S_AUDIO",
                       "ref_index" : 0
                    }
                 },
```

```
                    {
                        "value" : 9,
                        "description" : "I2S Audio Local Loop Out",
                        "details" : {
                            "ref_class" : "NODE",
                            "ref_type" : "I2S_AUDIO_INPUT",
                            "ref_index" : 0
                        }
                    }
                ]
            }
        },
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "AUDIO",
                "ref_index" : 0
            }
        }
    }
  ]
}
```

7.7.5 Button Node (Type BUTTON)

This section describes a Node object with **type** member set to BUTTON.

Refer to section 7.7 Nodes (Node) for details.

7.7.5.1 Configuration (ButtonNodeConfig)

```
{
   "action" : ButtonActionConfig,
}
```

Member name	Since	Type	Description
action	2.0	ButtonActionConfig	Action triggered when this button is pressed

7.7.5.1.1 Button Action Configuration (ButtonActionConfig)

```
{
   "value" : Integer,
   "choices" : [ButtonActionChoice, ...]
}
```

Member name	Since	Type	Description
value	2.0	Integer	Button action selection, must match the **value** member of one of the choices listed in the **choices** member. **Note:** Not all actions are available for all buttons and all devices. The valid choices are the ones listed in the choices member.
choices	2.0	Array of ButtonActionChoice	Button action choices **Note:** This member is not included in the response of a `get settings` command. A `get device` command is required.

7.7.5.1.2 ButtonActionChoice

```
{
   "value" : Integer,
   "description" : String,
   "details" : String
}
```

Member name	Since	Type	Description
value	2.0	Integer	Value of this choice
description	2.0	String	Human-readable description of this choice.
details	2.0	String	Machine-readable name of this action, which is one of the following: • `NONE` no action • `SEND_EDID` send monitor EDID to all transmitter devices. • `SWITCH_VIDEO_INPUT` switch between video inputs.

7.7.5.2 Status

None

7.7.5.3 Inputs

None

7.7.5.4 Example

The example below shows a button node JSON object as returned as part of the output of the `get device` command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "BUTTON",
  "index" : 0,
  "configuration" : {
     "action" : {
        "value" : 0,
        "choices" : [
           {
              "value" : 0,
              "description" : "No action",
              "details" : "NONE"
           },
           {
              "value" : 3,
              "description" : "Switch video input",
              "details" : "SWITCH_VIDEO_INPUT"
           }
        ]
     }
  },
  "status" : {},
  "inputs" : []
}
```

7.7.6 CEC Node (Type CEC)

This section describes a Node object with type member set to CEC. This node is as an input source to CEC streams.

Refer to section 7.7 Nodes (Node) for details.

7.7.6.1 Configuration

None

7.7.6.2 Status

None

7.7.6.3 Inputs

None

7.7.6.4 Example

The example below shows a null source node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "CEC",
  "index" : 0,
  "configuration" : {},
  "status" : {},
  "inputs" : []
}
```

7.7.7 Color Generator Node (Type COLOR_GENERATOR)

This section describes a Node object with type member set to `COLOR_GENERATOR`.

Refer to section 7.7 Nodes (Node) for details.

7.7.7.1 Configuration

```
{
   "enable" : Boolean,
   "color" : String
}
```

Member name	Since	Type	Description
enable	2.13	Boolean	When true, blank the video output with a uniform colour. This is useful to implement a video mute function. This feature is only supported when the device is in a display mode other than genlock.
color	2.13	String	String representation of the colour to use for blanking. This string has format RRGGGBB where each of RR, GG and BB is a pair of hexadecimal digits that represent respectively the 8-bit value of the red, green and blue components. These 8-bit values must be interpreted in the context of the default quantization range (full or limited) for the current output

Member name	Since	Type	Description
			video format as specified in standard CEA 861-F. This feature is only supported when the device is in a display mode other than genlock.

7.7.7.2 Status

None

7.7.7.3 Inputs

None

7.7.7.4 Example

The example below shows a color generator node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "COLOR_GENERATOR",
  "index" : 0,
  "configuration" : {
     "enable" : false,
     "color" : "000000"
  },
  "status" : {},
  "inputs" : []
}
```

7.7.8 Frame Buffer Node (Type FRAME_BUFFER)

This section describes a Node object with **type** member set to `FRAME_BUFFER`.

Refer to section 7.7 Nodes (Node) for details.

7.7.8.1 Configuration (FrameBufferNodeConfig)

```
{
   "display_mode" : String,
   "width" : Integer,
   "height" : Integer,
   "horiz_offset" : Integer,
   "vert_offset" : Integer,
   "frames_per_second" : Integer
}
```

Member name	Since	Type	Description
display_mode	2.0	String	Frame buffer display mode, which is one of the following: `FAST_SWITCHED` Full screen fast switch mode `GENLOCK_SCALING` Full screen zero-frame latency scaling mode `WALL_FAST_SWITCHED` Fast switch (legacy) display wall mode `WALL_GENLOCKED` Zero-frame latency display wall mode `MULTIVIEW` Multiview mode
width	2.0	Integer	Video width

Member name	Since	Type	Description
height	2.0	Integer	Video height
horiz_offset	2.0	Integer	Video horizontal offset
vert_offset	2.0	Integer	Video vertical offset
frames_per_second	2.0	Integer	Frame rate

7.7.8.2 Status (FrameBufferNodeStatus)

```
{
   "source_video" : VideoFormat
}
```

Member name	Since	Type	Description
source_video	2.0	VideoFormat	Format of video source (full screen and wall modes only)

7.7.8.3 Inputs

Input name	Since	Configurable	Description
main	2.0	No	Video input for full screen and wall modes. Source is a subscription.
multiview	2.0	No	Video inputs for the multiview. There are multiple inputs with this name (with different indexes), each is connected to a subscription.

7.7.8.4 Example

The example below shows a frame buffer node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "FRAME_BUFFER",
  "index" : 0,
  "configuration" : {
     "display_mode" : "FAST_SWITCHED",
     "width" : 1920,
     "height" : 1080,
     "horiz_offset" : 0,
     "vert_offset" : 0,
     "frames_per_second" : 60
  },
  "status" : {
     "source_video" : {
        "width" : 1920,
        "height" : 1080,
        "frames_per_second" : 60,
        "color_space" : "RGB",
        "bits_per_pixel" : 8,
        "scan_mode" : "PROGRESSIVE"
     }
  },
  "inputs" : [
```

```
{
   "name" : "main",
   "index" : 0,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 0
      }
   }
},
{
   "name" : "multiview",
   "index" : 0,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 0
      }
   }
},
{
   "name" : "multiview",
   "index" : 1,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 1
      }
   }
},
{
   "name" : "multiview",
   "index" : 2,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 2
      }
   }
},
{
   "name" : "multiview",
   "index" : 3,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
```

```
                "ref_index" : 3
            }
        }
    }
  ]
}
```

7.7.9 Frame Rate Converter Node (Type FRAME_RATE_CONVERTER)

This section describes a Node object with **type** member set to `FRAME_RATE_CONVERTER`.

Refer to section 7.7 Nodes (Node) for details.

7.7.9.1 Configuration

None

7.7.9.2 Status (FrameRateConverterNodeStatus)

```
{
   "divide_factor" : Integer
}
```

Member name	Since	Type	Description
divide_factor	2.8	Integer	Factor by which the frame rate is divided

7.7.9.3 Inputs

Input name	Since	Configurable	Description
Main	2.8	No	Main video input.

7.7.9.4 Example

The example below shows a frame rate converter node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "FRAME_RATE_CONVERTER",
  "index" : 0,
  "configuration" : {},
  "status" : {
     "divide_factor" : 2
  },
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {},
        "status" : {
           "source" : {
              "ref_class" : "NODE",
              "ref_type" : "HDMI_DECODER",
              "ref_index" : 0
           }
        }
     }
```

```
    ]
}
```

7.7.10 General-Purpose Input-Output (GPIO) Node (Type GPIO)

This section describes a Node object with **type** member set to `GPIO`.

Refer to section 7.7 Nodes (Node) for details.

Each GPIO node represents a GPIO pin.

Note: This node type is only included in the response to a `get gpio` or a `get device` command.

7.7.10.1 Configuration (GpioNodeConfig)

```
{
   "tristate" : Integer,
   "cfg_val" : Integer
}
```

Member name	Since	Type	Description
tristate	2.10	Integer	Current direction of the GPIO: `0` -- indicates the pin is an input. `1` -- indicates the pin is an output.
cfg_val	2.10	Integer	The output value (0 or 1) currently configured for the GPIO pin.

7.7.10.2 Status (GpioNodeStatus)

```
{
   "stat_val" : Integer
}
```

Member name	Since	Type	Description
stat_val	2.10	Integer	The value (`0` or `1`) read from the GPIO pin.

7.7.10.3 Inputs

None

7.7.10.4 Example

The example shown below shows a GPIO node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
   "type" : "GPIO",
   "index" : 0,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   },
   "inputs" : []
}
```

7.7.11 HDMI Decoder Node (Type HDMI_DECODER)

This section describes a Node object with **type** member set to `HDMI_DECODER`.

Refer to section 7.7 Nodes (Node) for details.

Note: This node type is included in the response to a `get edid` command (in addition to `get settings`). However, in the response to a `get edid` command, the returned node only contains the configuration member, which itself only contains the `edid` member.

7.7.11.1 Configuration (HdmiDecoderNodeConfig)

```
{
   "edid" : String,
   "hdcp_support_enable" : Boolean,
   "hdcp_22_support_disable" : Boolean,
}
```

Member name	Since	Type	Description
edid	2.0	String	Configured EDID seen by connected HDMI sources. The format of this member is a hexadecimal string with a length of 512 characters (represents 256 bytes of data). **Note:** This member is not included in the response of a `get settings` command. It is, however, included in response to a `get edid` command.
hdcp_support_enable	2.5	Boolean	Enable HDCP support. When disabled, the device reports to the HDMI source that is does not support HDCP. **Notes:** • Some laptops and possibly other HDMI sources always encrypt video with HDCP if the sink connected to it supports it, even when not playing back HDCP-protected content. • This can cause problems when trying to display video from theses HDMI sources on monitors without HDCP support. Disabling HDCP support in the transmitter device solves this issue.
hdcp_22_support_disable	2.6	Boolean	Disable HDCP 2.2 support. When set to `true`, device reports to the HDMI source that only supports HDCP 1.4 (if `hdcp_support_enable` is also true).

7.7.11.2 Status (HdmiDecoderNodeStatus)

```
{
   "hdcp_protected" : Boolean,
   "hdcp_version" : String,
   "hdmi_2_0_support" : Boolean,
   "source_stable" : Boolean,
   "video" : VideoFormat,
   "has_video_details" : Boolean,
   "video_details" : VideoFormatDetails,
   "has_audio_details" : Boolean,
```

```
    "audio_details" : AudioFormatDetails,
    "audio_downmix_support" : Boolean
}
```

Member name	Since	Type	Description
hdcp_proctected	2.0	Boolean	Whether the HDMI source is HDCP protected.
hdcp_version	2.5	String	Version of HDCP content protection: `NONE` -- if content is not HDCP-protected. `HDCP_1_4` -- for HDCP 2.1 and below (including 1.4). `HDCP_2_2` -- for HDCP 2.2.
hdmi_2_0_support	2.2	Boolean	Whether this HDMI decoder supports HDMI 2.0 resolutions (e.g. 4K60).
source_stable	2.0	Boolean	Whether device is receiving video from a connected source.
Video	2.0	VideoFormat	Format of video received from the source (only valid if source stable is true). Undefined if `source_stable` is false.
has_video_details	2.8	Boolean	Indicates whether detailed video format and timing information available from the device.
video_details	2.8	VideoFormatDetails	Detailed video format and timing information. Undefined if `source_stable` or `has_video_details` is false.
has_audio_details	2.10	Boolean	Indicates whether audio format information is available from the device.
audio_details	2.10	AudioFormatDetails	Information regarding the audio format. Undefined if `source_stable` or `has_audio_details` is false.
audio_downmix_support	2.10	Boolean	Indicates whether the HDMI decoder generates a downmix version of extracted HDMI audio.

7.7.11.3 Inputs

Input name	Since	Configurable	Description
Main	2.0	Yes*	Main video input. The video input connector can be selected on devices that have multiple inputs available. Modes can also be selected where the device automatically selects an input with a video source present.

***** This input is only configurable on devices with multiple video inputs.

7.7.11.4 Example

The example below shows an HDMI decoder node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.


```
{
  "type" : "HDMI_DECODER",
  "index" : 0,
  "configuration" : {
     "edid" :
"00ffffffffffff00614000156524c420816010380341d780a01c1a057479827124
c4c210800814001010101010101010101010101010101023a801871382d40582c4600408
46300001e08e80030f2705a80b0588a00ba892100001e000000fc004170746f56697
3696f6e20200a000000fd0018550e853c000a20202020202001200203407358010 20
30405111213141f2021220607905d5e5f6263646061230d0707830f00006d030c001
000383c20406801020367d85dc401788003e40f0000c0011d00bc52d01e20b828554
040846300001e023a80d072382d40102c458040846300001e023a801871382d40582
c450040846300001e000000000000000000f2",
     "hdcp_support_enable" : true,
     "hdcp_22_support_disable" : false
  },
  "status" : {
     "video" : {
        "width" : 0,
        "height" : 0,
        "frames_per_second" : 0,
        "color_space" : "RGB",
        "bits_per_pixel" : 10,
        "scan_mode" : "PROGRESSIVE"
     },
     "has_video_details" : true,
     "video_details" : {
        "pixel_clock" : 0,
        "total_width" : 0,
        "total_height" : 0,
        "hsync_width" : 0,
        "hsync_front_porch" : 0,
        "hsync_negative" : false,
        "vsync_width" : 0,
        "vsync_front_porch" : 0,
        "vsync_negative" : false,
        "vic" : 0,
        "has_hdmi_vic" : false,
        "hdmi_vic" : 0,
        "picture_aspect" : "ASPECT_NO_DATA",
        "has_active_format" : false,
        "active_format" : 0,
        "it_content_type" : "NO_DATA",
        "scan_information" : "NO_DATA",
        "colorimetry" : "RGB",
        "rgb_range" : "DEFAULT",
        "ycc_range" : "LIMITED"
     },
```

```
      "has_audio_details" : true,
      "audio_details" : {
         "sampling_frequency" : 48000,
         "number_of_channels" : 8
      },
      "hdcp_protected" : false,
      "hdcp_version" : "NONE",
      "hdmi_2_0_support" : true,
      "source_stable" : false,
      "audio_downmix_support" : true
   },
   "inputs" : [
      {
         "name" : "main",
         "index" : 0,
         "configuration" : {
            "source" : {
               "value" : 0,
               "choices" : [
                  {
                     "value" : 0,
                     "description" : "HDMI Input 1",
                     "details" : {
                        "ref_class" : "NODE",
                        "ref_type" : "VIDEO_INPUT",
                        "ref_index" : 0
                     }
                  },
                  {
                     "value" : 1,
                     "description" : "Display Port Input 1",
                     "details" : {
                        "ref_class" : "NODE",
                        "ref_type" : "VIDEO_INPUT",
                        "ref_index" : 1
                     }
                  },
                  {
                     "value" : 32,
                     "description" : "Automatic Input Selection (HDMI
Input 1)",
                     "details" : {
                        "ref_class" : "AUTOMATIC",
                        "ref_type" : null,
                        "ref_index" : 0
                     }
                  }
               ]
            }
         },
         "status" : {
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "VIDEO_INPUT",
               "ref_index" : 0
            }
```

```
                }
            }
        ]
}
```

7.7.12 HDMI Encoder Node (Type HDMI_ENCODER)

This section describes a Node object with **type** member set to `HDMI_ENCODER`.

Refer to section 7.7 Nodes (Node) for details.

7.7.12.1 Configuration

None

7.7.12.2 Status (HdmiEncoderNodeStatus)

```
{
    "hdcp_protected" : Boolean,
    "hdmi_2_0_support" : Boolean,
    "source_stable" : Boolean,
    "video" : VideoFormat,
    "has_video_details" : Boolean,
    "video_details" : VideoFormatDetails,
}
```

Member name	Since	Type	Description
hdcp_protected	2.0	Boolean	Whether HDCP protection is enabled (automatically enabled if source is HDCP protected).
hdmi_2_0_support	2.2	Boolean	Whether the HDMI encoder supports HDMI 2.0 resolutions (e.g. 4K60).
source_stable	2.0	Boolean	Whether device is receiving video from the source
video	2.0	VideoFormat	Format of video being output by the HDMI encoder. Undefined if `source_stable` is false.
has_video_details	2.8	Boolean	Indicates whether detailed video format and timing information is available from this device.
video_details	2.8	VideoFormatDetails	Detailed video format and timing information. Undefined if `source_stable` or `has_video_details` is false.

7.7.12.3 Inputs

Input name	Since	Configurable	Description
main	2.0	No	Main video input. A device may support one or both of the following values: 0 -- Video from a video subscription (genlock mode) 1 -- Video from a frame buffer (frame buffer mode)

Input name	Since	Configurable	Description
audio	2.0	Yes	HDMI audio source. A device may support none, some or all of the following choices: 2 -- HDMI audio (original from video subscription in genlock mode) 3 -- Analog audio 6 -- HDMI audio (stereo downmix) 7 -- HDMI audio (all available channels) 8 -- I2S audio subscription 9 -- I2S audio local loop out

7.7.12.4 Example

The example below shows an HDMI encoder node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.


```
{
  "type" : "HDMI_ENCODER",
  "index" : 0,
  "configuration" : {},
  "status" : {
     "video" : {
        "width" : 1920,
        "height" : 1080,
        "frames_per_second" : 60,
        "color_space" : "RGB",
        "bits_per_pixel" : 8,
        "scan_mode" : "PROGRESSIVE"
     },
     "has_video_details" : true,
     "video_details" : {
        "pixel_clock" : 148500000,
        "total_width" : 2200,
        "total_height" : 1125,
        "hsync_width" : 44,
        "hsync_front_porch" : 88,
        "hsync_negative" : false,
        "vsync_width" : 5,
        "vsync_front_porch" : 4,
        "vsync_negative" : false,
        "vic" : 16,
        "has_hdmi_vic" : false,
        "hdmi_vic" : 0,
        "picture_aspect" : "ASPECT_16_9",
        "has_active_format" : false,
        "active_format" : 0,
        "it_content_type" : "GRAPHICS",
        "scan_information" : "NO_DATA",
        "colorimetry" : "BT709",
        "rgb_range" : "DEFAULT",
        "ycc_range" : "LIMITED"
     },
     "hdcp_protected" : false,
     "hdcp_version" : "NONE",
```

```
      "hdmi_2_0_support" : true,
      "source_stable" : false
   },
   "inputs" : [
      {
         "name" : "main",
         "index" : 0,
         "configuration" : {
            "source" : {
               "value" : 0,
               "choices" : [
                  {
                     "value" : 0,
                     "description" : "Genlock Mode",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "HDMI",
                        "ref_index" : 0
                     }
                  },
                  {
                     "value" : 1,
                     "description" : "Frame Buffer Mode",
                     "details" : {
                        "ref_class" : "NODE",
                        "ref_type" : "FRAME_BUFFER",
                        "ref_index" : 0
                     }
                  }
               ]
            }
         },
         "status" : {
            "source" : {
               "ref_class" : "SUBSCRIPTION",
               "ref_type" : "HDMI",
               "ref_index" : 0
            }
         }
      },
      {
         "name" : "audio",
         "index" : 0,
         "configuration" : {
            "source" : {
               "value" : 2,
               "choices" : [
                  {
                     "value" : 2,
                     "description" : "HDMI Audio (from HDMI Video in
Genlock Mode)",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "HDMI_AUDIO",
                        "ref_index" : 0
```

```
                        }
                    },
                    {
                        "value" : 3,
                        "description" : "Analog Audio",
                        "details" : {
                            "ref_class" : "SUBSCRIPTION",
                            "ref_type" : "AUDIO",
                            "ref_index" : 0
                        }
                    },
                    {
                        "value" : 6,
                        "description" : "HDMI Audio (Stereo Downmix)",
                        "details" : {
                            "ref_class" : "SUBSCRIPTION",
                            "ref_type" : "HDMI_AUDIO",
                            "ref_index" : 0
                        }
                    },
                    {
                        "value" : 7,
                        "description" : "HDMI Audio (All Available
Channels)",
                        "details" : {
                            "ref_class" : "SUBSCRIPTION",
                            "ref_type" : "HDMI_AUDIO",
                            "ref_index" : 0
                        }
                    },
                    {
                        "value" : 8,
                        "description" : "I2S Audio",
                        "details" : {
                            "ref_class" : "SUBSCRIPTION",
                            "ref_type" : "I2S_AUDIO",
                            "ref_index" : 0
                        }
                    },
                    {
                        "value" : 9,
                        "description" : "I2S Audio Local Loop Out",
                        "details" : {
                            "ref_class" : "NODE",
                            "ref_type" : "I2S_AUDIO_INPUT",
                            "ref_index" : 0
                        }
                    }
                ]
            }
        },
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI_AUDIO",
                "ref_index" : 0
```

```
                }
            }
        }
    ]
}
```

7.7.13 HDMI Monitor Node (Type HDMI_MONITOR)

This section describes a Node object with **type** member set to `HDMI_MONITOR`.

Refer to section 7.7 Nodes (Node) for details.

Note: This node type is included in the response to a "`get edid`" command (in addition to `get settings`). However, in the response to a `get edid` command, the returned node only contains the status member.

7.7.13.1 Configuration

None

7.7.13.2 Status (HdmiMonitorNodeStatus)

```
{
   "connected" : Boolean,
   "edid" : String
}
```

Member name	Since	Type	Description
connected	2.0	Boolean	Whether there is a connected monitor (HPD – Hot-Plug Detect) **Note:** This member is included in the response of both the `get edid` and `get settings` command.
edid	2.0	String	EDID of the connected monitor. The format of this member is a hexadecimal string with a length of 512 characters (represents 256 bytes of data). If no monitor is connected (see **connected** member), this member contains an empty string. **Note:** This member is not included in the response of a `get settings` command. It is, however, included in the response to a `get edid` command.

7.7.13.3 Inputs

Input name	Since	Configurable	Description
main	2.0	No	HDMI encoder which sends video to this monitor

7.7.13.4 Example

The example below shows an HDMI monitor node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "HDMI_MONITOR",
  "index" : 0,
  "configuration" : {},
  "status" : {
     "connected" : true,
     "edid" :
"00ffffffffffff004c2d5706333244590a140103801009782aee91a3544c99260f5
054230800a940818081408100950b30001010101023a801871382d40582c4500a05
a0000001e011d007251d01e206e285500a05a0000001e000000fd00323c1b5111000
a2020202020200000000fc0053796e634d61737465720a2020013b02031cf34890041
f05141303122309070783010000660300c00100080011d80d0721c1620102c2580a05
a0000009e011d8018711c1620582c2500a05a0000009e011d00bc52d01e20b828554
0a05a0000001e011d007251d01e206e285500a05a0000001e8c0ad090204031200c4
05500a05a000001800000000000000000075"
  },
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {},
        "status" : {
           "source" : {
              "ref_class" : "NODE",
              "ref_type" : "HDMI_ENCODER",
              "ref_index" : 0
           }
        }
     }
  ]
}
```

7.7.14 I2S Audio Input Node (Type I2S_AUDIO_INPUT)

This section describes a Node object with **type** member set to `I2S_AUDIO_INPUT`.

Refer to section 7.7 Nodes (Node) for details.

7.7.14.1 Configuration

```
{
   "sampling_frequency" : SamplingFrequencyConfig,
   "number_of_channels" : ChannelsNumberConfig
}
```

Member name	Since	Type	Description
sampling_frequency	2.10	SamplingFrequencyConfig	Audio sampling rate configuration
number_of_channels	2.10	ChannelsNumberConfig	Number of audio channels configuration

7.7.14.1.1 Sampling Frequency Configuration (SamplingFrequencyConfig)

```
{
   "value" : Integer,
   "choices" : [SamplingFrequencyChoice, ...]
}
```

Member name	Since	Type	Description
value	2.10	Integer	Sampling frequency selection. This value is the sampling frequency in Hertz and must match the **value** member of one of the choices listed in the **choices** member. **Note:** Not all sampling rates are supported by all devices. The valid choices are the ones listed in the choices member.
choices	2.10	Array of SamplingFrequencyChoice	Sampling rate choices. **Note:** This member is not included in the response of a `get settings` command. A `get device` command is required.

7.7.14.1.2 SamplingFrequencyChoice

```
{
   "value" : Integer,
   "description" : String,
   "details" : Undefined
}
```

Member name	Since	Type	Description
value	2.10	Integer	Sampling frequency in Hertz.
description	2.10	String	Human-readable description of this choice.
details	2.10	Undefined	Undefined. Reserved for future use.

7.7.14.1.3 Number of Audio Channels Configuration (ChannelsNumberConfig)

```
{
   "value" : Integer,
   "choices" : [ChannelsNumberChoice, ...]
}
```

Member name	Since	Type	Description
value	2.10	Integer	Number of channels selection. Values from 2 to 8 represent the number of audio channels. Other values are reserved for future use. This value must match the **value** member of one of the choices listed in the **choices** member. **Note:** Not all number of channels are supported by all devices. The valid choices are the ones listed in the choices member.
choices	2.10	Array of ChannelsNumberChoice	Number of audio channels choices. **Note:** This member is not included in the response of a `get settings` command. A `get device` command is required.

7.7.14.1.4 ChannelsNumberChoice

```
{
   "value" : Integer,
   "description" : String,
   "details" : Undefined
}
```

Member name	Since	Type	Description
value	2.10	Integer	Value of this choice. Values from 2 to 8 represent the number of audio channels. Other values are reserved for future use.
description	2.10	String	Human-readable description of this choice.
details	2.10	Undefined	Undefined. Reserved for future use.

7.7.14.2 Status

None

7.7.14.3 Inputs

None

7.7.14.4 Example

The example below shows an I2S audio input node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "I2S_AUDIO_INPUT",
  "index" : 0,
  "configuration" : {
     "sampling_frequency" : {
        "value" : 48000,
        "choices" : [
           {
              "value" : 22050,
              "description" : "22.05kHz",
              "details" : null
           },
           {
              "value" : 24000,
              "description" : "24kHz",
              "details" : null
           },
           {
              "value" : 32000,
              "description" : "32kHz",
              "details" : null
           },
           {
              "value" : 44100,
              "description" : "44.1kHz",
              "details" : null
           },
```

```
            {
               "value" : 48000,
               "description" : "48kHz",
               "details" : null
            },
            {
               "value" : 88200,
               "description" : "88.2kHz",
               "details" : null
            },
            {
               "value" : 96000,
               "description" : "96kHz",
               "details" : null
            },
            {
               "value" : 176400,
               "description" : "176.4kHz",
               "details" : null
            },
            {
               "value" : 192000,
               "description" : "192kHz",
               "details" : null
            },
            {
               "value" : 768000,
               "description" : "768kHz",
               "details" : null
            }
         ]
      },
      "number_of_channels" : {
         "value" : 8,
         "choices" : [
            {
               "value" : 2,
               "description" : "2 Channels",
               "details" : null
            },
            {
               "value" : 3,
               "description" : "3 Channels",
               "details" : null
            },
            {
               "value" : 4,
               "description" : "4 Channels",
               "details" : null
            },
            {
               "value" : 5,
               "description" : "5 Channels",
               "details" : null
            },
```

```
                {
                   "value" : 6,
                   "description" : "6 Channels",
                   "details" : null
                },
                {
                   "value" : 7,
                   "description" : "7 Channels",
                   "details" : null
                },
                {
                   "value" : 8,
                   "description" : "8 Channels",
                   "details" : null
                }
             ]
          }
       },
       "status" : {},
       "inputs" : []
    }
```

7.7.15 I2S Audio Output Node (Type I2S_AUDIO_OUTPUT)

This section describes a Node object with **type** member set to I2S_AUDIO_OUTPUT. See section 7.7 Nodes (Node) for details.

7.7.15.1 Configuration

None

7.7.15.2 Status

None

7.7.15.3 Inputs

Input name	Since	Configurable	Description
main	2.9	Yes	Audio source. A transmitter may support none, some or all of the following choices: 3 -- Analog audio local loop out 6 -- HDMI audio (stereo downmix) local loop out 7 -- HDMI audio (all channels) local loop out 8 -- I2S audio subscription (audio return channel) A receiver may support none, some or all of the following choices: 3 -- Analog audio 6 -- HDMI audio (stereo downmix) 7 -- HDMI audio (all channels) 8 -- I2S audio subscription If a device has an I2S audio output but does not support any of the choices above, then the following choice is listed: • 0 No audio source available Trying to set this last value as the selected input using a set property command yields an `ILLEGAL_ARGUMENT` error.

7.7.15.4 Example

The example below shows an I2S audio output node JSON object as returned as part of the output of the "`get device`" command called on a transmitter target device.

Refer to section 5.5 Command get for details.


```
{
  "type" : "I2S_AUDIO_OUTPUT",
  "index" : 0,
  "configuration" : {},
  "status" : {},
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {
           "source" : {
              "value" : 7,
              "choices" : [
                 {
                    "value" : 3,
                    "description" : "Analog Audio Local Loop Out",
                    "details" : {
                       "ref_class" : "NODE",
                       "ref_type" : "ANALOG_AUDIO_INPUT_OUTPUT",
                       "ref_index" : 0
                    }
                 },
                 {
                    "value" : 6,
                    "description" : "HDMI Audio Local Loop Out (Downmix)",
                    "details" : {
                       "ref_class" : "NODE",
                       "ref_type" : "HDMI_DECODER",
                       "ref_index" : 0
                    }
                 },
                 {
                    "value" : 7,
                    "description" : "HDMI Audio Local Loop Out (All
Channels)",
                    "details" : {
                       "ref_class" : "NODE",
                       "ref_type" : "HDMI_DECODER",
                       "ref_index" : 0
                    }
                 }
              ]
           }
        },
        "status" : {
           "source" : {
              "ref_class" : "NODE",
              "ref_type" : "HDMI_DECODER",
              "ref_index" : 0
```

```
            }
         }
      }
   ]
}
```

The example below shows an I2S audio output node JSON object as returned as part of the output of the "`get device`" command called on a receiver target device.

Refer to section 5.5 Command get for details.

```
{
   "type" : "I2S_AUDIO_OUTPUT",
   "index" : 0,
   "configuration" : {},
   "status" : {},
   "inputs" : [
      {
         "name" : "main",
         "index" : 0,
         "configuration" : {
            "source" : {
               "value" : 8,
               "choices" : [
                  {
                     "value" : 3,
                     "description" : "Analog Audio",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "AUDIO",
                        "ref_index" : 0
                     }
                  },
                  {
                     "value" : 6,
                     "description" : "HDMI Audio (Stereo Downmix)",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "HDMI_AUDIO",
                        "ref_index" : 0
                     }
                  },
                  {
                     "value" : 7,
                     "description" : "HDMI Audio (All Available Channels)",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "HDMI_AUDIO",
                        "ref_index" : 0
                     }
                  },
                  {
                     "value" : 8,
                     "description" : "I2S Audio",
                     "details" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "I2S_AUDIO",
                        "ref_index" : 0
```

```
                    }
                }
            ]
        }
    },
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "I2S_AUDIO",
            "ref_index" : 0
        }
    }
  }
]
}
```

7.7.16 Infrared Decoder Node (Type INFRARED_DECODER)

This section describes a Node object with **type** member set to `INFRARED_DECODER`.

Refer to section 7.7 Nodes (Node) for details.

7.7.16.1 Configuration

None

7.7.16.2 Status

None

7.7.16.3 Inputs

None

7.7.16.4 Example

The example below shows an infrared decoder node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
        "type" : "INFRARED_DECODER",
        "index" : 0,
        "configuration" : {},
        "status" : {},
        "inputs" : []
}
```

7.7.17 Infrared Encoder Node (Type INFRARED_ENCODER)

This section describes a Node object with **type** member set to `INFRARED_ENCODER`.

Refer to section 7.7 Nodes (Node) for details.

7.7.17.1 Configuration

None

7.7.17.2 Status

```
{
   "data_length_max" : Integer
}
```

Member name	Since	Type	Description
data_length_max	2.12	Integer	Maximum length, in bytes, of the Pronto code that can be sent to this infrared encoder.

7.7.17.3 Inputs

Input name	Since	Configurable	Description
network	2.0	No	Associated network subscription.

7.7.17.4 Example

The example below shows an infrared encoder node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
      "type" : "INFRARED_ENCODER",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : [{
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
                  "source" : {
                        "ref_class" : "SUBSCRIPTION",
                        "ref_type" : "INFRARED",
                        "ref_index" : 0
                  }
            }
      }]
}
```

7.7.18 LED Node (Type LED)

This section describes a Node object with **type** member set to LED.

Refer to section 7.7 Nodes (Node) for details.

7.7.18.1 Configuration (LedNodeConfig)

```
{
   "function" : LedFunctionConfig,
}
```

Member name	Since	Type	Description
function	2.0	LedFunctionConfig	LED Function

7.7.18.1.1 LED Function Configuration (LedFunctionConfig)

```
{
   "value" : Integer,
   "choices" : [LedFunctionChoice, ...]
}
```

Member name	Since	Type	Description
value	2.0	Integer	LED function selection. When setting this property using the `set property` command, the following values are allowed: `1` -- blinking LED `2` -- power LED `3` -- video is stable indicator `4` -- network TX indicator `5` -- network RX indicator `6` -- always ON `7` -- always OFF `8` -- default function for this LED Device firmware allows LED function to be set but does not report current function back to the API server. For this reason, the value reported in this member is always: • 0 current function is unknown This value cannot be set with `set property` command. **Note:** Not all functions are available for all LEDs. The valid choices are the ones listed in the choices member.
choices	2.0	Array of LedFunctionChoice	LED function choices **Note:** This member is not included in the response of a `get settings` command.

7.7.18.1.2 LedFunctionChoice

```
{
   "value" : Integer,
   "description" : String,
   "details" : String
}
```

Member name	Since	Type	Description
value	2.0	Integer	Value of this choice
description	2.0	String	Human-readable description of this choice.
details	2.0	String	Machine-readable name of this function, which is one of the following: `UNKNOWN` -- current function is unknown `BLINK` -- blinking LED `POWER` -- power LED `VIDEO_STABLE` -- video is stable `NETWORK_TX` -- network TX indicator `NETWORK_RX` -- network RX indicator `ALWAYS_ON` -- always ON

Member name	Since	Type	Description
			`ALWAYS_OFF` -- always OFF `DEFAULT` -- default function for this LED

7.7.18.2 Status

None

7.7.18.3 Inputs

None

7.7.18.4 Example

The example shown below shows a LED node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
   "type" : "LED",
   "index" : 0,
   "configuration" : {
      "function" : {
         "value" : 0,
         "choices" : [
            {
               "value" : 0,
               "description" : "(Unknown)",
               "details" : "UNKNOWN"
            },
            {
               "value" : 1,
               "description" : "Blink LED",
               "details" : "BLINK"
            },
            {
               "value" : 8,
               "description" : "Default functionality",
               "details" : "DEFAULT"
            }
         ]
      }
   },
   "status" : {},
   "inputs" : []
}
```

7.7.19 Network Interface Node (Type NETWORK_INTERFACE)

This section describes a Node object with **type** member set to NETWORK_INTERFACE. See section 7.7 Nodes (Node) for detail.

7.7.19.1 Configuration (NetworkInterfaceNodeConfig)

```
{
   "hostname" : String,
   "ip" : NetInterfaceIPConfig
}
```

Member name	Since	Type	Description
hostname	2.0	String	Host name for the device on this interface
ip	2.0	NetInterfaceIPConfig	Internet Protocol (IP) configuration

7.7.19.1.1 Network Interface IP Configuration (NetInterfaceIPConfig)

```
{
   "address" : String,
   "mode" : String,
   "mask" : String,
   "gateway" : String
}
```

Member name	Since	Type	Description
address	2.0	String	IP address.
mode	2.0	String	One of the following: `MANUAL` - Internet Protocol configuration is fixed and configured manually. `DHCP` - Internet Protocol configuration is obtained from a DHCP server, with fallback on auto-IP if no DHCP server is available.
mask	2.0	String	IP subnet mask
gateway	2.0	String	IP address of default gateway

7.7.19.2 Status (NetworkInterfaceNodeStatus)

```
{
   "mac_address" : String,
   "ip" : NetInterfaceIPStatus
}
```

Member name	Since	Type	Description
mac_address	2.0	String	MAC address for this interface
ip	2.2	NetInterfaceIPStatus	Internet Protocol (IP) status

7.7.19.2.1 Network Interface IP Status (NetInterfaceIPStatus)

```
{
   "address" : String
}
```

Member name	Since	Type	Description
address	2.2	String	IP address

7.7.19.3 Inputs

None

7.7.19.4 Example

The example below shows a network interface node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
	"type" : "NETWORK_INTERFACE",
	"index" : 0,
	"configuration" : {
		"hostname" : "PHILIPPE_TX68",
		"ip" :
{
			"address" : "10.1.1.37",
			"mode" : "DHCP",
			"mask" : "255.255.255.0",
			"gateway" : "10.1.1.1"
		}
	},
	"status" : {
		"mac_address" : "001ec0f03668",
		"ip" : {
			"address" : "10.1.1.37"
		}
	},
	"inputs" : []
}
```

7.7.20 Network Switch Node (Type NETWORK_SWITCH)

This section describes a Node object with **type** member set to `NETWORK_SWITCH`.

Refer to section 7.7 Nodes (Node) for details.

7.7.20.1 Configuration (NetworkSwitchNodeConfig)

```
{
   "gigabit_port_1_enable" : Boolean
}
```

Member name	Since	Type	Description
gigabit_port_1_enable	2.0	Boolean	Enable first Gigabit Ethernet port on the device

7.7.20.2 Status

None

7.7.20.3 Inputs

None

7.7.20.4 Example

The example below shows a network switch node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
	"type" : "NETWORK_SWITCH",
	"index" : 0,
	"configuration" : {
		"gigabit_port_1_enable" : false
	},
	"status" : {},
	"inputs" : []
```

```
}
```

7.7.21 Null Source Node (Type NULL_SOURCE)

This section describes a Node object with type member set to `NULL_SOURCE`. This node is used as an input source to other nodes to represent the fact that no valid input is available.

Refer to section 7.7 Nodes (Node) for details.

7.7.21.1 Configuration

None

7.7.21.2 Status

None

7.7.21.3 Inputs

None

7.7.21.4 Example

The example below shows a null source node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "NULL_SOURCE",
  "index" : 0,
  "configuration" : {},
  "status" : {},
  "inputs" : []
}
```

7.7.22 Scaler Node (Type SCALER)

This section describes a Node object with **type** member set to SCALER.

Refer to section 7.7 Nodes (Node) for details.

7.7.22.1 Configuration (ScalerNodeConfig)

```
{
   "width" : Integer,
   "height" : Integer
}
```

Member name	Since	Type	Description
width	2.6	Integer	Scaled video width, in pixels
height	2.6	Integer	Scaled video height, in pixels

7.7.22.2 Status (ScalerNodeStatus)

```
{
   "video" : VideoFormat
}
```

Member name	Since	Type	Description
video	2.9	VideoFormat	Output video format of the scaler

7.7.22.3 Inputs

Input name	Since	Configurable	Description
main	2.6	Yes (since 2.8)	Main video input. An HDMI decoder node or, for devices that support it, a frame rate converter to select a different frame rate.

7.7.22.4 Example

The example shown below shows a scaler node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "SCALER",
  "index" : 0,
  "configuration" : {
     "width" : 0,
     "height" : 0
  },
  "status" : {
     "video" : {
        "width" : 0,
        "height" : 0,
        "frames_per_second" : 60,
        "color_space" : "RGB",
        "bits_per_pixel" : 8,
        "scan_mode" : "PROGRESSIVE"
     }
  },
  "inputs" : [
     {
        "name" : "main",
        "index" : 0,
        "configuration" : {
           "source" : {
              "value" : 0,
              "choices" : [
                 {
                    "value" : 0,
                    "description" : "HDMI video input",
                    "details" : {
                       "ref_class" : "NODE",
                       "ref_type" : "HDMI_DECODER",
                       "ref_index" : 0
                    }
                 },
                 {
                    "value" : 1,
                    "description" : "Frame rate converter",
                    "details" : {
                       "ref_class" : "NODE",
                       "ref_type" : "FRAME_RATE_CONVERTER",
```

```
                        "ref_index" : 0
                     }
                  }
               ]
            }
         },
         "status" : {
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "HDMI_DECODER",
               "ref_index" : 0
            }
         }
      }
   ]
}
```

7.7.23 UART Node (Type UART)

This section describes a Node object with **type** member set to UART.

Refer to section 7.7 Nodes (Node) for details.

7.7.23.1 Configuration (UartNodeConfig)

```
{
   "baud_rate" : Integer,
   "data_bits" : Integer,
   "stop_bits" : Integer,
   "parity" : String
}
```

Member name	Since	Type	Description
baud_rate	2.0	Integer	Baud rate.
data_bits	2.0	Integer	Number of data bits: `6`, `7` or `8`.
stop_bits	2.0	Integer	Number of stop bits: `1` or `2`.
parity	2.0	String	Parity: `NONE`, `ODD` or `EVEN`.

7.7.23.2 Status

None

7.7.23.3 Inputs

Input name	Since	Configurable	Description
network	2.0	No	Associated network subscription

7.7.23.4 Example

The example shown below shows a UART node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
        "type" : "UART",
        "index" : 0,
        "configuration" : {
              "baud_rate" : 57600,
              "data_bits" : 8,
              "stop_bits" : 1,
              "parity" : "NONE"
        },
        "status" : {},
        "inputs" : [{
              "name" : "network",
              "index" : 0,
              "configuration" : {},
              "status" : {
                    "source" : {
                          "ref_class" : "SUBSCRIPTION",
                          "ref_type" : "RS232",
                          "ref_index" : 0
                    }
              }
        }]
}
```

7.7.24 Icron Local USB Extender Node (USB_ICRON_CHIP_LOCAL)

This section describes a Node object with **type** member set to `USB_ICRON_CHIP_LOCAL`. This node represents an Icron USB extender that is a local extender (LEX) by default.

Refer to section 7.7 Nodes (Node) for details.

7.7.24.1 Configuration (UsbIcronChipNodeConfig)

```
{
   "extender_type" : String,
   "program_mode" : String,
   "rs232_port" : Integer
}
```

Member name	Since	Type	Description
extender_type	2.4	String	The type of extender: • `LOCAL` for a local extender (LEX). • `REMOTE` for a remote extender (REX).
program_mode	2.13	String	The currently enabled programming mode for the USB extender. One of the following: FIRMWARE for the extender's firmware EEPROM for the extender's non-volatile configuration memory NONE if programming is disabled Note: This programming mode can be set using the icron program command. Refer to section 5.6.1 "icron program".
rs232_port	2.13	Integer	The (zero-based) number of the RS-232 port through which programming of the USB extender is enabled. If the value of the program_mode member is NONE, this value is undefined.

7.7.24.2 Status (UsbIcronChipNodeStatus)

```
{
   "chip_present" : Boolean,
   "mac_address" : String
}
```

Member name	Since	Type	Description
chip_present	2.0	Boolean	Indicates whether the Icron chip is present. This property is false if the device board has a connector for an Icron USB extender daughter board, but no such daughter board is installed.
mac_address	2.0	String	MAC address of the ICRON chip

7.7.24.3 Inputs

None

7.7.24.4 Example

The example below shows an Icron local USB extender node JSON object as returned as part of the output of the `get device`" command.

Refer to section 5.5 Command get for details.

```
{
        "type" : "USB_ICRON_CHIP_LOCAL",
        "index" : 0,
        "configuration" : {
                "extender_type" : "LOCAL"
        },
        "status" : {
                "chip_present" : true,
                "mac_address" : "001b13000013"
        },
        "inputs" : []
}
```

7.7.25 Icron Remote USB Extender Node (USB_ICRON_CHIP_REMOTE)

This section describes a Node *object* with **type** member set to `USB_ICRON_CHIP_REMOTE`. This node represents an Icron USB extender that is a remote extender (REX) by default.

Refer to section 7.7 Nodes (Node) for details.

7.7.25.1 Configuration

Refer to section 7.7.24.1 Configuration (UsbIcronChipNodeConfig).

7.7.25.2 Status

Refer to section 7.7.24.2 Status (UsbIcronChipNodeStatus).

7.7.25.3 Inputs

None

7.7.25.4 Example

The example below shows an Icron remote USB extender node JSON object as returned as part of the output of the `get device` command.

Refer to section 5.5 Command get for details.

```
{
      "type" : "USB_ICRON_CHIP_REMOTE",
      "index" : 0,
      "configuration" : {
            "extender_type" : "REMOTE"
      },
      "status" : {
            "chip_present" : true,
            "mac_address" : "001b13000013"
      },
      "inputs" : []
}
```

7.7.26 Video Input Node (Type VIDEO_INPUT)

This section describes a Node object with **type** member set to `VIDEO_INPUT`. This node represents a physical video input connector. For more information on Nodes refer to section 7.7 Nodes (Node).

7.7.26.1 Configuration

None

7.7.26.2 Status

```
{
   "connector_type" : String
```

7.7.26.3 }

Member name	Since	Type	Description
connector_type	2.2	String	Indicates type of video input connector and is one of the following: `DISPLAY_PORT` for a DisplayPort connector. `HDMI` for an HDMI connector.

7.7.26.4 Inputs

None

7.7.26.5 Example

The example below shows a video input node JSON object as returned as part of the output of the "`get device`" command.

Refer to section 5.5 Command get for details.

```
{
  "type" : "VIDEO_INPUT",
  "index" : 0,
  "configuration" : {},
  "status" : {
     "connector_type" : "HDMI"
  },
  "inputs" : []
}
```

8 Configurable Device Properties

This section lists all the device properties that can be set using the `set property` command.

Refer to section 5.15.4 set property for details. Note that not all properties are supported by all devices.

As specified in the arguments description of the `set property` command, the key argument to this command is a path that refers to an object member inside the device object, starting at the root of the device object. Refer to section 5.15.4.3 Arguments and section 7 Device Object Definition (DeviceObject) for details.

8.1 Device Configuration

The following table lists the configurable properties in the device configuration section of the device object.

Refer to section 0

A FirmwareDetails object has the following structure:

```
{
   "type" : String,
   "details_available" : Boolean,
   "firmware_version" : String,
   "firmware_rc" : String
}
```

Member name	Since	Type	Description
type	2.13	String	The type of the firmware image being described. One of the following: • PRIMARY for the primary firmware image. • GOLDEN for the golden firmware image.
details_available	2.13	Boolean	Whether information is available for this firmware image.
firmware_version	2.13	String	Version of this firmware image. If the value of the **details_available** member is false, this value is undefined.
firmware_rc	2.13	String	Release candidate number of this firmware image. If the value of the **details_available** member is false, this value is undefined. **Note:** For Semtech AptoVision Product Group internal use only.

Device configuration (DeviceConfig)
for details.

Property key		
configuration.device_name		
	Since	2.0
	Type	String
	Description	Device name, maximum 19 characters

Property key		
		Note: This property can only be set for a single device at a time.

8.2 Stream Configuration

The following table lists the configurable properties in the stream configuration section of the device object.

Refer to section 7.5.1 Stream Configuration (StreamConfig) for details.

Property key		
streams[stream_type:index].configuration.source.value		
	Since	2.8
	Type	integer
	Description	Stream source. Currently only for HDMI stream 0 to select between original and frame rate converted versions on devices that support it.

8.3 Analog Audio Input/Output Node Configuration

The following table lists the configurable properties of an analog audio input/output node of the device object.

Refer to section 7.7.3 Analog Audio Input/Output Node (Type ANALOG_AUDIO_INPUT_OUTPUT) for details.

Property key		
nodes[ANALOG_AUDIO_INPUT_OUTPUT:index].configuration.direction		
	Since	2.1
	Type	string
	Description	Direction of the analog audio port: • `INPUT` to set the port as an input • `OUTPUT` to set the port as an output
nodes[ANALOG_AUDIO_INPUT_OUTPUT:index].inputs[main:0].configuration.source.value		
	Since	2.10
	Type	integer
	Description	Audio input source. A device may support none, some or all of the following choices: `2` -- local HDMI audio (stereo downmix). `8` -- I2S audio subscription (audio return channel).

8.4 Analog Audio Output Node Configuration

The following table lists the configurable properties of an analog audio output node of the device object.

Refer to section 7.7.4 Analog Audio Output Node (Type ANALOG_AUDIO_OUTPUT) for details.

Property key		
nodes[ANALOG_AUDIO_OUTPUT:index].inputs[main:0].configuration.source.value		
	Since	2.0
	Type	integer

Property key		
	Description	Audio input source. A device may support none, some or all of the following choices: 2 -- HDMI audio (stereo downmix) 3 -- Analog audio 8 -- I2S audio subscription 9 -- I2S audio local loop out

8.5 Button Node Configuration

The following table lists the configurable properties of a button node of the device object.

Refer to section 7.7.5 Button Node (Type BUTTON) for details.

Property key		
nodes[BUTTON:index].configuration.action.value		
	Since	2.0
	Type	integer
	Description	Action triggered when the button is pressed

8.6 Frame Buffer Node Configuration

The following table lists the configurable properties of a frame buffer node of the device object.

Refer to section 7.7.8 Frame Buffer Node (Type FRAME_BUFFER) for details.

Property key		
nodes[FRAME_BUFFER:index].configuration.width		
	Since	2.0
	Type	integer
	Description	Video width
nodes[FRAME_BUFFER:index].configuration.height		
	Since	2.0
	Type	integer
	Description	Video height
nodes[FRAME_BUFFER:index].configuration.horiz_offset		
	Since	2.0
	Type	integer
	Description	Video horizontal offset
nodes[FRAME_BUFFER:index].configuration.vert_offset		
	Since	2.0
	Type	integer
	Description	Video vertical offset

Property key		
nodes[FRAME_BUFFER:index].configuration.frames_per_second		
	Since	2.0
	Type	integer
	Description	Frame rate

8.7 HDMI Decoder Node Configuration

The following table lists the configurable properties of an HDMI decoder node of the device object.

See section 7.7.11 HDMI Decoder Node (Type HDMI_DECODER) for details.

Property key		
nodes[HDMI_DECODER:index].inputs[main:0].configuration.source.value		
	Since	2.0
	Type	integer
	Description	Video source among available video connectors.
nodes[HDMI_DECODER:index].configuration.hdcp_support_enable		
	Since	2.5
	Type	boolean
	Description	Enable or disable HDCP support.
nodes[HDMI_DECODER:index].configuration.hdcp_22_support_disable		
	Since	2.6
	Type	boolean
	Description	Disable HDCP 2.2 support (report HDCP 1.4 support only).

8.8 HDMI Encoder Node Configuration

The following table lists the configurable properties of an HDMI encoder node of the device object.

Refer to section 7.7.12 HDMI Encoder Node (Type HDMI_ENCODER) for details.

Property key		
nodes[HDMI_ENCODER:*index*].inputs[audio:0].configuration.source.value		
	Since	2.0
	Type	integer
	Description	HDMI audio source. A device may support none, some or all of the following choices: 2 -- HDMI audio (original audio from video subscription in genlock mode) 3 -- Analog audio 6 -- HDMI audio (stereo downmix) 7 -- HDMI audio (all available channels) 8 -- I2S audio subscription 9 -- I2S audio local loop out

8.9 I2S Audio Input Node Configuration

The following table lists the configurable properties of an I2S audio input node of the device object.

Refer to section 7.7.14 I2S Audio Input Node (Type I2S_AUDIO_INPUT) for details.

Property key		
nodes[I2S_AUDIO_INPUT:*index*].configuration.sampling_frequency.value		
	Since	2.10
	Type	integer
	Description	Audio sampling frequency
nodes[I2S_AUDIO_INPUT:*index*].configuration.number_of_channels.value		
	Since	2.10
	Type	integer
	Description	Number of audio channels

8.10 I2S Audio Output Node Configuration

The following table lists the configurable properties of an I2S audio output node of the device object.

Refer to section 7.7.15 I2S Audio Output Node (Type I2S_AUDIO_OUTPUT) for details.

Property key		
nodes[I2S_AUDIO_OUTPUT:*index*].inputs[main:0].configuration.source.value		
	Since	2.9
	Type	integer
	Description	Audio source. A transmitter device may support none, some or all of the following choices: 3 -- Analog audio local loop out 6 -- HDMI audio (stereo downmix) local loop out 7 -- HDMI audio (all channels) local loop out 8 -- I2S audio subscription (audio return channel) A receiver device may support none, some or all of the following choices: 3 -- Analog audio 6 -- HDMI audio (stereo downmix) 7 -- HDMI audio (all channels) 8 -- I2S audio subscription

8.11 LED Node Configuration

The following table lists the configurable properties of a led node of the device object.

Refer to section 7.7.18 LED Node (Type LED) for details.

Property key		
nodes[LED:*index*].configuration.function.value		
	Since	2.0
	Type	integer
	Description	When setting this property using the `set property` command, the following values are allowed: `1` -- blinking LED `2` -- power LED `3` -- video is stable indicator `4` -- network TX indicator `5` -- network RX indicator `6` -- always ON `7` -- always OFF `8` -- default function for this LED **Note:** Not all functions are available for all LEDs. The valid choices are the ones listed in the choices member.

8.12 Network Switch Node Configuration

The following table lists the configurable properties of a network switch node of the device object.

Refer to section 7.7.20 Network Switch Node (Type NETWORK_SWITCH) for details.

Property key		
nodes[NETWORK_SWITCH:*index*].configuration.gigabit_port_1_enable		
	Since	2.0
	Type	boolean
	Description	Enable/disable first Gigabit Ethernet port on the device

8.13 Scaler Node Configuration

The following table lists the configurable properties of a scaler node of the device object.

Refer to section 7.7.22 Scaler Node (Type SCALER) for details.

Property key		
nodes[SCALER:*index*].inputs[main:0].configuration.source.value		
	Since	2.8
	Type	integer
	Description	Main video source.

8.14 UART Node Configuration

The following table lists the configurable properties of a UART node of the device object.

Refer to section 7.7.23 UART Node (Type UART) for details.

Property key		
nodes[UART:*index*].configuration.baud_rate		
	Since	2.0
	Type	integer
	Description	Baud rate
nodes[UART:*index*].configuration.data_bits		
	Since	2.0
	Type	integer
	Description	Number of data bits: 6, 7 or 8
nodes[UART:*index*].configuration.stop_bits		
	Since	2.0
	Type	integer
	Description	Number of stop bits: 1 or 2
nodes[UART:*index*].configuration.parity		
	Since	2.0
	Type	string
	Description	Parity: `NONE`, `ODD` or `EVEN`

8.15 Icron Local USB Extender Node Configuration

The following table lists the configurable properties of an Icron local USB extender node of the device object.

Refer to section 7.7.24 Icron Local USB Extender Node (USB_ICRON_CHIP_LOCAL) for details.

Property key		
nodes[USB_ICRON_CHIP_LOCAL:*index*].configuration.extender_type		
	Since	2.4
	Type	string
	Description	Extender type: `LOCAL` for a local extender (LEX) `REMOTE` for a remote extender (REX)

8.16 Icron Remote USB Extender Node Configuration

The following table lists the configurable properties of an Icron remote USB extender node of the device object.

Refer to section 7.7.25 Icron Remote USB Extender Node (USB_ICRON_CHIP_REMOTE) for details.

Property key		
nodes[USB_ICRON_CHIP_REMOTE:*index*].configuration.extender_type		
	Since	2.4
	Type	string

Property key		
	Description	Extender type: `LOCAL` for a local extender (LEX) `REMOTE` for a remote extender (REX)

Appendix A Get Command Output Examples

The following sections provide example outputs for the `get` command.

For details on the get command refer to section 5.5 Command get.

A.1 Command "get device" for Transmitter Device

The example provided below is the output for the command `get device` called with a transmitter device target argument.

Refer to section 5.5 Command get for details.

```
get d8803962e132 device
{
   "status" : "PROCESSING",
   "request_id" : 19,
   "result" : null,
   "error" : null
}
request 19
{
   "status" : "SUCCESS",
   "request_id" : 19,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "identity" : {
               "engine" : "PLETHORA",
               "vendor_id" : 0,
               "product_id" : 0,
               "firmware_comment" : "",
               "firmware_version" : "3.4.0.0",
               "firmware_rc" : "25",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "configuration" : {
               "device_name" : "PHILIPPE_FCV2_TX"
            },
            "status" : {
               "active" : true,
               "point_to_point" : false,
               "temperature" : 78
            },
            "streams" : [
               {
                  "type" : "HDMI",
                  "index" : 0,
                  "configuration" : {
                     "address" : "0.0.0.0",
                     "enable" : false,
                     "source" : {
                        "value" : 0,
                        "choices" : [
```

```
                {
                   "value" : 0,
                   "description" : "HDMI video input",
                   "details" : {
                      "ref_class" : "NODE",
                      "ref_type" : "HDMI_DECODER",
                      "ref_index" : 0
                   }
                },
                {
                   "value" : 1,
                   "description" : "Frame rate converter",
                   "details" : {
                      "ref_class" : "NODE",
                      "ref_type" : "FRAME_RATE_CONVERTER",
                      "ref_index" : 0
                   }
                }
             ]
          }
       },
       "status" : {
          "state" : "STOPPED"
       },
       "source" : {
          "ref_class" : "NODE",
          "ref_type" : "HDMI_DECODER",
          "ref_index" : 0
       }
    },
    {
       "type" : "HDMI",
       "index" : 1,
       "configuration" : {
          "address" : "0.0.0.0",
          "enable" : false
       },
       "status" : {
          "state" : "STOPPED"
       },
       "source" : {
          "ref_class" : "NODE",
          "ref_type" : "SCALER",
          "ref_index" : 0
       }
    },
    {
       "type" : "HDMI_AUDIO",
       "index" : 0,
       "configuration" : {
          "address" : "224.1.1.10",
          "enable" : true
       },
       "status" : {
          "state" : "STREAMING"
       },
```

```
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "HDMI_DECODER",
               "ref_index" : 0
            }
         },
         {
            "type" : "AUDIO",
            "index" : 0,
            "configuration" : {
               "address" : "0.0.0.0",
               "enable" : false
            },
            "status" : {
               "state" : "STOPPED"
            },
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "ANALOG_AUDIO_INPUT",
               "ref_index" : 0
            }
         },
         {
            "type" : "RS232",
            "index" : 0,
            "configuration" : {
               "address" : "169.254.234.228",
               "enable" : true
            },
            "status" : {
               "state" : "STREAMING"
            },
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "UART",
               "ref_index" : 0
            }
         },
         {
            "type" : "INFRARED",
            "index" : 0,
            "configuration" : {
               "address" : "0.0.0.0",
               "enable" : false
            },
            "status" : {
               "state" : "STOPPED"
            },
            "source" : {
               "ref_class" : "NODE",
               "ref_type" : "INFRARED_DECODER",
               "ref_index" : 0
            }
         }
      ],
```

```
            "subscriptions" : [
               {
                  "type" : "I2S_AUDIO",
                  "index" : 0,
                  "configuration" : {
                     "address" : "0.0.0.0",
                     "enable" : false
                  },
                  "status" : {
                     "state" : "STOPPED"
                  }
               },
               {
                  "type" : "RS232",
                  "index" : 0,
                  "configuration" : {
                     "address" : "0.0.0.0",
                     "enable" : true
                  },
                  "status" : {
                     "state" : "STREAMING"
                  }
               },
               {
                  "type" : "INFRARED",
                  "index" : 0,
                  "configuration" : {
                     "address" : "0.0.0.0",
                     "enable" : true
                  },
                  "status" : {
                     "state" : "STREAMING"
                  }
               }
            ],
            "nodes" : [
               {
                  "type" : "ANALOG_AUDIO_INPUT_OUTPUT",
                  "index" : 0,
                  "configuration" : {
                     "direction" : "INPUT"
                  },
                  "status" : {},
                  "inputs" : [
                     {
                        "name" : "main",
                        "index" : 0,
                        "configuration" : {
                           "source" : {
                              "value" : 2,
                              "choices" : [
                                 {
                                    "value" : 2,
                                    "description" : "HDMI Audio Local Loop
Out (Downmix)",
                                    "details" : {
```

```
                                    "ref_class" : "NODE",
                                    "ref_type" : "HDMI_DECODER",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 8,
                                 "description" : "I2S Audio (Audio
Return Channel)",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "I2S_AUDIO",
                                    "ref_index" : 0
                                 }
                              }
                           ]
                        }
                     },
                     "status" : {
                        "source" : {
                           "ref_class" : "NODE",
                           "ref_type" : "HDMI_DECODER",
                           "ref_index" : 0
                        }
                     }
                  }
               ]
            },
            {
               "type" : "BUTTON",
               "index" : 0,
               "configuration" : {
                  "action" : {
                     "value" : 0,
                     "choices" : [
                        {
                           "value" : 0,
                           "description" : "No action",
                           "details" : "NONE"
                        },
                        {
                           "value" : 3,
                           "description" : "Switch video input",
                           "details" : "SWITCH_VIDEO_INPUT"
                        }
                     ]
                  }
               },
               "status" : {},
               "inputs" : []
            },
            {
               "type" : "BUTTON",
               "index" : 1,
               "configuration" : {
```

```
            "action" : {
               "value" : 0,
               "choices" : [
                  {
                     "value" : 0,
                     "description" : "No action",
                     "details" : "NONE"
                  }
               ]
            }
         },
         "status" : {},
         "inputs" : []
      },
      {
         "type" : "FRAME_RATE_CONVERTER",
         "index" : 0,
         "configuration" : {},
         "status" : {
            "divide_factor" : 2
         },
         "inputs" : [
            {
               "name" : "main",
               "index" : 0,
               "configuration" : {},
               "status" : {
                  "source" : {
                     "ref_class" : "NODE",
                     "ref_type" : "HDMI_DECODER",
                     "ref_index" : 0
                  }
               }
            }
         ]
      },
      {
         "type" : "GPIO",
         "index" : 0,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 1,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
```

```
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 2,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 3,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 4,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 5,
         "configuration" : {
            "tristate" : 0,
            "cfg_val" : 0
         },
         "status" : {
            "stat_val" : 1
         },
         "inputs" : []
      },
      {
         "type" : "GPIO",
         "index" : 6,
```

```
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 7,
      "configuration" : {
         "tristate" : 1,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 8,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 9,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 10,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
```

```
},
{
   "type" : "GPIO",
   "index" : 11,
   "configuration" : {
      "tristate" : 1,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 12,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 13,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 14,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   },
   "inputs" : []
},
{
   "type" : "HDMI_DECODER",
   "index" : 0,
   "configuration" : {
```

```
                    "edid" :
"00ffffffffffff000614000156524c420816010380341d780a01c1a057479827124c4c210
8008140010101010101010101010101010101023a801871382d40582c460040846300001e08e
80030f2705a80b0588a00ba892100001e000000fc004170746f566973696f6e20200a00000
0fd0018550e853c000a2020202020200120020340735801020304051112131 41f202122060
7905d5e5f6263646061230d0707830f00006d030c001000383c20406801020367d85dc4017
88003e40f0000c0011d00bc52d01e20b828554040846300001e023a80d072382d40102c458
040846300001e023a801871382d40582c450040846300001e0000000000000000000f2",
                    "hdcp_support_enable" : true,
                    "hdcp_22_support_disable" : false
                },
                "status" : {
                    "video" : {
                        "width" : 1920,
                        "height" : 1080,
                        "frames_per_second" : 60,
                        "color_space" : "RGB",
                        "bits_per_pixel" : 8,
                        "scan_mode" : "PROGRESSIVE"
                    },
                    "has_video_details" : true,
                    "video_details" : {
                        "pixel_clock" : 148500000,
                        "total_width" : 2200,
                        "total_height" : 1125,
                        "hsync_width" : 44,
                        "hsync_front_porch" : 88,
                        "hsync_negative" : false,
                        "vsync_width" : 5,
                        "vsync_front_porch" : 3,
                        "vsync_negative" : false,
                        "vic" : 16,
                        "has_hdmi_vic" : false,
                        "hdmi_vic" : 0,
                        "picture_aspect" : "ASPECT_16_9",
                        "has_active_format" : true,
                        "active_format" : 8,
                        "it_content_type" : "NO_DATA",
                        "scan_information" : "NO_DATA",
                        "colorimetry" : "BT709",
                        "rgb_range" : "DEFAULT",
                        "ycc_range" : "LIMITED"
                    },
                    "has_audio_details" : true,
                    "audio_details" : {
                        "sampling_frequency" : 44100,
                        "number_of_channels" : 2
                    },
                    "hdcp_protected" : false,
                    "hdcp_version" : "NONE",
                    "hdmi_2_0_support" : true,
                    "source_stable" : true,
                    "audio_downmix_support" : true
                },
                "inputs" : [
                    {
```

```
                        "name" : "main",
                        "index" : 0,
                        "configuration" : {
                           "source" : {
                              "value" : 0,
                              "choices" : [
                                 {
                                    "value" : 0,
                                    "description" : "HDMI Input 1",
                                    "details" : {
                                       "ref_class" : "NODE",
                                       "ref_type" : "VIDEO_INPUT",
                                       "ref_index" : 0
                                    }
                                 },
                                 {
                                    "value" : 1,
                                    "description" : "Display Port Input
1",
                                    "details" : {
                                       "ref_class" : "NODE",
                                       "ref_type" : "VIDEO_INPUT",
                                       "ref_index" : 1
                                    }
                                 },
                                 {
                                    "value" : 32,
                                    "description" : "Automatic Input
Selection (HDMI Input 1)",
                                    "details" : {
                                       "ref_class" : "AUTOMATIC",
                                       "ref_type" : null,
                                       "ref_index" : 0
                                    }
                                 }
                              ]
                           }
                        },
                        "status" : {
                           "source" : {
                              "ref_class" : "NODE",
                              "ref_type" : "VIDEO_INPUT",
                              "ref_index" : 0
                           }
                        }
                     }
                  ]
               },
               {
                  "type" : "INFRARED_ENCODER",
                  "index" : 0,
                  "configuration" : {},
                  "status" : {
                     "data_length_max" : 1032
                  },
```

```
      "inputs" : [
         {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "INFRARED",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "INFRARED_DECODER",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "LED",
      "index" : 0,
      "configuration" : {
         "function" : {
            "value" : 0,
            "choices" : [
               {
                  "value" : 0,
                  "description" : "(Unknown)",
                  "details" : "UNKNOWN"
               },
               {
                  "value" : 1,
                  "description" : "Blink LED",
                  "details" : "BLINK"
               },
               {
                  "value" : 8,
                  "description" : "Default functionality",
                  "details" : "DEFAULT"
               }
            ]
         }
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "NETWORK_INTERFACE",
      "index" : 0,
      "configuration" : {
         "hostname" : "PHILIPPE_FCV2_TX",
         "ip" : {
```

```
            "address" : "169.254.51.225",
            "mode" : "DHCP",
            "mask" : "255.255.0.0",
            "gateway" : "0.0.0.0"
         }
      },
      "status" : {
         "mac_address" : "d8803962e132",
         "ip" : {
            "address" : "169.254.51.225"
         }
      },
      "inputs" : []
   },
   {
      "type" : "NETWORK_SWITCH",
      "index" : 0,
      "configuration" : {
         "gigabit_port_1_enable" : true
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "SCALER",
      "index" : 0,
      "configuration" : {
         "width" : 0,
         "height" : 0
      },
      "status" : {
         "video" : {
            "width" : 0,
            "height" : 0,
            "frames_per_second" : 60,
            "color_space" : "RGB",
            "bits_per_pixel" : 8,
            "scan_mode" : "PROGRESSIVE"
         }
      },
      "inputs" : [
         {
            "name" : "main",
            "index" : 0,
            "configuration" : {
               "source" : {
                  "value" : 0,
                  "choices" : [
                     {
                        "value" : 0,
                        "description" : "HDMI video input",
                        "details" : {
                           "ref_class" : "NODE",
                           "ref_type" : "HDMI_DECODER",
                           "ref_index" : 0
```

```
                                    }
                                },
                                {
                                    "value" : 1,
                                    "description" : "Frame rate
converter",
                                    "details" : {
                                        "ref_class" : "NODE",
                                        "ref_type" :
"FRAME_RATE_CONVERTER",
                                        "ref_index" : 0
                                    }
                                }
                            ]
                        }
                    },
                    "status" : {
                        "source" : {
                            "ref_class" : "NODE",
                            "ref_type" : "HDMI_DECODER",
                            "ref_index" : 0
                        }
                    }
                }
            ]
        },
        {
            "type" : "UART",
            "index" : 0,
            "configuration" : {
                "baud_rate" : 57600,
                "data_bits" : 8,
                "stop_bits" : 1,
                "parity" : "NONE"
            },
            "status" : {},
            "inputs" : [
                {
                    "name" : "network",
                    "index" : 0,
                    "configuration" : {},
                    "status" : {
                        "source" : {
                            "ref_class" : "SUBSCRIPTION",
                            "ref_type" : "RS232",
                            "ref_index" : 0
                        }
                    }
                }
            ]
        },
        {
            "type" : "USB_ICRON_CHIP_LOCAL",
            "index" : 0,
            "configuration" : {
                "extender_type" : "LOCAL"
```

```
                },
                "status" : {
                   "chip_present" : true,
                   "mac_address" : "001b1300000c"
                },
                "inputs" : []
             },
             {
                "type" : "VIDEO_INPUT",
                "index" : 0,
                "configuration" : {},
                "status" : {
                   "connector_type" : "HDMI"
                },
                "inputs" : []
             },
             {
                "type" : "VIDEO_INPUT",
                "index" : 1,
                "configuration" : {},
                "status" : {
                   "connector_type" : "DISPLAY_PORT"
                },
                "inputs" : []
             }
          ]
       }
    ],
    "error" : []
 },
 "error" : null
}
```


A.2 Command "get device" for Receiver Device

The following is an example output for the command **get device** called with a receiver device target argument.

Refer to section 5.5 Command get for details.


```
get d88039631d77 device
{
   "status" : "PROCESSING",
   "request_id" : 20,
   "result" : null,
   "error" : null
}
request 20
{
   "status" : "SUCCESS",
   "request_id" : 20,
   "result" : {
      "devices" : [
         {
            "device_id" : "d88039631d77",
            "identity" : {
               "engine" : "PLETHORA",
```

```
      "vendor_id" : 0,
      "product_id" : 0,
      "firmware_comment" : "",
      "firmware_version" : "3.4.0.0",
      "firmware_rc" : "25",
      "is_receiver" : true,
      "is_transmitter" : false
   },
   "configuration" : {
      "device_name" : "PHILIPPE_FCV2_RX"
   },
   "status" : {
      "active" : true,
      "point_to_point" : false,
      "temperature" : 77
   },
   "streams" : [
      {
         "type" : "RS232",
         "index" : 0,
         "configuration" : {
            "address" : "169.254.234.228",
            "enable" : true
         },
         "status" : {
            "state" : "STREAMING"
         },
         "source" : {
            "ref_class" : "NODE",
            "ref_type" : "UART",
            "ref_index" : 0
         }
      },
      {
         "type" : "INFRARED",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         },
         "source" : {
            "ref_class" : "NODE",
            "ref_type" : "INFRARED_DECODER",
            "ref_index" : 0
         }
      }
   ],
   "subscriptions" : [
      {
         "type" : "HDMI",
         "index" : 0,
         "configuration" : {
            "address" : "224.1.2.3",
```

```
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 1,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 2,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 3,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 4,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 5,
         "configuration" : {
            "address" : "0.0.0.0",
```

```
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 6,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 7,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 8,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 9,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 10,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
```


Semtech
Proprietary & Confidential

```
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 11,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 12,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 13,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 14,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 15,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
```

```
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 16,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 17,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 18,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 19,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 20,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
```

```
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 21,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 22,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 23,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 24,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 25,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
```

```
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 26,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 27,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 28,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 29,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 30,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
```

```
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 31,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI_AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "I2S_AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "RS232",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : true
         },
         "status" : {
```

```
                  "state" : "STREAMING"
               }
            },
            {
               "type" : "INFRARED",
               "index" : 0,
               "configuration" : {
                  "address" : "0.0.0.0",
                  "enable" : true
               },
               "status" : {
                  "state" : "STREAMING"
               }
            }
         ],
         "nodes" : [
            {
               "type" : "ANALOG_AUDIO_OUTPUT",
               "index" : 0,
               "configuration" : {},
               "status" : {},
               "inputs" : [
                  {
                     "name" : "main",
                     "index" : 0,
                     "configuration" : {
                        "source" : {
                           "value" : 3,
                           "choices" : [
                              {
                                 "value" : 2,
                                 "description" : "HDMI Audio
(Downmix)",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "HDMI_AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 3,
                                 "description" : "Analog Audio",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 8,
                                 "description" : "I2S Audio",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "I2S_AUDIO",
                                    "ref_index" : 0
                                 }
```

```
                                    }
                                ]
                            }
                        },
                        "status" : {
                            "source" : {
                                "ref_class" : "SUBSCRIPTION",
                                "ref_type" : "AUDIO",
                                "ref_index" : 0
                            }
                        }
                    }
                ]
            },
            {
                "type" : "BUTTON",
                "index" : 0,
                "configuration" : {
                    "action" : {
                        "value" : 0,
                        "choices" : [
                            {
                                "value" : 0,
                                "description" : "No action",
                                "details" : "NONE"
                            },
                            {
                                "value" : 1,
                                "description" : "Send monitor EDID to all
transmitter devices",
                                "details" : "SEND_EDID"
                            }
                        ]
                    }
                },
                "status" : {},
                "inputs" : []
            },
            {
                "type" : "BUTTON",
                "index" : 1,
                "configuration" : {
                    "action" : {
                        "value" : 0,
                        "choices" : [
                            {
                                "value" : 0,
                                "description" : "No action",
                                "details" : "NONE"
                            }
                        ]
                    }
                },
                "status" : {},
                "inputs" : []
```

```
},
{
   "type" : "FRAME_BUFFER",
   "index" : 0,
   "configuration" : {
      "display_mode" : "FAST_SWITCHED",
      "width" : 1920,
      "height" : 1080,
      "horiz_offset" : 0,
      "vert_offset" : 0,
      "frames_per_second" : 60
   },
   "status" : {
      "source_video" : {
         "width" : 1920,
         "height" : 1080,
         "frames_per_second" : 60,
         "color_space" : "RGB",
         "bits_per_pixel" : 8,
         "scan_mode" : "PROGRESSIVE"
      }
   },
   "inputs" : [
      {
         "name" : "main",
         "index" : 0,
         "configuration" : {},
         "status" : {
            "source" : {
               "ref_class" : "SUBSCRIPTION",
               "ref_type" : "HDMI",
               "ref_index" : 0
            }
         }
      },
      {
         "name" : "multiview",
         "index" : 0,
         "configuration" : {},
         "status" : {
            "source" : {
               "ref_class" : "SUBSCRIPTION",
               "ref_type" : "HDMI",
               "ref_index" : 0
            }
         }
      },
      {
         "name" : "multiview",
         "index" : 1,
         "configuration" : {},
         "status" : {
            "source" : {
               "ref_class" : "SUBSCRIPTION",
               "ref_type" : "HDMI",
               "ref_index" : 1
```

```
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 2,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 2
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 3,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 3
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 4,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 4
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 5,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 5
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 6,
        "configuration" : {},
```

```
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 6
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 7,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 7
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 8,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 8
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 9,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 9
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 10,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 10
         }
      }
   },
```


Semtech
Proprietary & Confidential

```
{
   "name" : "multiview",
   "index" : 11,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 11
      }
   }
},
{
   "name" : "multiview",
   "index" : 12,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 12
      }
   }
},
{
   "name" : "multiview",
   "index" : 13,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 13
      }
   }
},
{
   "name" : "multiview",
   "index" : 14,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 14
      }
   }
},
{
   "name" : "multiview",
   "index" : 15,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
```

```
            "ref_type" : "HDMI",
            "ref_index" : 15
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 16,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 16
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 17,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 17
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 18,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 18
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 19,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 19
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 20,
```

```
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 20
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 21,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 21
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 22,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 22
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 23,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 23
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 24,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 24
         }
```

```
      }
   },
   {
      "name" : "multiview",
      "index" : 25,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 25
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 26,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 26
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 27,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 27
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 28,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 28
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 29,
      "configuration" : {},
      "status" : {
         "source" : {
```

```
                    "ref_class" : "SUBSCRIPTION",
                    "ref_type" : "HDMI",
                    "ref_index" : 29
                }
            }
        },
        {
            "name" : "multiview",
            "index" : 30,
            "configuration" : {},
            "status" : {
                "source" : {
                    "ref_class" : "SUBSCRIPTION",
                    "ref_type" : "HDMI",
                    "ref_index" : 30
                }
            }
        },
        {
            "name" : "multiview",
            "index" : 31,
            "configuration" : {},
            "status" : {
                "source" : {
                    "ref_class" : "SUBSCRIPTION",
                    "ref_type" : "HDMI",
                    "ref_index" : 31
                }
            }
        }
    ]
},
{
    "type" : "GPIO",
    "index" : 0,
    "configuration" : {
        "tristate" : 0,
        "cfg_val" : 0
    },
    "status" : {
        "stat_val" : 1
    },
    "inputs" : []
},
{
    "type" : "GPIO",
    "index" : 1,
    "configuration" : {
        "tristate" : 0,
        "cfg_val" : 0
    },
    "status" : {
        "stat_val" : 0
    },
    "inputs" : []
```

```
},
{
   "type" : "GPIO",
   "index" : 2,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 3,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 4,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 5,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   },
   "inputs" : []
},
{
   "type" : "GPIO",
   "index" : 6,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
```

```
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 7,
      "configuration" : {
         "tristate" : 1,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 8,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 9,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 10,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
```

```
      "index" : 11,
      "configuration" : {
         "tristate" : 1,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 1
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 12,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 13,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "GPIO",
      "index" : 14,
      "configuration" : {
         "tristate" : 0,
         "cfg_val" : 0
      },
      "status" : {
         "stat_val" : 0
      },
      "inputs" : []
   },
   {
      "type" : "HDMI_ENCODER",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "video" : {
            "width" : 1920,
            "height" : 1080,
            "frames_per_second" : 60,
            "color_space" : "RGB",
```

```
            "bits_per_pixel" : 8,
            "scan_mode" : "PROGRESSIVE"
         },
         "has_video_details" : true,
         "video_details" : {
            "pixel_clock" : 0,
            "total_width" : 0,
            "total_height" : 0,
            "hsync_width" : 0,
            "hsync_front_porch" : 0,
            "hsync_negative" : false,
            "vsync_width" : 0,
            "vsync_front_porch" : 0,
            "vsync_negative" : false,
            "vic" : 16,
            "has_hdmi_vic" : false,
            "hdmi_vic" : 0,
            "picture_aspect" : "ASPECT_16_9",
            "has_active_format" : true,
            "active_format" : 8,
            "it_content_type" : "NO_DATA",
            "scan_information" : "NO_DATA",
            "colorimetry" : "BT709",
            "rgb_range" : "DEFAULT",
            "ycc_range" : "LIMITED"
         },
         "has_audio_details" : false,
         "audio_details" : {},
         "hdcp_protected" : false,
         "hdcp_version" : "NONE",
         "hdmi_2_0_support" : true,
         "source_stable" : false
      },
      "inputs" : [
         {
            "name" : "main",
            "index" : 0,
            "configuration" : {
               "source" : {
                  "value" : 0,
                  "choices" : [
                     {
                        "value" : 0,
                        "description" : "Genlock Mode",
                        "details" : {
                           "ref_class" : "SUBSCRIPTION",
                           "ref_type" : "HDMI",
                           "ref_index" : 0
                        }
                     },
                     {
                        "value" : 1,
                        "description" : "Frame Buffer Mode",
                        "details" : {
                           "ref_class" : "NODE",
```

```
                                    "ref_type" : "FRAME_BUFFER",
                                    "ref_index" : 0
                                 }
                              }
                           ]
                        }
                     },
                     "status" : {
                        "source" : {
                           "ref_class" : "SUBSCRIPTION",
                           "ref_type" : "HDMI",
                           "ref_index" : 0
                        }
                     }
                  },
                  {
                     "name" : "audio",
                     "index" : 0,
                     "configuration" : {
                        "source" : {
                           "value" : 2,
                           "choices" : [
                              {
                                 "value" : 2,
                                 "description" : "HDMI Audio (from HDMI
Video in Genlock Mode)",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "HDMI_AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 3,
                                 "description" : "Analog Audio",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 6,
                                 "description" : "HDMI Audio (Stereo
Downmix)",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "HDMI_AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 7,
                                 "description" : "HDMI Audio (All
Available Channels)",
                                 "details" : {
```

```
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "HDMI_AUDIO",
                                    "ref_index" : 0
                                 }
                              },
                              {
                                 "value" : 8,
                                 "description" : "I2S Audio",
                                 "details" : {
                                    "ref_class" : "SUBSCRIPTION",
                                    "ref_type" : "I2S_AUDIO",
                                    "ref_index" : 0
                                 }
                              }
                           ]
                        }
                     },
                     "status" : {
                        "source" : {
                           "ref_class" : "SUBSCRIPTION",
                           "ref_type" : "HDMI_AUDIO",
                           "ref_index" : 0
                        }
                     }
                  }
               ]
            },
            {
               "type" : "HDMI_MONITOR",
               "index" : 0,
               "configuration" : {},
               "status" : {
                  "connected" : true,
                  "edid" :
"00ffffffffffff004c2d5706333244590a140103801009782aee91a3544c99260f5054230
800a9408180814081009500b30001010101023a801871382d40582c4500a05a0000001e011
d007251d01e206e285500a05a0000001e000000fd00323c1b5111000a20202020202000000
0fc0053796e634d61737465720a2020013b02031cf34890041f05141303122309070783010
00066030c00100080011d80d0721c1620102c2580a05a0000009e011d8018711c1620582c2
500a05a0000009e011d00bc52d01e20b8285540a05a0000001e011d007251d01e206e28550
0a05a0000001e8c0ad090204031200c405500a05a000000180000000000000000000075"
               },
               "inputs" : [
                  {
                     "name" : "main",
                     "index" : 0,
                     "configuration" : {},
                     "status" : {
                        "source" : {
                           "ref_class" : "NODE",
                           "ref_type" : "HDMI_ENCODER",
                           "ref_index" : 0
                        }
                     }
                  }
```

```
      ]
   },
   {
      "type" : "INFRARED_ENCODER",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "data_length_max" : 1032
      },
      "inputs" : [
         {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "INFRARED",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "INFRARED_DECODER",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "LED",
      "index" : 0,
      "configuration" : {
         "function" : {
            "value" : 0,
            "choices" : [
               {
                  "value" : 0,
                  "description" : "(Unknown)",
                  "details" : "UNKNOWN"
               },
               {
                  "value" : 1,
                  "description" : "Blink LED",
                  "details" : "BLINK"
               },
               {
                  "value" : 8,
                  "description" : "Default functionality",
                  "details" : "DEFAULT"
               }
            ]
         }
      },
```

```
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "NETWORK_INTERFACE",
      "index" : 0,
      "configuration" : {
         "hostname" : "PHILIPPE_FCV2_RX",
         "ip" : {
            "address" : "169.254.120.29",
            "mode" : "DHCP",
            "mask" : "255.255.0.0",
            "gateway" : "0.0.0.0"
         }
      },
      "status" : {
         "mac_address" : "d88039631d77",
         "ip" : {
            "address" : "169.254.120.29"
         }
      },
      "inputs" : []
   },
   {
      "type" : "NETWORK_SWITCH",
      "index" : 0,
      "configuration" : {
         "gigabit_port_1_enable" : true
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "UART",
      "index" : 0,
      "configuration" : {
         "baud_rate" : 57600,
         "data_bits" : 8,
         "stop_bits" : 1,
         "parity" : "NONE"
      },
      "status" : {},
      "inputs" : [
         {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "RS232",
                  "ref_index" : 0
               }
            }
         }
```

```
                ]
             },
             {
                "type" : "USB_ICRON_CHIP_REMOTE",
                "index" : 0,
                "configuration" : {
                   "extender_type" : "REMOTE"
                },
                "status" : {
                   "chip_present" : true,
                   "mac_address" : "001b1300001d"
                },
                "inputs" : []
             }
          ]
       }
    ],
    "error" : []
 },
 "error" : null
}
```


A.3 Command "get edid" for Transmitter Device

The following is an example output for the command **get edid** called with a transmitter device target argument.

Refer to section 5.5 Command get for details.


```
get d8803962e132 edid
{
   "status" : "PROCESSING",
   "request_id" : 265,
   "result" : null,
   "error" : null
}
request 265
{
   "status" : "SUCCESS",
   "request_id" : 265,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "status" : {
               "active" : true
            },
            "nodes" : [
               {
                  "type" : "HDMI_DECODER",
                  "index" : 0,
                  "configuration" : {
                     "edid" :
"00ffffffffffff0010ac82d042515230221901038035 1e78ea2195a756529c26105054a54
b00714f8180a9c0d1c00101010101010101023a801871382d40582c45000f282100001e000
000ff0035324d543935384a305251420a000000fc0044454c4c20534532343136480a00000
0fd00384c1e5311000a20202020202001400203 17b14c900504030207160114 1f121365030
c001000023a801871382d40582c45000f282100001e011d8018711c1620582c25000f28210
```

```
0009e011d007251d01e206e2855000f282100001e8c0ad08a20e02d10103e96000f2821000
0180000000000000000000000000000000000000000000000000000000000003d"
                  },
                  "status" : {}
               }
            ]
         }
      ],
      "error" : []
   },
   "error" : null
}
```


A.4 Command "get edid" for Receiver Device

The following is an example output for the command **get edid** called with a receiver device target argument.

Refer to section 5.5 Command get for details.


```
get d88039631d77 edid
{
   "status" : "PROCESSING",
   "request_id" : 284,
   "result" : null,
   "error" : null
}
request 284
{
   "status" : "SUCCESS",
   "request_id" : 284,
   "result" : {
      "devices" : [
         {
            "device_id" : "d88039631d77",
            "status" : {
               "active" : true
            },
            "nodes" : [
               {
                  "type" : "HDMI_MONITOR",
                  "index" : 0,
                  "configuration" : {},
                  "status" : {
                     "connected" : true,
                     "edid" :
"00ffffffffffff0009d11e8045540000031a0103803c22782efcd0a6544a9d240e5054a56
b80d1c081c081008180a9c0b300010101014dd000a0f0703e8030203500544f2100001a000
000ff004b31473030343934534c300a20000000fd001e4c1ea03c000a20202020202000000
0fc0042656e51204c43440a2020202001c402033af253101f010203040506071112131415 1
620615e5f23090707830100006c030c00200038782000400102 67d85dc40178c000e40f000
0014dd000a0f0703e8030203500544f2100001e565e00a0a0a029503020350054 4f2100001
e000000000000000000000000000000000000000000000000000000000000cf"
                  }
               }
            ]
         }
```

```
      ],
      "error" : []
   },
   "error" : null
}
```

A.5 Command “get gpio”

The following is an example output for the command `get gpio`.

Refer to section 5.5 Command get for details.

```
request 23
{
   "status" : "SUCCESS",
   "request_id" : 23,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "status" : {
               "active" : true
            },
            "nodes" : [
               {
                  "type" : "GPIO",
                  "index" : 0,
                  "configuration" : {
                     "tristate" : 0,
                     "cfg_val" : 0
                  },
                  "status" : {
                     "stat_val" : 1
                  }
               },
               {
                  "type" : "GPIO",
                  "index" : 1,
                  "configuration" : {
                     "tristate" : 0,
                     "cfg_val" : 0
                  },
                  "status" : {
                     "stat_val" : 1
                  }
               },
               {
                  "type" : "GPIO",
                  "index" : 2,
                  "configuration" : {
                     "tristate" : 0,
                     "cfg_val" : 0
                  },
                  "status" : {
                     "stat_val" : 1
                  }
               },
```

```
{
   "type" : "GPIO",
   "index" : 3,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 4,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 5,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 6,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   }
},
{
   "type" : "GPIO",
   "index" : 7,
   "configuration" : {
      "tristate" : 1,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   }
},
```

```
{
   "type" : "GPIO",
   "index" : 8,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 9,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 10,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 11,
   "configuration" : {
      "tristate" : 1,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 1
   }
},
{
   "type" : "GPIO",
   "index" : 12,
   "configuration" : {
      "tristate" : 0,
      "cfg_val" : 0
   },
   "status" : {
      "stat_val" : 0
   }
},
{
```

```
                  "type" : "GPIO",
                  "index" : 13,
                  "configuration" : {
                     "tristate" : 0,
                     "cfg_val" : 0
                  },
                  "status" : {
                     "stat_val" : 0
                  }
               },
               {
                  "type" : "GPIO",
                  "index" : 14,
                  "configuration" : {
                     "tristate" : 0,
                     "cfg_val" : 0
                  },
                  "status" : {
                     "stat_val" : 0
                  }
               }
            ]
         }
      ],
      "error" : []
   },
   "error" : null
}
```

A.6 Command "get hello"

The following is an example output for the command **get hello**.

Refer to section 5.5 Command get for details.

```
get 001ec0f03668 hello
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "devices" : [
         {
            "device_id" : "001ec0f03668",
            "identity" : {
               "firmware_version" : "2.9.1",
               "is_receiver" : false,
               "is_transmitter" : true
            },
            "status" : {
               "active" : true
            },
            "nodes" : [
               {
                  "type" : "NETWORK_INTERFACE",
                  "index" : 0,
                  "configuration" : {},
```

```
                  "status" : {
                     "mac_address" : "001ec0f03668",
                     "ip" : {
                        "address" : "10.1.1.15"
                     }
                  }
               }
            ]
         }
      ],
      "error" : []
   },
   "error" : null
}
```

A.7 Command "get identity"

The following is an example output for the command `get identity`.

Refer to section 5.5 Command get for details.

```
get 001ec0f03668 identity
```

A.8 Command "get list"

The following is an example output for the command **get list**. This command is typically called with **all** as a target argument as below in order to list all devices.

Refer to section 5.5 Command get for details.

```
get all list
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "devices" : [
         {
            "device_id" : "001ec0f03668",
            "status" : {
               "active" : true
            }
         },
         {
            "device_id" : "d88039631d77",
            "status" : {
               "active" : true
            }
         },
         {
            "device_id" : "001ec0f03cc1",
            "status" : {
               "active" : true
            }
         },
         {
            "device_id" : "d8803962e132",
            "status" : {
               "active" : true
```

```
            }
         }
      ],
      "error" : [
         {
            "device_id" : "d88039634e85",
            "reason" : "DEVICE_DISCONNECTED",
            "message" : "Device is disconnected from the network"
         },
         {
            "device_id" : "d8803962ffa4",
            "reason" : "DEVICE_DISCONNECTED",
            "message" : "Device is disconnected from the network"
         }
      ]
   },
   "error" : null
}
```

A.9 Command "get settings" for Transmitter Device

The following is an example output for the command **get settings** called with a transmitter device target argument.

Refer to section 5.5 Command get for details.

```
get d8803962e132 settings
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "devices" : [
         {
            "device_id" : "d8803962e132",
            "identity" : {
               "engine" : "PLETHORA",
               "vendor_id" : 1,
               "product_id" : 2,
               "firmware_comment" : "BlueRiver 3.5.0.0rc51 evaluation",
               "firmware_version" : "3.5.0.0",
               "firmware_rc" : "51",
               "is_receiver" : false,
               "is_transmitter" : true,
               "firmware_details" : [
                  {
                     "type" : "PRIMARY",
```

```
            "details_available" : true,
            "firmware_version" : "3.5.0.0",
            "firmware_rc" : "51"
         },
         {
            "type" : "BACKUP",
            "details_available" : true,
            "firmware_version" : "3.5.0.0",
            "firmware_rc" : "45"
         }
      ]
   },
   "configuration" : {
      "device_name" : "PHILIPPE_FCV2_TX",
      "locate_mode" : false
   },
   "status" : {
      "active" : true,
      "point_to_point" : false,
      "boot_status" : "PRIMARY",
      "update_in_progress" : false,
      "error_status" : {
         "has_error_code" : false,
         "error_code" : 0
      }
   },
   "streams" : [
      {
         "type" : "HDMI",
         "index" : 0,
         "configuration" : {
            "address" : "224.1.1.1",
            "enable" : true,
            "auto_stop" : false,
            "source" : {
               "value" : 0
            }
```

```
      },
      "status" : {
         "state" : "STREAMING"
      },
      "source" : {
         "ref_class" : "NODE",
         "ref_type" : "HDMI_DECODER",
         "ref_index" : 0
      }
   },
   {
      "type" : "HDMI",
      "index" : 1,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false,
         "auto_stop" : false
      },
      "status" : {
         "state" : "STOPPED"
      },
      "source" : {
         "ref_class" : "NODE",
         "ref_type" : "SCALER",
         "ref_index" : 0
      }
   },
   {
      "type" : "HDMI_AUDIO",
      "index" : 0,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false,
         "auto_stop" : false
      },
      "status" : {
```

```
            "state" : "STOPPED"
         },
         "source" : {
            "ref_class" : "NODE",
            "ref_type" : "HDMI_DECODER",
            "ref_index" : 0
         }
      },
      {
         "type" : "AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false,
            "auto_stop" : false
         },
         "status" : {
            "state" : "STOPPED"
         },
         "source" : {
            "ref_class" : "NODE",
            "ref_type" : "ANALOG_AUDIO_INPUT",
            "ref_index" : 0
         }
      },
      {
         "type" : "RS232",
         "index" : 0,
         "configuration" : {
            "address" : "169.254.9.93",
            "enable" : true,
            "auto_stop" : false
         },
         "status" : {
            "state" : "STREAMING"
         },
         "source" : {
```

```
            "ref_class" : "NODE",
            "ref_type" : "UART",
            "ref_index" : 0
        }
    },
    {
        "type" : "INFRARED",
        "index" : 0,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false,
            "auto_stop" : false
        },
        "status" : {
            "state" : "STOPPED"
        },
        "source" : {
            "ref_class" : "NODE",
            "ref_type" : "INFRARED_DECODER",
            "ref_index" : 0
        }
    },
    {
        "type" : "CEC",
        "index" : 0,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false,
            "auto_stop" : false
        },
        "status" : {
            "state" : "STOPPED"
        },
        "source" : {
            "ref_class" : "NODE",
            "ref_type" : "CEC",
```

```
            "ref_index" : 0
         }
      }
   ],
   "subscriptions" : [
      {
         "type" : "I2S_AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "RS232",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : true
         },
         "status" : {
            "state" : "STREAMING"
         }
      },
      {
         "type" : "INFRARED",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : true
         },
         "status" : {
            "state" : "STREAMING"
         }
```

```
    }
],
"nodes" : [
    {
        "type" : "ANALOG_AUDIO_INPUT_OUTPUT",
        "index" : 0,
        "configuration" : {
            "direction" : "INPUT"
        },
        "status" : {},
        "inputs" : [
            {
                "name" : "main",
                "index" : 0,
                "configuration" : {
                    "source" : {
                        "value" : 2
                    }
                },
                "status" : {
                    "source" : {
                        "ref_class" : "NODE",
                        "ref_type" : "HDMI_DECODER",
                        "ref_index" : 0
                    }
                }
            }
        ]
    },
    {
        "type" : "BUTTON",
        "index" : 0,
        "configuration" : {
            "action" : {
                "value" : 0
            }
```

```
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "CEC",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "FRAME_RATE_CONVERTER",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "divide_factor" : 2
      },
      "inputs" : [
         {
            "name" : "main",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "NODE",
                  "ref_type" : "HDMI_DECODER",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "HDMI_DECODER",
      "index" : 0,
      "configuration" : {
```

```
      "hdcp_support_enable" : true,
      "hdcp_22_support_disable" : false
   },
   "status" : {
      "video" : {
         "width" : 1920,
         "height" : 1080,
         "frames_per_second" : 60,
         "color_space" : "RGB",
         "bits_per_pixel" : 8,
         "scan_mode" : "PROGRESSIVE"
      },
      "has_video_details" : true,
      "video_details" : {
         "frame_rate" : 59.94,
         "pixel_clock" : 148349771,
         "total_width" : 2200,
         "total_height" : 1125,
         "hsync_width" : 44,
         "hsync_front_porch" : 88,
         "hsync_negative" : false,
         "vsync_width" : 6,
         "vsync_front_porch" : 3,
         "vsync_negative" : false,
         "vic" : 16,
         "has_hdmi_vic" : false,
         "hdmi_vic" : 0,
         "picture_aspect" : "ASPECT_16_9",
         "has_active_format" : false,
         "active_format" : 0,
         "it_content_type" : "GRAPHICS",
         "scan_information" : "NO_DATA",
         "colorimetry" : "BT709",
         "rgb_range" : "DEFAULT",
         "ycc_range" : "LIMITED"
      },
```

```
            "has_audio_details" : true,
            "audio_details" : {
               "sampling_frequency" : 48000,
               "number_of_channels" : 2
            },
            "hdcp_protected" : false,
            "hdcp_version" : "NONE",
            "hdmi_2_0_support" : true,
            "source_stable" : true,
            "audio_downmix_support" : true
         },
         "inputs" : [
            {
               "name" : "main",
               "index" : 0,
               "configuration" : {
                  "source" : {
                     "value" : 0
                  }
               },
               "status" : {
                  "source" : {
                     "ref_class" : "NODE",
                     "ref_type" : "VIDEO_INPUT",
                     "ref_index" : 0
                  }
               }
            }
         ]
      },
      {
         "type" : "INFRARED_ENCODER",
         "index" : 0,
         "configuration" : {},
         "status" : {
            "data_length_max" : 1032
         },
```

```
      "inputs" : [
         {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "INFRARED",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "INFRARED_DECODER",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "LED",
      "index" : 0,
      "configuration" : {
         "function" : {
            "value" : 0
         }
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "NETWORK_INTERFACE",
      "index" : 0,
```

```
      "configuration" : {
         "hostname" : "PHILIPPE_FCV2_TX",
         "ip" : {
            "address" : "169.254.51.225",
            "mode" : "DHCP",
            "mask" : "255.255.0.0",
            "gateway" : "0.0.0.0"
         }
      },
      "status" : {
         "mac_address" : "d8803962e132",
         "ip" : {
            "address" : "169.254.51.225"
         }
      },
      "inputs" : []
   },
   {
      "type" : "NETWORK_SWITCH",
      "index" : 0,
      "configuration" : {
         "gigabit_port_1_enable" : true
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "SCALER",
      "index" : 0,
      "configuration" : {
         "width" : 0,
         "height" : 0
      },
      "status" : {
         "video" : {
            "width" : 0,
            "height" : 0,
```

```
            "frames_per_second" : 60,
            "color_space" : "RGB",
            "bits_per_pixel" : 8,
            "scan_mode" : "PROGRESSIVE"
         }
      },
      "inputs" : [
         {
            "name" : "main",
            "index" : 0,
            "configuration" : {
               "source" : {
                  "value" : 0
               }
            },
            "status" : {
               "source" : {
                  "ref_class" : "NODE",
                  "ref_type" : "HDMI_DECODER",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "UART",
      "index" : 0,
      "configuration" : {
         "baud_rate" : 57600,
         "data_bits" : 8,
         "stop_bits" : 1,
         "parity" : "NONE"
      },
      "status" : {},
      "inputs" : [
```

```
            {
               "name" : "network",
               "index" : 0,
               "configuration" : {},
               "status" : {
                  "source" : {
                     "ref_class" : "SUBSCRIPTION",
                     "ref_type" : "RS232",
                     "ref_index" : 0
                  }
               }
            }
         ]
      },
      {
         "type" : "USB_ICRON_CHIP_LOCAL",
         "index" : 0,
         "configuration" : {
            "extender_type" : "LOCAL",
            "program_mode" : "NONE",
            "rs232_port" : 0
         },
         "status" : {
            "chip_present" : true,
            "mac_address" : "001b1300000c"
         },
         "inputs" : []
      },
      {
         "type" : "VIDEO_COMPRESSOR",
         "index" : 0,
         "configuration" : {
            "always_compress" : false
         },
         "status" : {},
         "inputs" : []
      },
```

```
            {
               "type" : "VIDEO_INPUT",
               "index" : 0,
               "configuration" : {},
               "status" : {
                  "connector_type" : "HDMI"
               },
               "inputs" : []
            },
            {
               "type" : "VIDEO_INPUT",
               "index" : 1,
               "configuration" : {},
               "status" : {
                  "connector_type" : "DISPLAY_PORT"
               },
               "inputs" : []
            }
         ]
      }
   ],
   "error" : []
},
"error" : null
}
```

A.10 Command "get settings" for Receiver Device

The example provided below is the output for the command `get settings` called with a receiver device target argument.

Refer to section 5.5 Command get for details.

```
get d88039630546 settings
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "devices" : [
         {
```

```
"device_id" : "d88039630546",
"identity" : {
   "engine" : "PLETHORA",
   "vendor_id" : 1,
   "product_id" : 1,
   "firmware_comment" : "BlueRiver 3.5.0.0rc51 evaluation",
   "firmware_version" : "3.5.0.0",
   "firmware_rc" : "51",
   "is_receiver" : true,
   "is_transmitter" : false,
   "firmware_details" : [
      {
         "type" : "PRIMARY",
         "details_available" : true,
         "firmware_version" : "3.5.0.0",
         "firmware_rc" : "51"
      },
      {
         "type" : "BACKUP",
         "details_available" : true,
         "firmware_version" : "3.5.0.0",
         "firmware_rc" : "45"
      }
   ]
},
"configuration" : {
   "device_name" : "PHILIPPE_FCV2_RX",
   "locate_mode" : false
},
"status" : {
   "active" : true,
   "point_to_point" : false,
   "boot_status" : "PRIMARY",
   "update_in_progress" : false,
   "error_status" : {
      "has_error_code" : false,
      "error_code" : 0
```

```
    }
},
"streams" : [
    {
        "type" : "RS232",
        "index" : 0,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false,
            "auto_stop" : false
        },
        "status" : {
            "state" : "STOPPED"
        },
        "source" : {
            "ref_class" : "NODE",
            "ref_type" : "UART",
            "ref_index" : 0
        }
    },
    {
        "type" : "INFRARED",
        "index" : 0,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false,
            "auto_stop" : false
        },
        "status" : {
            "state" : "STOPPED"
        },
        "source" : {
            "ref_class" : "NODE",
            "ref_type" : "INFRARED_DECODER",
            "ref_index" : 0
        }
```

```
    },
    {
       "type" : "CEC",
       "index" : 0,
       "configuration" : {
          "address" : "0.0.0.0",
          "enable" : false,
          "auto_stop" : false
       },
       "status" : {
          "state" : "STOPPED"
       },
       "source" : {
          "ref_class" : "NODE",
          "ref_type" : "CEC",
          "ref_index" : 0
       }
    }
 ],
 "subscriptions" : [
    {
       "type" : "HDMI",
       "index" : 0,
       "configuration" : {
          "address" : "224.1.1.1",
          "enable" : true
       },
       "status" : {
          "state" : "STREAMING"
       }
    },
    {
       "type" : "HDMI",
       "index" : 1,
       "configuration" : {
          "address" : "0.0.0.0",
          "enable" : false
```

```
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 2,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 3,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 4,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
```

```
        }
    },
    {
        "type" : "HDMI",
        "index" : 5,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
        "type" : "HDMI",
        "index" : 6,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
        "type" : "HDMI",
        "index" : 7,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
        "type" : "HDMI",
```

```
      "index" : 8,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 9,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 10,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 11,
      "configuration" : {
         "address" : "0.0.0.0",
```

```
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 12,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 13,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 14,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
```

```
        }
    },
    {
        "type" : "HDMI",
        "index" : 15,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
        "type" : "HDMI",
        "index" : 16,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
        "type" : "HDMI",
        "index" : 17,
        "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
        },
        "status" : {
            "state" : "STOPPED"
        }
    },
    {
```

```
      "type" : "HDMI",
      "index" : 18,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 19,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 20,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 21,
      "configuration" : {
         "address" : "0.0.0.0",
```

```
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 22,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 23,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 24,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
```

```
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 25,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 26,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI",
         "index" : 27,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
```

```
      "type" : "HDMI",
      "index" : 28,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 29,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 30,
      "configuration" : {
         "address" : "0.0.0.0",
         "enable" : false
      },
      "status" : {
         "state" : "STOPPED"
      }
   },
   {
      "type" : "HDMI",
      "index" : 31,
      "configuration" : {
```

```
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "HDMI_AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
            "state" : "STOPPED"
         }
      },
      {
         "type" : "I2S_AUDIO",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : false
         },
         "status" : {
```

```
            "state" : "STOPPED"
         }
      },
      {
         "type" : "RS232",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : true
         },
         "status" : {
            "state" : "STREAMING"
         }
      },
      {
         "type" : "INFRARED",
         "index" : 0,
         "configuration" : {
            "address" : "0.0.0.0",
            "enable" : true
         },
         "status" : {
            "state" : "STREAMING"
         }
      }
   ],
   "nodes" : [
      {
         "type" : "ANALOG_AUDIO_OUTPUT",
         "index" : 0,
         "configuration" : {},
         "status" : {},
         "inputs" : [
            {
               "name" : "main",
               "index" : 0,
```

```
            "configuration" : {
               "source" : {
                  "value" : 3
               }
            },
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "AUDIO",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "BUTTON",
      "index" : 0,
      "configuration" : {
         "action" : {
            "value" : 0
         }
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "CEC",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "COLOR_GENERATOR",
      "index" : 0,
      "configuration" : {
```


Semtech
Proprietary & Confidential

```
            "enable" : false,
            "color" : "0000c0"
         },
         "status" : {},
         "inputs" : []
      },
      {
         "type" : "FRAME_BUFFER",
         "index" : 0,
         "configuration" : {
            "display_mode" : "FAST_SWITCHED",
            "width" : 1920,
            "height" : 1080,
            "horiz_offset" : 0,
            "vert_offset" : 0,
            "frames_per_second" : 60
         },
         "status" : {
            "source_video" : {
               "width" : 1920,
               "height" : 1080,
               "frames_per_second" : 60,
               "color_space" : "RGB",
               "bits_per_pixel" : 8,
               "scan_mode" : "PROGRESSIVE"
            }
         },
         "inputs" : [
            {
               "name" : "main",
               "index" : 0,
               "configuration" : {},
               "status" : {
                  "source" : {
                     "ref_class" : "SUBSCRIPTION",
                     "ref_type" : "HDMI",
```

```
            "ref_index" : 0
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 0
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 1,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 1
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 2,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 2
```

```
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 3,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 3
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 4,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 4
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 5,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 5
```

```
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 6,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 6
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 7,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 7
            }
        }
    },
    {
        "name" : "multiview",
        "index" : 8,
        "configuration" : {},
        "status" : {
            "source" : {
                "ref_class" : "SUBSCRIPTION",
                "ref_type" : "HDMI",
                "ref_index" : 8
            }
```

```
    }
},
{
    "name" : "multiview",
    "index" : 9,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 9
        }
    }
},
{
    "name" : "multiview",
    "index" : 10,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 10
        }
    }
},
{
    "name" : "multiview",
    "index" : 11,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 11
        }
```

```
    }
},
{
    "name" : "multiview",
    "index" : 12,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 12
        }
    }
},
{
    "name" : "multiview",
    "index" : 13,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 13
        }
    }
},
{
    "name" : "multiview",
    "index" : 14,
    "configuration" : {},
    "status" : {
        "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 14
        }
    }
```

```
},
{
   "name" : "multiview",
   "index" : 15,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 15
      }
   }
},
{
   "name" : "multiview",
   "index" : 16,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 16
      }
   }
},
{
   "name" : "multiview",
   "index" : 17,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 17
      }
   }
```

```
},
{
   "name" : "multiview",
   "index" : 18,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 18
      }
   }
},
{
   "name" : "multiview",
   "index" : 19,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 19
      }
   }
},
{
   "name" : "multiview",
   "index" : 20,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 20
      }
   }
},
```

```
{
   "name" : "multiview",
   "index" : 21,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 21
      }
   }
},
{
   "name" : "multiview",
   "index" : 22,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 22
      }
   }
},
{
   "name" : "multiview",
   "index" : 23,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 23
      }
   }
},
```

```
{
   "name" : "multiview",
   "index" : 24,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 24
      }
   }
},
{
   "name" : "multiview",
   "index" : 25,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 25
      }
   }
},
{
   "name" : "multiview",
   "index" : 26,
   "configuration" : {},
   "status" : {
      "source" : {
         "ref_class" : "SUBSCRIPTION",
         "ref_type" : "HDMI",
         "ref_index" : 26
      }
   }
},
{
```

```
      "name" : "multiview",
      "index" : 27,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 27
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 28,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 28
         }
      }
   },
   {
      "name" : "multiview",
      "index" : 29,
      "configuration" : {},
      "status" : {
         "source" : {
            "ref_class" : "SUBSCRIPTION",
            "ref_type" : "HDMI",
            "ref_index" : 29
         }
      }
   },
   {
```

```
            "name" : "multiview",
            "index" : 30,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "HDMI",
                  "ref_index" : 30
               }
            }
         },
         {
            "name" : "multiview",
            "index" : 31,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "HDMI",
                  "ref_index" : 31
               }
            }
         }
      ]
   },
   {
      "type" : "HDMI_ENCODER",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "video" : {
            "width" : 1920,
            "height" : 1080,
            "frames_per_second" : 60,
            "color_space" : "RGB",
            "bits_per_pixel" : 8,
            "scan_mode" : "PROGRESSIVE"
```


Semtech
Proprietary & Confidential

```
    },
    "has_video_details" : true,
    "video_details" : {
       "frame_rate" : 59.94,
       "pixel_clock" : 148351646,
       "total_width" : 2200,
       "total_height" : 1125,
       "hsync_width" : 44,
       "hsync_front_porch" : 88,
       "hsync_negative" : false,
       "vsync_width" : 5,
       "vsync_front_porch" : 4,
       "vsync_negative" : false,
       "vic" : 16,
       "has_hdmi_vic" : false,
       "hdmi_vic" : 0,
       "picture_aspect" : "ASPECT_16_9",
       "has_active_format" : false,
       "active_format" : 0,
       "it_content_type" : "NO_DATA",
       "scan_information" : "NO_DATA",
       "colorimetry" : "RGB",
       "rgb_range" : "DEFAULT",
       "ycc_range" : "LIMITED"
    },
    "has_audio_details" : false,
    "audio_details" : {},
    "hdcp_protected" : false,
    "hdcp_version" : "NONE",
    "hdmi_2_0_support" : true,
    "source_stable" : true
},
"inputs" : [
    {
       "name" : "main",
       "index" : 0,
```

```
            "configuration" : {
               "source" : {
                  "value" : 1
               }
            },
            "status" : {
               "source" : {
                  "ref_class" : "NODE",
                  "ref_type" : "FRAME_BUFFER",
                  "ref_index" : 0
               }
            }
         },
         {
            "name" : "audio",
            "index" : 0,
            "configuration" : {
               "source" : {
                  "value" : 2
               }
            },
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "HDMI_AUDIO",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "HDMI_MONITOR",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "connected" : true
```

```
      },
      "inputs" : [
         {
            "name" : "main",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "NODE",
                  "ref_type" : "HDMI_ENCODER",
                  "ref_index" : 0
               }
            }
         }
      ]
   },
   {
      "type" : "INFRARED_ENCODER",
      "index" : 0,
      "configuration" : {},
      "status" : {
         "data_length_max" : 1032
      },
      "inputs" : [
         {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
               "source" : {
                  "ref_class" : "SUBSCRIPTION",
                  "ref_type" : "INFRARED",
                  "ref_index" : 0
               }
            }
         }
```

```
      ]
   },
   {
      "type" : "INFRARED_DECODER",
      "index" : 0,
      "configuration" : {},
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "LED",
      "index" : 0,
      "configuration" : {
         "function" : {
            "value" : 0
         }
      },
      "status" : {},
      "inputs" : []
   },
   {
      "type" : "NETWORK_INTERFACE",
      "index" : 0,
      "configuration" : {
         "hostname" : "PHILIPPE_FCV2_RX",
         "ip" : {
            "address" : "169.254.71.5",
            "mode" : "DHCP",
            "mask" : "255.255.0.0",
            "gateway" : "0.0.0.0"
         }
      },
      "status" : {
         "mac_address" : "d88039630546",
         "ip" : {
            "address" : "169.254.71.5"
         }
```

```
    },
    "inputs" : []
},
{
    "type" : "NETWORK_SWITCH",
    "index" : 0,
    "configuration" : {
        "gigabit_port_1_enable" : true
    },
    "status" : {},
    "inputs" : []
},
{
    "type" : "UART",
    "index" : 0,
    "configuration" : {
        "baud_rate" : 57600,
        "data_bits" : 8,
        "stop_bits" : 1,
        "parity" : "NONE"
    },
    "status" : {},
    "inputs" : [
        {
            "name" : "network",
            "index" : 0,
            "configuration" : {},
            "status" : {
                "source" : {
                    "ref_class" : "SUBSCRIPTION",
                    "ref_type" : "RS232",
                    "ref_index" : 0
                }
            }
        }
    ]
```

```
            },
            {
               "type" : "USB_ICRON_CHIP_REMOTE",
               "index" : 0,
               "configuration" : {
                  "extender_type" : "REMOTE",
                  "program_mode" : "NONE",
                  "rs232_port" : 0
               },
               "status" : {
                  "chip_present" : true,
                  "mac_address" : "001b13000111"
               },
               "inputs" : []
            }
         ]
      }
   ],
   "error" : []
},
"error" : null
}
```


Appendix B Command Output Examples (Other than Get)

B.1Command "layout describe"

The following is an example output for the command **layout describe**.

Refer to section 6.4.3 layout describe for details.


```
layout compatibility_4k_2x2 describe
{
   "status" : "SUCCESS",
   "request_id" : null,
   "result" : {
      "layout_description" : {
         "name" : "compatibility_4k_2x2",
         "width" : 3840,
         "height" : 2160,
         "read_only" : true,
         "surfaces" : [
            {
               "index" : 0,
               "horizontal_position" : 0,
```

```
      "vertical_position" : 0,
      "width" : 4096,
      "height" : 2160
   },
   {
      "index" : 1,
      "horizontal_position" : 4160,
      "vertical_position" : 0,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 2,
      "horizontal_position" : 6240,
      "vertical_position" : 0,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 3,
      "horizontal_position" : 8320,
      "vertical_position" : 0,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 4,
      "horizontal_position" : 4160,
      "vertical_position" : 1080,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 5,
      "horizontal_position" : 6240,
      "vertical_position" : 1080,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 6,
      "horizontal_position" : 8320,
      "vertical_position" : 1080,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 7,
      "horizontal_position" : 0,
      "vertical_position" : 2160,
      "width" : 2048,
      "height" : 1080
   },
   {
      "index" : 8,
```

```
            "horizontal_position" : 2080,
            "vertical_position" : 2160,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 9,
            "horizontal_position" : 4160,
            "vertical_position" : 2160,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 10,
            "horizontal_position" : 6240,
            "vertical_position" : 2160,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 11,
            "horizontal_position" : 8320,
            "vertical_position" : 2160,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 12,
            "horizontal_position" : 0,
            "vertical_position" : 3240,
            "width" : 2048,
            "height" : 856
        },
        {
            "index" : 13,
            "horizontal_position" : 2080,
            "vertical_position" : 3240,
            "width" : 2048,
            "height" : 856
        },
        {
            "index" : 14,
            "horizontal_position" : 4160,
            "vertical_position" : 3240,
            "width" : 2048,
            "height" : 856
        },
        {
            "index" : 15,
            "horizontal_position" : 6240,
            "vertical_position" : 3240,
            "width" : 2048,
            "height" : 856
        },
        {
            "index" : 16,
```

```
            "horizontal_position" : 8320,
            "vertical_position" : 3240,
            "width" : 2048,
            "height" : 856
        },
        {
            "index" : 17,
            "horizontal_position" : 0,
            "vertical_position" : 0,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 18,
            "horizontal_position" : 2080,
            "vertical_position" : 0,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 19,
            "horizontal_position" : 0,
            "vertical_position" : 1080,
            "width" : 2048,
            "height" : 1080
        },
        {
            "index" : 20,
            "horizontal_position" : 2080,
            "vertical_position" : 1080,
            "width" : 2048,
            "height" : 1080
        }
    ],
    "windows" : [
        {
            "index" : 0,
            "horizontal_position" : 0,
            "vertical_position" : 0,
            "width" : 1920,
            "height" : 1080,
            "horizontal_offset" : 0,
            "vertical_offset" : 0,
            "content" : "VIDEO",
            "target_surface" : 0
        },
        {
            "index" : 1,
            "horizontal_position" : 1920,
            "vertical_position" : 0,
            "width" : 1920,
            "height" : 1080,
            "horizontal_offset" : 0,
            "vertical_offset" : 0,
            "content" : "VIDEO",
```

```
            "target_surface" : 1
         },
         {
            "index" : 2,
            "horizontal_position" : 0,
            "vertical_position" : 1080,
            "width" : 1920,
            "height" : 1080,
            "horizontal_offset" : 0,
            "vertical_offset" : 0,
            "content" : "VIDEO",
            "target_surface" : 2
         },
         {
            "index" : 3,
            "horizontal_position" : 1920,
            "vertical_position" : 1080,
            "width" : 1920,
            "height" : 1080,
            "horizontal_offset" : 0,
            "vertical_offset" : 0,
            "content" : "VIDEO",
            "target_surface" : 3
         }
      ]
   }
},
"error" : null
}
```

IMPORTANTNOTICE

Information relating to this product and the application or design described herein is believed to be reliable, however such information is provided as a guide only and Semtech assumes no liability for any errors in this document, or for the application or design described herein. Semtech reserves the right to make changes to the product or this document at any time without notice. Buyers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. Semtech warrants performance of its products to the specifications applicable at the time of sale, and all sales are made in accordance with Semtech's standard terms and conditions of sale.

SEMTECH PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS, OR IN NUCLEAR APPLICATIONS IN WHICH THE FAILURE COULD BE REASONABLY EXPECTED TO RESULT IN PERSONAL INJURY, LOSS OF LIFE OR SEVERE PROPERTY OR ENVIRONMENTAL DAMAGE. INCLUSION OF SEMTECH PRODUCTS IN SUCH APPLICATIONS IS UNDERSTOOD TO BE UNDERTAKEN SOLELY AT THE CUSTOMER'S OWN RISK. Should a customer purchase or use Semtech products for any such unauthorized application, the customer shall indemnify and hold Semtech and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs damages and attorney fees which could arise.

The Semtech name and logo are registered trademarks of the Semtech Corporation. All other trademarks and trade names mentioned may be marks and names of Semtech or their respective companies. Semtech reserves the right to make changes to, or discontinue any products described in this document without further notice. Semtech makes no warranty, representation or guarantee, express or implied, regarding the suitability of its products for any particular purpose. All rights reserved.

© Semtech 2018

Contact Information

Semtech Quebec Inc.
2344 Alfred Nobel Blvd. Suite 102, St. Laurent, QC, Canada H4S 0A4
Tel: 514-446-2400 • www.semtech.com