Command Guide

aegis

CONNECTION TOGETHER

4KPROIP-CBS

4K AV Over IP Solution Control Box Software

Encoder

Decoder

Control Box Software

Table of Contents

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

Table of Contents continued…

1 Introduction

This document describes everything that a developer needs to be aware of to use the iMAGsystems LIGHTNING command guide and develop client control applications for LIGHTNING SDVoE devices.

1.1 Licence Requirements

The SDVoE Director Controller must have a valid licence key entered before use or trying to connect to the TCP control port 6980.
If no valid licence is active the SDVoE Director Controller will return 'Invalid License' and terminate the TCP connection. Contact iMAGsystems or your local distributor for licencing information.

1.2 Telnet Connection

Third party controllers connect to the SDVoE Director Controller and issue commands using ascii strings terminated with a carriage return. This allows any Telnet client to be used with the system.

The SDVoE Director Controller listens on TCP port 6980. Once a successful TCP connection is established you will receive a welcome message 'Connection Successful'.

A constant TCP connection to the SDVoE Director Controller is recommended to maintain status changes of the system from notification events.

An optional security key can be used with all TCP API commands made to port 6980. The keyword 'key:' along with the security key are added to the API command before any parameters.

1.3 HTTP requests

It is also possible to control the SDVoE Director Controller with HTTP GET and POST requests.
A security key must be sent with any request. Security keys are generated by 'admin' level UI users on the Global Settings / Security Key tab.

GET = http://*<controllerURL>*/api/command/*<LIGHTNING_API_COMMAND>*/*<KEY>*
POST = http://<controllerURL>/api/command/{'cmd':'*<LIGHTNING_API_COMMAND>*','key':'*<KEY>*'}

Refer Appendix D - How to HTTP request

2 Command Overview

Commands are in a simple ascii text format. For each command, the SDVoE Director Controller responds with a response which contains the return status (i.e. whether the command succeeded or not) and, if successful, the return value of the command if required.

- All commands and returns are terminated with a carriage return <cr> 0x0D
- Commands are not case sensitive
- Invalid commands will return **error [Unknown]**<cr>
- Missing security key will return **error [security key missing]**<cr>

ToC

3 System Commands

3.1 Command default

The **default** command is used to set the default video resolution and frame rate used for join fast and wall modes. By default this setting is 1920 1080 60. During fast and wall joins the controller may request the current input resolution and frame rate on the incoming video signal if not specified and use this information to control the switching of streams. If no video is present and no resolution specified with the command then this **default** value will be used.

When the **join** parameter **lock** is present this default value is always used.

3.1.1 Command usage

default **[**key:<*security_key*>**]** <*width*> <*height*> <*fps*><cr>

3.1.2 Description

This command can be set once or as many times as required.

3.1.3 Arguments

width	Horizontal video size
height	Vertical video size
fps	Frames per Second

3.1.4 Notes

3.1.5 Return Value

command	<space>	**status**	**terminator**
default	<space>	*success / error [message]*	<cr>

3.1.6 Command Examples

```
default 1920 1080 60<cr>
default 3840 2160 30<cr>
default 3840 2160 60<cr>
default key:abc123 3840 2160 60<cr>
```

3.1.7 Return Examples

```
default success<cr>

default error [incomplete]<cr>
default error [invalid input value -X, non-negative number expected]<cr>
```

ToC

3.2 Command reboot

The **reboot** command is used to restart any or all Encoders and Decoders.

3.2.1 Command usage

reboot **[**key:<*security_key*>**]** <*device_name*><cr>

3.2.2 Description

Causes the target device(s) to restart. The target device(s) becomes unavailable for a short period while restarting.

3.2.3 Arguments

device_name	Name of the Encoder, Decoder, Group or '**all**', '**all_rx**', '**all_tx**'

3.2.4 Notes

- **all** is used as a destination when all devices are required to reboot.
- **all_rx** is used as a destination when all Decoders are required to reboot.
- **all_tx** is used as a destination when all Encoders are required to reboot.
- ***group_name*** is used as a destination when all Encoders and Decoders in a group are required to reboot.

3.2.5 Return Value

command	<space>	**status**	**terminator**
reboot	<space>	*success / error [message]*	<cr>

3.2.6 Command Examples

```
reboot Encoder1<cr>
reboot all<cr>
reboot all_rx<cr>
reboot all_tx<cr>
reboot MyGroup<cr>
reboot key:abc123 Encoder1<cr>
```

3.2.7 Return Examples

```
reboot success<cr>

reboot error [incomplete]<cr>
reboot error [device 'Encoder1' not found]<cr>
```

ToC

4 Command join

The **join** commands are used for routing all the signals to their required destinations. Video, digital audio, analog audio, USB, infrared and serial can all be independently routed to their destinations.

join fast command provides the fastest switching possible between video streams. This is achieved by maintaining a constant resolution and frame rate to the display at all times. This does however add 1-2 frames of latency as the video is buffered in the Decoder.

In **fast** 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 (pillarbox) or at the top and bottom (letterbox) of the image to preserve the aspect ratio. An alternate behaviour can be specified by adding the keyword **crop** or **stretch** to the command line.

- 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 regardless of the aspect ratio.

join sync command is used to maintain source resolution and frame rate at the display. This mode also provides the lowest latency possible of just a few lines, yes *lines* not frames.

join sync_scale command is used to scale display resolution while maintaining the source format. This mode also provides the lowest latency possible of just a few lines, *yes lines not frames*.

join adv command provides a combined method of joining in either fast, sync or sync_scale modes.

join audio_a command provides independent routing of the analog audio.

join audio_d command provides independent routing of the HDMI digital audio.

join infrared command provides independent routing of infrared (IR).

join serial command provides independent routing of serial RS-232.

join usb command provides independent routing of USB.

join wall commands are used to send a cropped portion of the video source to a display in a video wall configuration.

join multi command is used to rout a video source to a portion of the display in multiview mode.

ToC

4.1 Command join fast

4.1.1 Command usage

join fast **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*>
[<av>**]** **[**<lock>**]** **[**<exclusive>**]** **[**<auto>**]** **[**<*aspect*>**]** **[**size <*width*> <*height*> <*fps*>**]**<cr>

4.1.2 Description

The command **join fast** is used for fast switching of video and embedded audio signals whereby the display video timing is kept as a constant and resyncing with the display is not required. The Decoder will maintain a constant scaled output resolution. When the **lock** parameter is used then the resolution defined with the command **default** is set. *Refer to 3.1 'Command default'.* When **lock** is not used, then the optional **size width**, **height** and **fps** can be used. If these are not specified then the Encoder's present video resolution is used. If no video is available, then the resolution defined by the **default** command is set. When **auto** is used, the displays EDID is analysed to obtain the highest preferred resolution and is set automatically as the join resolution.

4.1.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the *Decoder*, *Group* or '**all**'
av	Keyword '**av**' will also join the digital audio stream (optional)
lock	Keyword '**lock**' sets the default output resolution (optional)
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)
auto	Keyword '**auto**' will create the join to the maximum preferred displays resolution. (optional)
aspect	Defines the aspect ratio with keyword '**keep**' / '**crop**' / '**stretch**' (optional)
size *width height fps*	Defines the scaled video width height and frame rate (optional)

4.1.4 Notes

- **all** can be used as a destination when the Encoder is required to be joined to all Decoders.
- ***group_name*** is used as a destination when the Encoder is required to be joined to all Decoders in a group.
- **auto** is used in place of **lock** or **size**
- **all auto** will join the Encoder at the highest common resolution detected on the connected displays.
- **aspect** by default is keep, crop and stretch can be specified when the source aspect ratio does not match the display aspect ratio. 'size' is required for aspect crop and stretch.

4.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	fast	<space>	*success / error [message]*	<cr>

ToC

4.1.6 Command Examples

```
join fast Encoder1 Decoder1<cr>
join fast Encoder1 Decoder1 av<cr>
join fast Encoder1 Decoder1 av lock exclusive<cr>
join fast Encoder1 Decoder1 size 3840 2160 60<cr>
join fast Encoder1 all<cr>
join fast Encoder1 all auto<cr>
join fast Encoder1 all size 3840 2160 60<cr>
join fast Encoder1 MyGroup<cr>
join fast Encoder1 Decoder1 auto<cr>
join fast Encoder1 Decoder1 av auto<cr>
join fast Encoder1 Decoder1 keep av auto<cr>
join fast Encoder1 Decoder1 keep size 3840 2160 60<cr>
join fast Encoder1 Decoder1 crop size 3840 2160 60<cr>
join fast Encoder1 Decoder1 stretch size 3840 2160 60<cr>
join fast key:abc123 Encoder1 Decoder1<cr>
```

4.1.7 Return Examples

```
join fast success<cr>

join fast error [incomplete]<cr>
join fast error [encoder 'Encoder1' not found]<cr>
join fast error [decoder 'Decoder1' not found]<cr>
join fast error [invalid input value -X, non-negative number expected]<cr>
join fast error [unsupported monitor]<cr>
join fast error [monitor not detected]<cr>
```

ToC

4.2 Command join sync

4.2.1 Command usage

join sync **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> **[**<av>**]** **[**<exclusive>**]**<cr>

4.2.2 Description

The command **join sync** is used whenever the display device is required to maintain the same video format as the incoming video with minimum latency. The original source frame rate is always maintained.

4.2.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
av	Keyword '**av**' will also join the digital audio stream (optional)
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)

4.2.4 Notes

- **all** can be used as a destination when the Encoder is required to be connected to all Decoders.
- ***group_name*** is used as a destination when the Encoder is required to be joined to all Decoders in a group.
- This mode can cause a delay in video switching time as the display device must sync to the current video signal. This delay time can vary depending on the display device used.

4.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	sync	<space>	*success / error [message]*	<cr>

4.2.6 Command Examples

```
join sync Encoder1 Decoder1<cr>
join sync Encoder1 Decoder1 av<cr>
join sync Encoder1 all<cr>
join sync Encoder1 MyGroup<cr>
join sync key:abc123 Encoder1 Decoder1<cr>
```

4.2.7 Return Examples

```
join sync success<cr>

join sync error [incomplete]<cr>
join sync error [encoder 'Encoder1' not found]<cr>
join sync error [decoder 'Decoder1' not found]<cr>
join sync error [invalid input value -X, non-negative number expected]<cr>
```

ToC

4.3 Command join sync_scale

4.3.1 Command usage

join sync_scale **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*>
[<av>**]** **[**<lock>**]** **[**<exclusive>**]** **[**<auto>**]** **[**size <*width*> <*height*>**]**<cr>

4.3.2 Description

The command **join sync_scale** is used whenever the display device is required to have the same video format as the incoming video and allows scaling of the output resolution. This mode has the minimum latency possible. The Decoder will maintain a constant scaled output resolution. When the **locked** parameter is used then the resolution defined with the command **default** is set. *Refer to 3.1 'Command default'.* When **locked** is not used, then the optional **size width**, **height** can be used. If these are not specified then the Encoder's present video resolution is set. If no video is available, then the resolution defined by the default command is set. When **auto** is used, the displays EDID is analysed to obtain the highest preferred resolution and is set automatically as the join resolution. The original source frame rate is always maintained.

4.3.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
size *width height*	Defines the scaled video resolution width height
av	Keyword '**av**' will also join the digital audio stream (optional)
lock	Keyword '**lock**' sets the default output resolution (optional)
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)
auto	Keyword '**auto**' will create the join to the maximum preferred displays resolution. (optional)

4.3.4 Notes

- **all** can be used as a destination when the Encoder is required to be joined to all Decoders.
- ***group_name*** is used as a destination when the Encoder is required to be joined to all Decoders in a group.
- This mode can cause a delay in video switching time as the display device must sync to the current video signal. This delay time can vary depending on the display device used.
- **auto** is used in place of **lock** or **size**
- **all auto** will join the Encoder at the highest common resolution detected on the connected displays.

ToC

4.3.5 Return Value

command	<space>	mode	<space>	status	terminator
join	<space>	sync_scale	<space>	*success / error [message]*	<cr>

4.3.6 Command Examples

```
join sync_scale Encoder1 Decoder1 size 1920 1080<cr>
join sync_scale Encoder1 Decoder1 size 1920 1080 av<cr>
join sync_scale Encoder1 Decoder1 size 3840 2160 av exclusive<cr>
join sync_scale Encoder1 all size 1920 1080<cr>
join sync_scale Encoder1 all auto<cr>
join sync_scale Encoder1 MyGroup size 1920 1080<cr>
join sync_scale Encoder1 Decoder1 auto<cr>
join sync_scale key:abc123 Encoder1 Decoder1 size 1920 1080<cr>
```

4.3.7 Return Examples

```
join sync_scale success<cr>

join sync_scale error [incomplete]<cr>
join sync_scale error [encoder 'Encoder1' not found]<cr>
join sync_scale error [decoder 'Decoder1' not found]<cr>
join sync_scale error [invalid input value -X, non-negative number expected]<cr>
join sync_scale error [unsupported monitor]<cr>
join sync_scale error [monitor not detected]<cr>
```

ToC

4.4 Command join adv

4.4.1 Command usage

join adv **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*>
[<sync>**]** **[**size <*width*> <*height*>**]** **[**<fast>**]** **[**<*aspect*>**]** **[**size <*width*> <*height*> <*fps*>**]** **[**<av>**]** **[**<exclusive>**]**
[<auto>**]**<cr>

4.4.2 Description

The command **join adv** is used for switching of video and embedded audio signals. This mode of join is used so the Decoder only changes modes or resolution when required and not nessesarily with each join command.

4.4.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
fast	Keyword '**fast**' sets fast mode (seamsless switching with 1-2 frames of latency)
sync	Keyword '**sync**' sets sync mode (display remains in sync with the source and provides the lowest possible latency)
aspect (with 'fast' only)	Defines the aspect ratio with keyword '**keep**' / '**crop**' / '**stretch**' (optional)
size	Keyword '**size**' followed by width, height and fps for fast mode will set the Decoder's scaler resolution to match. (optional)
av	Keyword '**av**' will also join the digital audio stream (optional)
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)
auto	Keyword '**auto**' will create the join to the maximum preferred displays resolution. (optional)

4.4.4 Notes

- **all** can be used as a destination when the Encoder is required to be joined to all Decoders.
- ***group_name*** is used as a destination when the Encoder is required to be joined to all Decoders in a group.
- **auto** is used to switch at the connected displays preferred resolution.
- **all auto** will join the Encoder at the highest common resolution detected on the connected displays.
- **aspect** in fast mode by default will be keep, crop and stretch can be specified when the source aspect ratio does not match the display aspect ratio. '**size**' is required for aspect crop and stretch.

4.4.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	adv	<space>	*success / error [message]*	<cr>

ToC

4.4.6 Command Examples

```
join adv Encoder1 Decoder1 <cr>
join adv Encoder1 Decoder1 exclusive<cr>
join adv Encoder1 Decoder1 av<cr>
join adv Encoder1 Decoder1 av exclusive<cr>
join adv Encoder1 all<cr>
join adv Encoder1 Decoder1 sync<cr>
join adv Encoder1 Decoder1 size 1920 1080<cr>
join adv Encoder1 Decoder1 auto<cr>
join adv Encoder1 Decoder1 fast<cr>
join adv Encoder1 Decoder1 fast size 1920 1080 60<cr>
join adv Encoder1 Decoder1 fast keep<cr>
join adv Encoder1 Decoder1 fast keep size 1920 1080 60<cr>
join adv Encoder1 Decoder1 fast crop size 1920 1080 60<cr>
join adv Encoder1 Decoder1 fast stretch size 1920 1080 60<cr>
join adv Encoder1 Decoder1 fast auto<cr>
join adv Encoder1 Decoder1 fast auto av excusive<cr>
```

4.4.7 Return Examples

```
join adv success<cr>

join adv error [incomplete]<cr>
join adv error [encoder 'Encoder1' not found]<cr>
join adv error [decoder 'Decoder1' not found]<cr>
join adv error [invalid input value -X, non-negative number expected]<cr>
```

ToC

4.5 Command join audio_a

4.5.1 Command usage

join audio_a **[**key:<*security_key*>**]** <*encoder_device_name*> <decoder_device_name> **[**<exclusive>**]**<cr>

4.5.2 Description

The command **join audio_a** is used for routing analog audio.
Join one or more Decoders to an Encoder's analog audio stream.

4.5.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)

4.5.4 Notes

- **all** can be used as a destination when the analog audio is required to be connected to all Decoders.
- ***group_name*** can be used as a destination when the analog audio is required to be connected to all Decoders in a group.
- Analog audio is not routed with any other command. **join audio_a** is the only method of routing analog audio.

4.5.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	audio_a	<space>	*success / error [message]*	<cr>

4.5.6 Command Examples

```
join audio_a Encoder1 Decoder1<cr>
join audio_a Encoder1 Decoder1 exclusive<cr>
join audio_a Encoder1 all<cr>
join audio_a Encoder1 all<cr>
join audio_a Encoder1 MyGroup<cr>
join audio_a key:abc123 Encoder1 Decoder1<cr>
```

4.5.7 Return Examples

```
join audio_a success<cr>

join audio_a error [incomplete]<cr>
join audio_a error [encoder 'Encoder1' not found]<cr>
join audio_a error [decoder 'Decoder1' not found]<cr>
join audio_a error [not supported]<cr>
```

ToC

4.6 Command join audio_d

4.6.1 Command usage

join audio_d **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> **[**<exclusive>**]**<cr>

4.6.2 Description

The command **join audio_d** is used for routing digital audio independently of the video.

Join one or more Decoders to an Encoder's digital audio stream.

4.6.3 Arguments

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
exclusive	Keyword '**exclusive**' allows for only the specified Decoder to be joined with the Encoder. All other Decoders joined with the Encoder will be removed. (optional)

4.6.4 Notes

- **all** can be used as a destination when the embedded audio is required to be connected to all Decoders.
- ***group_name*** can be used as a destination when the embedded audio is required to be connected to all Decoders in a group.

4.6.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	audio_d	<space>	*success / error [message]*	<cr>

4.6.6 Examples

```
join audio_d Encoder1 Decoder1<cr>
join audio_d Encoder1 Decoder1 exclusive<cr>
join audio_d Encoder1 all<cr>
join audio_d Encoder1 all<cr>
join audio_d Encoder1 MyGroup<cr>
join audio_d key:abc123 Encoder1 Decoder1<cr>
```

4.6.7 Return Examples

```
join audio_d success<cr>

join audio_d error [incomplete]<cr>
join audio_d error [encoder 'Encoder1' not found]<cr>
join audio_d error [decoder 'Decoder1' not found]<cr>
```

ToC

4.7 Command join ir

4.7.1 Command usage

join ir **[**key:<*security_key*>**]** <*source*> <*destination*><cr>

4.7.2 Description

The command **join ir** is used for routing infrared IR signals.
Join one or more Decoders to an Encoder's infrared stream.

4.7.3 Arguments

source	Device name of the source
destination	Name of the destination device or '**all**' or '**all_rx**' or '**all_tx**'

4.7.4 Notes

- **all** can be used as a destination when the infrared is required to be routed to all other devices.

4.7.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	ir	<space>	*success / error [message]*	<cr>

4.7.6 Command Examples

```
join ir Encoder1 Decoder1<cr>
join ir Decoder1 Decoder2<cr>
join ir Encoder1 all<cr>
join ir key:abc123 Encoder1 Decoder1<cr>
```

4.7.7 Return Examples

```
join ir success<cr>

join ir error [incomplete]<cr>
join ir error [source 'Encoder1' not found]<cr>
join ir error [destination 'Decoder1' not found]<cr>
join ir error [not supported]<cr>
```

4.8 Command join serial

4.8.1 Command usage

join serial **[**key:<*security_key*>**]** <*source*> <*destination*> **[**<bi>**]** **[**<exclusive>**]**<cr>

4.8.2 Description

The command **join serial** is used for routing serial RS-232 signals. Join a devices serial port to another devices serial port or the API.

4.8.3 Arguments

source	Device name of the source
destination	Name of the destination device or '**api**' or '**all**'
bi	Keyword '**bi**' for bi-directional communication, creates a 2-way connection (optional)
exclusive	Keyword '**exclusive**' will remove all other joins (optional)

4.8.4 Notes

- **api** must be used as a destination when 2-way communication is required with the control system.
- **all** can be used as a destination when the serial is required to be routed to all other devices.
- **bi** parameter is used to activate a 2-way serial point-to-point connection.
- **exclusive** parameter when true will stop any other sender connected to the destination receiver.
- When a serial reply is required, then the Encoder or Decoder's serial port must be connected to the controller. *Refer to 9.1 'Command send serial'.*

4.8.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	serial	<space>	*success / error [message]*	<cr>

4.8.6 Command Examples

```
join serial Encoder1 Decoder1 bi<cr>
join serial Decoder1 Decoder2 exclusive<cr>
join serial Encoder1 all<cr>
join serial Encoder1 api<cr>
join serial key:abc123 Encoder1 Decoder1<cr>
```

4.8.7 Return Examples

```
join serial success<cr>

join serial error [incomplete]<cr>
join serial error [source 'Encoder1' not found]<cr>
join serial error [destination 'Decoder1' not found]<cr>
join serial error [invalid input value -X, non-negative number expected]<cr>
join serial error [not supported]<cr>
```

ToC

4.9 Command join usb

4.9.1 Command usage

join usb **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*><cr>

4.9.2 Description

The command **join usb** is used for routing USB from an Encoder to a Decoder.

4.9.3 Arguments

encoder_device_name	Device name of the source Encoder
decoder_device_name	Device name of the destination Decoder

4.9.4 Notes

- Intended for point-to-point KVM (keyboard, video and mouse) routing.

4.9.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	usb	<space>	*success / error [message]*	<cr>

4.9.6 Command Examples

```
join usb Encoder1 Decoder1<cr>
join usb key:abc123 Encoder1 Decoder1<cr>
```

4.9.7 Return Examples

```
join usb success<cr>

join usb error [incomplete]<cr>
join usb error [encoder 'Encoder1' not found]<cr>
join usb error [decoder 'Decoder1' not found]<cr>
join usb error [not supported]<cr>
```

4.10 Command join multi

4.10.1 Command usage

join multi **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> <*subscription*>
[scaled **[**<*layout_name*>**]]**<cr>

4.10.2 Description

The command **join multi** is used for routing video in a multiview configuration. The source video for any given multiview window. Multiview windows are defined with the **layout window** command. Each window has a subscription to a particular video source, so one or many windows can display the same video content.

4.10.3 Arguments

encoder_device_name	Device name of the source Encoder
decoder_device_name	Device name of the destination Decoder
subscription	Defines the source for multiview window(s) defined with the same subscription
scaled	Keyword '**scaled**' is used to define the use of the scaled sub stream (optional)
layout_name	When keyword '**scaled**' is present, the **layout_name** is used to find the resolution

4.10.4 Notes

- The **join multi** command will automatically start the required stream and can apply the scaled resolution.
- If **scaled** keyword is present the sub stream will be used and the scaler automatically set if a **layout_name** has been specified. Otherwise if no **layout_name** has been specified the **set scaler** command must be used to set the correct resolution of the video for the specific window size.
- The Multiview command will aslways use the substream if the source video is not PROGRESSIVE or 8-bit.
- Cannot be used on subscriptions greater than 0 unless the **multiview** command has been called.
- The frame rate of the sub stream will be automatically limited to 30Hz.

4.10.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	multi	<space>	*success / error [message]*	<cr>

4.10.6 Command Examples

```
join multi Encoder1 Decoder1 0 scaled MyLayout<cr>
join multi Encoder2 Decoder1 31<cr>
join multi Encoder1 Decoder1 0 scaled<cr>
join multi key:abc123 Encoder2 Decoder1 31<cr>
```

ToC

4.10.7 Return Examples

```
join multi success<cr>

join multi error [incomplete]<cr>
join multi error [encoder 'Encoder1' not found]<cr>
join multi error [decoder 'Decoder1' not found]<cr>
join multi error [layout 'MyLayout' not found]<cr>
join multi error [invalid subscription '32']<cr>
join multi error [subscription '3' not found]<cr>
join multi error [Only HDMI subscription index 0 can be joined in display modes other than MULTIVIEW]
```

ToC

4.11 Command join wall

Command join wall is separated into two sub modes, **wall** and **walladv**. **wall** mode is available for easy configuration and automatically compensating for bezel correction when the keyword **size** along with **display_width** and **viewable_width** are specified. Use the **walladv** mode for advanced features like maintaining aspect ratio. A video wall can contain a maximum of 8x5 which is 8 columns and 5 rows of displays.

4.11.1 Command join wall

4.11.1.1 Command usage

join wall **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> <*wall_type*> <*display_position*> **[**size <*width*> <*height*> <*fps*>**]**
[bezel <*display_width*> <*viewable_width*>**]** **[**<keep>**]** **[**<*wall_mode*>**]**<cr>

4.11.1.2 Description

The command **join wall** is used for routing video in a video wall configuration. The size of the video portion along with the displays physical size are both used to automatically calculate the bezel compensation required. If **display_width** or **viewable_width** equal 0 then no bezel compensation will be applied. (A constant bezel size is assumed around the display.) The **size** parameter is used to set the resolution of the displays.
If the displays bezel is not constant then use walladv. *Refer appendix B – How to Video wall.*

4.11.1.3 Arguments

encoder_device_name	Device name of the source Encoder
decoder_device_name	Device name of the destination Decoder
wall_type	Specifies the type of video wall configuration. See below
display_position	Specifies the position of the display within the video wall. See below
size	Keyword '**size**' used to specify width, height and frame rate of display (optional)
width	Output width resolution in pixels (eg 3840 / 1920 / 1280) (used with size)
height	Output height resolution in pixels (eg. 2160 / 1080 / 720) (used with size)
bezel	Keyword '**bezel**' used when specifying display_width and viewable_width (optional)
display_width	Physical width of the display in mm (optional with bezel)
viewable_width	Physical width of the displays viewable width in mm (optional with bezel)
keep	Keyword '**keep**' required to maintain the original image aspect ratio (optional)
wall_mode	Keyword '**fast**' required for seamless fast swiching (optional)*

* Device firmware > 3.5.2.0 and BlueRiver™ firmware > 2.14.0 required

4.11.1.4 Notes

- If the keyword **size** is not used then the system **default** values will be used as the display resolution.
- If the keyword **bezel** is used and both **display_width** and **viewable_width** values are present then bezel compensation will be automatically calculated and applied otherwise no bezel compensation will be applied.

ToC

4.11.1.5 Return Value

command	<space>	mode	<space>	status	terminator
join	<space>	wall	<space>	*success / error [message]*	<cr>

4.11.1.6 Command Example - 2x2 video wall with 4K source and 1080 displays

```
join wall Encoder1 Decoder1 2x2 1 size 1920 1080 60 bezel 615 595 keep<cr>
join wall Encoder1 Decoder1 2x2 2 size 1920 1080 60 bezel 615 595 keep<cr>
join wall Encoder1 Decoder1 2x2 3 size 1920 1080 60 bezel 615 595 keep<cr>
join wall Encoder1 Decoder1 2x2 4 size 1920 1080 60 bezel 615 595 keep<cr>

join wall key:abc123 Encoder1 Decoder1 2x2 1 size 1920 1080 60 bezel 615 595 keep<cr>
```

4.11.1.7 Return Examples

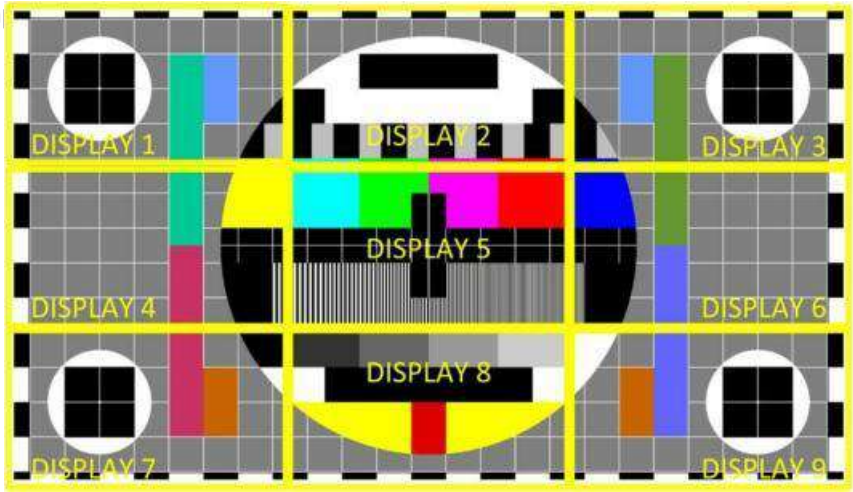
```
join wall success<cr>

join wall error [incomplete]<cr>
join wall error [encoder 'Encoder1' not found]<cr>
join wall error [decoder 'Decoder1' not found]<cr>
join wall error [invalid input value -X, non-negative number expected]<cr>
```

wall_type (columns x rows)

1x2 1x3 1x4 1x5
2x1 2x2 2x3 2x4 2x5
3x1 3x2 3x3 3x4 3x5
4x1 4x2 4x3 4x4 4x5
5x1 5x2 5x3 5x4 5x5
6x1 6x2 6x3 6x4 6x5
7x1 7x2 7x3 7x4 7x5
8x1 8x2 8x3 8x4 8x5

display_position

The position of the display is found by counting from left to right and top to bottom.
Below is an exam



ToC

4.11.2 Command join walladv

4.11.2.1 Command usage

join walladv **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> **[**<*wall_mode*>**]** <*width*> <*height*> <*h_offset*> <*v_offset*> <*keep_width*> <*keep_height*> <*viewport_horiz*> <*viewport_vert*> <*viewport_width*> <*viewport_height*> <*frames_per_second*><cr>

4.11.2.2 Description

The command **join walladv** is used for routing video in a fully managed video wall configuration. This mode is used to maintain output resolution. *Refer appendix B – How to Video wall*

4.11.2.3 Arguments

encoder_device_name	Device name of the source Encoder
decoder_device_name	Device name of the destination Decoder
fast	Keyword required for seamless fast swiching (optional)
width	Display horizontal resolution in pixels
height	Display vertical resolution in pixels
h_offset	Horizontal offset of the displayed video
v_offset	Vertical offset of the displayed video
keep_width	Width of the portion of the source video that is displayed in pixels
keep_height	Height of the portion of the source video that is displayed in pixels
viewport_horiz	Horizontal position of the viewport's top-left corner on the screen in pixels
viewport_vert	Vertical position of the viewport's top-left corner on the screen in pixels
viewport_width	Width of the viewable video in pixels
viewport_height	Height of the viewable video in pixels
frames_per_second	Video frame rate

4.11.2.4 Notes

- Presets created from the UI will contain a comment after each line indicating the Decoder position and wall type as per below:

  join walladv Encoder1 Decoder1 1920 1080 0 0 928 508 0 0 1920 1080 60 **// 2x2 1**
  join walladv Encoder1 Decoder2 1920 1080 992 0 928 508 0 0 1920 1080 60 **// 2x2 2**
  join walladv Encoder1 Decoder3 1920 1080 0 572 928 508 0 0 1920 1080 60 **// 2x2 3**
  join walladv Encoder1 Decoder4 1920 1080 992 572 928 508 0 0 1920 1080 60 **// 2x2 4**

  ***** These comments are required to load an advanced video wall preset into the UI.

ToC

4.11.2.4 Notes continued…

- The **width** and **height** arguments specify respectively the width and the height of the displayed video result after cropping, in pixels.
- 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 **keep_width** and **keep_height** specify respectively the width and the height of the portion of the original source video that is displayed (i.e. 'kept') in pixels.

The viewport is the rectangular region of a screen where video is displayed on Decoders. Areas on the screen outside this region are black. You can use the viewport to apply black to the top and bottom, or sides of the video to maintain original aspect ratio's or change the location of the image on the display.

- 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. This is the area of the displayed video.
- For LIGHTNING, **viewport_horiz** and **viewport_vert** would be set to 0, and **viewport_width** along with **viewport_height** would equal **width** and **height** respectively.
- The **width, keep_width, 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.

4.11.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
join	<space>	walladv	<space>	*success / error [message]*	<cr>

4.11.2.6 Command Examples

2x2 video wall with 4K source and 4K display resolution

```
join walladv Encoder1 Decoder1 3840 2160 0 0 950 1070 1920 0 1920 2160 60<cr>
join walladv Encoder1 Decoder2 3840 2160 970 0 1900 1070 0 0 3840 2160 60<cr>
join walladv Encoder1 Decoder3 3840 2160 2890 0 950 1070 0 0 1920 2160 60<cr>
join walladv Encoder1 Decoder4 3840 2160 970 1090 1900 1070 0 0 3840 2160 60<cr>

join walladv key:abc123 Encoder1 Decoder1 fast 3840 2160 0 0 950 1070 1920 0 1920 2160 60<cr>
```

4.11.2.7 Return Examples

```
join walladv success<cr>

join walladv error [incomplete]<cr>
join walladv error [encoder 'Encoder1' not found]<cr>
join walladv error [decoder 'Decoder1' not found]<cr>
join walladv error [invalid input value -X, non-negative number expected]<cr>
```

ToC

5 Command leave

The **leave** command is used with a Decoder to disconnect from an Encoder's stream.

5.1 Command leave video

5.1.1 Command usage

leave video **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

5.1.2 Description

The command **leave video** is used to disconnect a Decoder from the video stream it is receiving.

5.1.3 Arguments

decoder_device_name	Device name of the Decoder, Group or '**all**'

5.1.4 Notes

- ***group_name*** can be used as a destination when all Decoders in a group are required to leave video.
- **all** can be used as a destination when all the Decoders are required to leave video.
- A connection can be made again with the **join** commands.

5.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	video	<space>	*success / error [message]*	<cr>

5.1.6 Command Examples

```
leave video Decoder1<cr>
leave video Decoder2<cr>
leave video MyGroup<cr>
leave video all<cr>
leave video key:abc123 Decoder1<cr>
```

5.1.7 Return Examples

```
leave video success<cr>

leave video error [incomplete]<cr>
leave video error [decoder 'Decoder1' not found]<cr>
```

ToC

5.2 Command leave sub

5.2.1 Command usage

leave sub **[**key:<*security_key*>**]** <*decoder_device_name*> <*subscription*><cr>

5.2.2 Description

The command **leave sub** is used to disconnect a Decoder from the multiview video stream it is receiving on the specified subscription.

5.2.3 Arguments

decoder_device_name	Device name of the Decoder
subscription	Value from 1 to 31

5.2.4 Notes

- A connection can be made again with the **join multiview** command.
- To leave Decoder subscription 0 use the **leave video** command.

5.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	sub	<space>	*success / error [message]*	<cr>

5.2.6 Command Examples

```
leave sub Decoder1 1<cr>
leave sub Decoder1 3<cr>
leave sub key:abc123 Decoder1 10<cr>
```

5.2.7 Return Examples

```
leave sub success<cr>

leave sub error [incomplete]<cr>
leave sub error [incorrect subscription]<cr>
leave sub error [decoder 'Decoder1' not found]<cr>
leave sub error [invalid input value xyz, number expected]<cr>
```

5.3 Command leave av

5.3.1 Command usage

leave av **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

5.3.2 Description

The command **leave av** is used to disconnect a Decoder from both the video and digital audio streams it is receiving.

5.3.3 Arguments

decoder_device_name	Device name of the Decoder, Group or '**all**'

5.3.4 Notes

- ***group_name*** can be used as a destination when all Decoders in a group are required to leave video and digital audio.
- **all** can be used as a destination when all the Decoders are required to leave video and digital audio.
- A connection can be made again with the **join** commands.

5.3.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	av	<space>	*success / error [message]*	<cr>

5.3.6 Command Examples

```
leave av Decoder1<cr>
leave av Decoder2<cr>
leave av MyGroup<cr>
leave av all<cr>
leave av key:abc123 Decoder1<cr>
```

5.3.7 Return Examples

```
leave av success<cr>

leave av error [incomplete]<cr>
leave av error [decoder 'Decoder1' not found]<cr>
```

5.4 Command leave audio_a

5.4.1 Command usage

leave audio_a **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

5.4.2 Description

The command **leave audio_a** is used to disconnect a Decoder from the analog audio stream it is receiving.

5.4.3 Arguments

decoder_device_name	Device name of the Decoder, Group or '**all**'

5.4.4 Notes

- ***group_name*** can be used as a destination when all Decoders in a group are required to leave analog audio.
- **all** can be used as a destination when all the Decoders are required to leave analog audio.
- A connection can be made again with the **join audio_a** command.

5.4.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	audio_a	<space>	*success / error [message]*	<cr>

5.4.6 Command Examples

```
leave audio_a Decoder1<cr>
leave audio_a Decoder2<cr>
leave audio_a MyGroup<cr>
leave audio_a all<cr>
leave audio_a key:abc123 Decoder1<cr>
```

5.4.7 Return Examples

```
leave audio_a success<cr>

leave audio_a error [incomplete]<cr>
leave audio_a error [decoder 'Decoder1' not found]<cr>
leave audio_a error [not supported]<cr>
```

ToC

5.5 Command leave audio_d

5.5.1 Command usage

leave audio_d **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

5.5.2 Description

The command **leave audio_d** is used to disconnect a Decoder from the digital audio stream it is receiving.

5.5.3 Arguments

decoder_device_name	Device name of the Decoder, Group or '**all**'

5.5.4 Notes

- ***group_name*** can be used as a destination when all Decoders in a group are required to leave digital audio.
- **all** can be used as a destination when all the Decoders are required to leave digital audio.
- A connection can be made again with the **join audio_d** command.

5.5.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	audio_d	<space>	*success / error [message]*	<cr>

5.5.6 Command Examples

```
leave audio_d Decoder1<cr>
leave audio_d Decoder2<cr>
leave audio_d MyGroup<cr>
leave audio_d all<cr>
leave audio_d key:abc123 MyGroup<cr>
```

5.5.7 Return Examples

```
leave audio_d success<cr>

leave audio_d error [incomplete]<cr>
leave audio_d error [decoder 'Decoder1' not found]<cr>
```

ToC

5.6 Command leave all

5.6.1 Command usage

leave all **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

5.6.2 Description

The command **leave all** is used to disconnect a Decoder from all streams it is receiving on all subscription. Analog audio, digital audio along with 32 video subscriptions will be unsubscribed from the Decoder.

5.6.3 Arguments

decoder_device_name	Name of the Decoder or '**all**'

5.6.4 Notes

- **all** can be used as a destination when all the Decoders are required to leave all subscriptions.

5.6.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	all	<space>	*success / error [message]*	<cr>

5.6.6 Command Example

```
leave all Decoder1<cr>
leave all all<cr>
leave all key:abc123 Decoder1<cr>
```

5.6.7 Return Examples

```
leave all success<cr>

leave all error [incomplete]<cr>
leave all error [decoder 'Decoder1' not found]<cr>
```

ToC

6 Command stop

The **stop** command is used to stop an Encoder's stream from being sent on the network. Joins are maintained between Encoders and Decoders but no data is sent from the Encoder.

When the system is running in Multicast AUTO mode the keyword '**free**' and '**free_all**' can be used. These keywords will be ignored if the system is running in Multicast MANUAL mode.

The Encoder's stream multicast address can be released for use by another Encoder's stream when the keyword '**free**' is used in the command line.

Both the Encoder's stream multicast address and Decoder subscriptions are released when the keyword '**free_all**' is used in the command line.

ToC

6.1 Command stop video

6.1.1 Command usage

stop video **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<free/free_all>**]**<cr>

6.1.2 Description

The command **stop video** is used to stop an Encoder's video stream from being sent to any Decoder. The multicast address associated with the stream will be made available for another stream if '**free'** is used.

6.1.3 Arguments

encoder_device_name	Device name of the Encoder or Group
free	Keyword '**free**' removes the multicast address associated with the Encoder stream (optional)
free_all	Keyword '**free_all**' removes the multicast address associated with the Encoder stream and removes all Decoder subscriptions associated with this multicast address (optional)

6.1.4 Notes

- ***group_name*** can be used as a destination when all Encoders in a group are required to stop video.
- A connection can be made again with the command **join** or **start video**.
- Either '**free**' or '**free_all**' can be used but not both.

6.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	video	<space>	*success / error [message]*	<cr>

6.1.6 Command Examples

```
stop video Encoder1<cr>
stop video Encoder1 free_all<cr>
stop video key:abc123 Encoder1<cr>
stop video MyGroup<cr>
stop video MyGroup free<cr>
```

6.1.7 Return Examples

```
stop video success<cr>

stop video error [incomplete]<cr>
stop video error [encoder 'Encoder1' not found]<cr>
```

ToC

6.2 Command stop sub

6.2.1 Command usage

stop sub **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<free/free_all>**]**<cr>

6.2.2 Description

The command **stop sub** is used to stop an Encoder's second (scaled multiview) video stream from being sent to any Decoder. The multicast address associated with the stream will be made available for another stream if '**free**' is used.

6.2.3 Arguments

encoder_device_name	Device name of the Encoder
free	Keyword '**free**' removes the multicast address associated with the Encoder stream (optional)
free_all	Keyword '**free_all**' removes the multicast address associated with the Encoder stream and removes all Decoder subscriptions associated with this multicast address (optional)

6.2.4 Notes

- A connection can be made again with the command **join multiview** or **start sub**.
- Either '**free**' or '**free_all**' can be used but not both.

6.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	sub	<space>	*success / error [message]*	<cr>

6.2.6 Command Examples

```
stop sub Encoder1<cr>
stop sub Encoder1 free<cr>
stop sub Encoder1 free_all<cr>
stop sub key:abc123 Encoder1<cr>
```

6.2.7 Return Examples

```
stop sub success<cr>

stop sub error [incomplete]<cr>
stop sub error [encoder 'Encoder1' not found]<cr>
```

6.3 Command stop av

6.3.1 Command usage

stop av **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<free/free_all>**]**<cr>

6.3.2 Description

The command **stop av** is used to stop an Encoder's video and digital audio streams from being sent to any Decoder. The multicast address associated with the streams will be made available for other streams if '**free**' is used.

6.3.3 Arguments

encoder_device_name	Device name of the Encoder or Group
free	Keyword '**free**' removes the multicast addresses associated with the Encoder streams (optional)
free_all	Keyword '**free_all**' removes the multicast addresses associated with the Encoder streams and removes all Decoder subscriptions associated with this multicast addresses (optional)

6.3.4 Notes

- ***group_name*** can be used as a destination when all Encoders in a group are required to stop video and digital audio.
- A connection can be made again with the command **join** or **start audio_a**.
- Either '**free**' or '**free_all**' can be used but not both.

6.3.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	av	<space>	*success / error [message]*	<cr>

6.3.6 Command Examples

```
stop av Encoder1<cr>
stop av Encoder1 free_all<cr>
stop av key:abc123 Encoder1<cr>
stop av MyGroup<cr>
stop av MyGroup free<cr>
```

6.3.7 Return Examples

```
stop av success<cr>

stop av error [incomplete]<cr>
stop av error [encoder 'Encoder1' not found]<cr>
```

ToC

6.4 Command stop audio_a

6.4.1 Command usage

stop audio_a **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<free/free_all>**]**<cr>

6.4.2 Description

The command **stop audio_a** is used to stop an Encoder's analog audio stream from being sent to any Decoder. The multicast address associated with the stream will be made available for another stream if '**free**' is used.

6.4.3 Arguments

encoder_device_name	Device name of the Encoder or Group
free	Keyword '**free**' removes the multicast address associated with the stream (optional)
free_all	Keyword '**free_all**' removes the multicast address associated with the Encoder stream and removes all Decoder subscriptions associated with this multicast address (optional)

6.4.4 Notes

- ***group_name*** can be used as a destination when all Encoders in a group are required to stop analog audio.
- A connection can be made again with the command **join audio_a** or **start audio_a**.
- Either '**free**' or '**free_all**' can be used but not both.

6.4.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	audio_a	<space>	*success / error [message]*	<cr>

6.4.6 Command Examples

```
stop audio_a Encoder1<cr>
stop audio_a Encoder1 free_all<cr>
stop audio_a key:abc123 Encoder1<cr>
stop audio_a MyGroup<cr>
stop audio_a MyGroup free<cr>
```

6.4.7 Return Examples

```
stop audio_a success<cr>

stop audio_a error [incomplete]<cr>
stop audio_a error [encoder 'Encoder1' not found]<cr>
stop audio_a error [not supported]<cr>
```

6.5 Command stop audio_d

6.5.1 Command usage

stop audio_d **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<free/free_all>**]**<cr>

6.5.2 Description

The command **stop audio_d** is used to stop an Encoder's digital audio stream from being sent to any Decoder. The multicast address associated with the stream will be made available for another stream if '**free**' is used.

6.5.3 Arguments

encoder_device_name	Device name of the Encoder or Group
free	Keyword '**free**' removes the multicast address associated with the Encoder stream (optional)
free_all	Keyword '**free_all**' removes the multicast address associated with the Encoder stream and removes all Decoder subscriptions associated with this multicast address (optional)

6.5.4 Notes

- ***group_name*** can be used as a destination when all Encoders in a group are required to stop digital audio.
- A connection can be made again with the command **join audio_d** or **start audio_d**.
- Either '**free**' or '**free_all**' can be used but not both.

6.5.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	audio_d	<space>	*success / error [message]*	<cr>

6.5.6 Command Examples

```
stop audio_d Encoder1<cr>
stop audio_d Encoder1 free_all<cr>
stop audio_d key:abc123 Encoder1<cr>
stop audio_d MyGroup<cr>
stop audio_d MyGroup free<cr>
```

6.5.7 Return Examples

```
stop audio_d success<cr>

stop audio_d error [incomplete]<cr>
stop audio_d error [encoder 'Encoder1' not found]<cr>
```

ToC

6.6 Command stop ir

6.6.1 Command usage

stop ir [key:<*security_key*>**]** <*device_name*><cr>

6.6.2 Description

The command **stop ir** is used to stop a devices IR stream from being sent to any device.

6.6.3 Arguments

device_name	Device name of either Encoder or Decoder

6.6.4 Notes

- A connection can be made again with the **join ir** command.

6.6.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	ir	<space>	*success / error*	<cr>

6.6.6 Example

```
stop ir Decoder1<cr>
stop ir key:abc123 Decoder1<cr>
```

6.6.7 Return Examples

```
stop ir success<cr>

stop ir error [incomplete]<cr>
stop ir error [device 'Decoder1' not found]<cr>
stop ir error [not supported]<cr>
```

6.7 Command stop serial

6.7.1 Command usage

stop serial **[**key:<*security_key*>**]** <*device_name*> **[**<bi>**]**<cr>

6.7.2 Description

The command **stop serial** is used to stop a devices RS-232 serial stream from being sent to any device.

6.7.3 Arguments

device_name	Device name of either Encoder or Decoder
bi	Keyword '**bi**' will remove 2-way connection to the device (optional)

6.7.4 Notes

- A connection can be made again with the **join serial** command.

6.7.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	serial	<space>	*success / error*	<cr>

6.7.6 Examples

```
stop serial Decoder1<cr>
stop serial Decoder1 bi<cr>
stop serial key:abc123 Decoder1<cr>
```

6.7.7 Return Examples

```
stop serial success<cr>

stop serial error [incomplete]<cr>
stop serial error [device 'Encoder1' not found]<cr>
stop serial error [not supported]<cr>
```

6.8 Command stop usb

6.8.1 Command usage

stop usb **[**key:<*security_key*>**]** <*encoder_device_name*><cr>

6.8.2 Description

The command **stop usb** is used to stop a devices USB stream from being sent to any device.

6.8.3 Arguments

encoder_device_name	Device name of Encoder

6.8.4 Notes

- A connection can be made again with the **join usb** command.

6.8.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
stop	<space>	usb	<space>	*success / error*	<cr>

6.8.6 Example

```
stop usb Encoder1<cr>
stop usb key:abc123 Encoder1<cr>
```

6.8.7 Return Examples

```
stop usb success<cr>

stop usb error [incomplete]<cr>
stop usb error [encoder 'Encoder1' not found]<cr>
stop usb error [not supported]<cr>
```

ToC

7 Command start

The **start** command is used to start an Encoder's stream.

7.1 Command start video

7.1.1 Command usage

start video **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<*multicast_addr*>**]**<cr>

7.1.2 Description

The command **start video** is used to start an Encoder's video stream.

7.1.3 Arguments

encoder_device_name	Device name of the Encoder or Group
multicast_addr	Optional with an Encoder name to provide the streams multicast address

7.1.4 Notes

- A stream cannot be joined unless it has started, so a **join** command will automatically send the **start** command if the Encoder stream is stopped.
- **group_name** can be used as a destination when all Encoders in a group are required to start video.
- The **multicast_addr** argument, if preset, must be an available (unused) multicast IP within the allocation range configured in the SDVoE Director Controller. This argument will be ignored if Multicast Management is set to MANUAL.

7.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
start	<space>	video	<space>	*success / error [message]*	<cr>

7.1.6 Command Examples

```
start video Encoder2<cr>
start video MyGroup<cr>
start video Encoder1 224.1.1.1<cr>
start video key:abc123 Encoder2<cr>
```

7.1.7 Return Examples

```
start video success<cr>

start video error [incomplete]<cr>
start video error [out of range]<cr>
start video error [multicast IP address is in use]<cr>
start video error [encoder 'Encoder1' not found]<cr>
```

ToC

7.2 Command start sub

7.2.1 Command usage

start sub **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<*multicast_addr*>**]**<cr>

7.2.2 Description

The command **start sub** is used to start an Encoder's second scaled video stream used for multiview.

7.2.3 Arguments

encoder_device_name	Device name of the Encoder
multicast_addr	Optional to provide the streams multicast address

7.2.4 Notes

- A stream cannot be joined unless it has started, so the **join** commands will automatically send the **start** command if the Encoder stream is stopped.
- A sub stream cannot be started without a scaler resolution being set. The scaler resolution only needs to be set once or whenever a change is required. The scaler resolution is only cleared with a factory reset. Refer 'Command set scaler' for further details.
- The **multicast_addr** argument, if preset, must be an available (unused) multicast IP within the allocation range configured in the SDVoE Director Controller. This argument will be ignored if Multicast Management is set to MANUAL.

7.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
start	<space>	sub	<space>	*success / error [message]*	<cr>

7.2.6 Command Examples

```
start sub Encoder1 224.1.1.1<cr>
start sub Encoder2<cr>
start sub key:abc123 Encoder1<cr>
```

7.2.7 Return Examples

```
start sub success<cr>

start sub error [incomplete]<cr>
start sub error [encoder 'Encoder1' wrong type]<cr>
start sub error [encoder 'Encoder1' not found]<cr>
start sub error [multicast IP address is in use]<cr>
start sub error [encoder 'Encoder1' scaler not set]<cr>
```

ToC

7.3 Command start av

7.3.1 Command usage

start av **[**key:<*security_key*>**]** <*encoder_device_name*>
[<audio_*multicast_addr*> <video_multicast_addr>**]**<cr>

7.3.2 Description

The command **start av** is used to start an Encoder's video and digital audio streams.

7.3.3 Arguments

encoder_device_name	Device name of the Encoder or Group
audio_multicast_addr	Optional with an Encoder name to provide the audio stream multicast address
video_multicast_addr	Optional with an Encoder name to provide the video stream multicast address

7.3.4 Notes

- A stream cannot be joined unless it has started, so the **join** commands will automatically send the **start** command if the Encoder stream is stopped.
- ***group_name*** can be used as a destination when all Encoders in a group are required to start video and digital audio.
- The **audio_multicast_addr** and **video_multicast_addr** arguments, if preset, must be an available (unused) multicast IP within the allocation range configured in the SDVoE Director Controller. Both audio_multicast_addr and video_multicast_addr must both be preset if used. These arguments will be ignored if Multicast Management is set to MANUAL.

7.3.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
start	<space>	av	<space>	*success / error [message]*	<cr>

7.3.6 Command Examples

```
start av Encoder1 224.1.1.1 224.1.1.2<cr>
start av Encoder2<cr>
start av MyGroup<cr>
start av key:abc123 Encoder1<cr>
```

7.3.7 Return Examples

```
start av success<cr>

start av error [incomplete]<cr>
start av error [multicast IP address is in use]<cr>
start av error [encoder 'Encoder1' not found]<cr>
```

ToC

7.4 Command start audio_a

7.4.1 Command usage

start audio_a **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<*multicast_addr*>**]**<cr>

7.4.2 Description

The command **start audio_a** is used to start an Encoder's analog audio stream.

7.4.3 Arguments

encoder_device_name	Device name of the Encoder or Group
multicast_addr	Optional with an Encoder name to provide the streams multicast address

7.4.4 Notes

- A stream cannot be joined unless it has started, so the **join** commands will automatically send the **start** command if the Encoder stream is stopped.
- ***group_name*** can be used as a destination when all Encoders in a group are required to start analog audio.
- The **multicast_addr** argument, if preset, must be an available (unused) multicast IP within the allocation range configured in the SDVoE Director Controller.

7.4.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
start	<space>	audio_a	<space>	*success / error [message]*	<cr>

7.4.6 Command Examples

```
start audio_a Encoder1 224.1.1.1<cr>
start audio_a Encoder2<cr>
start audio_a MyGroup<cr>
start audio_a key:abc123 Encoder1<cr>
```

7.4.7 Return Examples

```
start audio_a success<cr>

start audio_a error [incomplete]<cr>
start audio_a error [encoder 'Encoder1' not found]<cr>
start audio_a error [multicast IP address is in use]<cr>
start audio_a error [not supported]<cr>
```

7.5 Command start audio_d

7.5.1 Command usage

start audio_d **[**key:<*security_key*>**]** <*encoder_device_name*> **[**<*multicast_addr*>**]**<cr>

7.5.2 Description

The command **start audio_d** is used to start an Encoder's digital audio stream.

7.5.3 Arguments

encoder_device_name	Device name of the Encoder or Group
multicast_addr	Optional with an Encoder name to provide the streams multicast address

7.5.4 Notes

- A stream cannot be joined unless it has started, so the **join** commands will automatically send the **start** command if the Encoder stream is stopped.
- ***group_name*** can be used as a destination when all Encoders in a group are required to start digital audio.
- The **multicast_addr** argument, if preset, must be an available (unused) multicast IP within the allocation range configured in the SDVoE Director Controller. This argument will be ignored if Multicast Management is set to MANUAL.

7.5.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
start	<space>	audio_d	<space>	*success / error [message]*	<cr>

7.5.6 Command Examples

```
start audio_d Encoder1 224.1.1.1<cr>
start audio_d Encoder2<cr>
start audio_d MyGroup<cr>
start audio_d key:abc123 Encoder1<cr>
```

7.5.7 Return Examples

```
start audio_d success<cr>

start audio_d error [incomplete]<cr>
start audio_d error [encoder 'Encoder1' not found]<cr>
start audio_d error [multicast IP address is in use]<cr>
```

ToC

8 Command set

The **set** commands are used to change the working conditions of an Encoder or Decoder.

8.1 Command set audio_io

8.1.1 Command usage

set audio_io **[**key:<*security_key*>**]** <*encoder_device_name*> <*value*><cr>

8.1.2 Description

The command **set audio_io** is used to change the analog audio connector of an Encoder from an input to an output.

8.1.3 Arguments

encoder_device_name	Device name of the Encoder
value	'in' / 'out'

8.1.4 Notes

- Use the command **get audio_io** to retrieve this setting.

8.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	audio_io	<space>	*success / error [message]*	<cr>

8.1.6 Command Examples

```
set audio_io Encoder1 in<cr>
set audio_io Encoder1 out<cr>
set audio_io key:abc123 Encoder1 in<cr>
```

8.1.7 Return Examples

```
set audio_io success<cr>

set audio_io error [incomplete]<cr>
set audio_io error [encoder 'Encoder1' not found]<cr>
set audio_io error [not supported]<cr>
```

8.2 Command set audio_out

8.2.1 Command usage

set audio_out **[**key:<*security_key*>**]** <*decoder_device_name*> <*value*><cr>

8.2.2 Description

The command **set audio_out** is used to change the analog audio output source on a Decoder from HDMI or analog audio.

8.2.3 Arguments

decoder_device_name	Device name of the Decoder
value	'hdmi' / 'analog'

8.2.4 Notes

- Use the command **get audio_out** to retrieve this setting.

8.2.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	audio_out	<space>	*success / error [message]*	<cr>

8.2.6 Command Examples

```
set audio_out Decoder1 in<cr>
set audio_ out Decoder1 out<cr>
set audio_out key:abc123 Decoder1 in<cr>
```

8.2.7 Return Examples

```
set audio_out success<cr>

set audio_out error [incomplete]<cr>
set audio_out error [decoder 'Decoder1' not found]<cr>
set audio_out error [not supported]<cr>
```

8.3 Command set audio_source

8.3.1 Command usage

set audio_source **[**key:<*security_key*>**]** <*decoder_device_name*> <*value*><cr>

8.3.2 Description

The command **set audio_source** is used to change a Decoder's HDMI audio from embedded HDMI or analog audio.

8.3.3 Arguments

decoder_device_name	Device name of the Decoder
value	'hdmi' / 'analog'

8.3.4 Notes

- Use the command **get audio_source** to retrieve this setting.

8.3.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	audio_source	<space>	*success / error [message]*	<cr>

8.3.6 Command Examples

```
set audio _source Decoder1 hdmi<cr>
set audio _source Decoder1 analog<cr>
set audio _source key:abc123 Decoder1 hdmi<cr>
```

8.3.7 Return Examples

```
set audio_source success<cr>

set audio_source error [incomplete]<cr>
set audio_source error [decoder 'Decoder1' not found]<cr>
set audio_source error [not supported]<cr>
```

8.4 Command set edid

8.4.1 Command usage

set edid **[**key:<*security_key*>**]** <*encoder_device_name*> <*edid_data*><cr>

8.4.2 Description

The command **set edid** is used to save EDID into Encoders.
While the command **get edid** is used to retrieve EDID from a Decoder.

8.4.3 Arguments

encoder_device_name	Name of the Encoder, Group or '**all**'
edid_data	String which represents the binary EDID data

8.4.4 Notes

- **all** can be used as a destination when all the Encoders are to be set with the same EDID.
- ***group_name*** can be used as a destination when all Encoders in a group are to be set with the same EDID.
- The data argument must be a 512 character hexadecimal string which represents the EDID to be set.

8.4.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	edid	<space>	*success / error [message]*	<cr>

8.4.6 Command Example

```
set edid Encoder1 00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818
0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000
00fd00384c1e5315000a20202020202000000fc005068696c69707320323231450a01b602031ef04b100501020304061213 1
41f23090707830100006503 0c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c110000
18011d007251d01e206e285500dc0c1100001e8c0ad090204031200c405500dc0c110000180000000000000000000000000000
000000000000000000000000ac<cr>

set edid key:abc123 Encoder1 …<cr>
```

8.4.7 Return Examples

```
set edid success<cr>

set edid error [incomplete]<cr>
set edid error [encoder 'Encoder1' not found]<cr>
```

8.5 Command set frame_converter

8.5.1 Command usage

set frame_converter **[**key:<*security_key*>**]** <encoder_*device_name*> <*stream*> <*state*><cr>

8.5.2 Description

The command **set frame_converter** is used to half the Encoder's video frame rate on either the main or sub streams. For example 60 fps video from the source becomes a 30 fps video stream.
Reducing the frame rate will dramatically reduce the bandwidth of a video stream with little affect to the resulting video with a 50/60Hz source frame rate.

8.5.3 Arguments

encoder_device_name	Device name of the Encoder
stream	'sub' / 'main'
state	'true' / 'false'

8.5.4 Notes

- Only PROGRESSIVE video signals are supported. INTERLACED video signals cannot have their frame rates reduced. The command will return an invalid format error with INTERLACED signals when setting the frame_converter to true.

  Use the "get <*encoder_device_name*> video sm" command to confirm a PROGRESSIVE signal.

8.5.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	frame_converter	<space>	*success / error [message]*	<cr>

8.5.6 Command Examples

```
set frame_converter Encoder1 sub true<cr>
set frame_converter Encoder1 sub false<cr>
set frame_converter Encoder1 main true<cr>
set frame_converter Encoder1 main false<cr>
set frame_converter key:abc123 Encoder1 sub true<cr>
```

8.5.7 Return Examples

```
set frame_converter success<cr>

set frame_converter error [incomplete]<cr>
set frame_converter error [invalid format]<cr>
set frame_converter error [encoder 'Encoder1' not found]<cr>
```

ToC

8.6 Command set listener

8.6.1 Command usage

set listener **[**key:<*security_key*>**]** <*ip*> <notify_*port*> <*protocol*> <*condition*> <*state*> <*device_port*> <*preset*><cr>

8.6.2 Description

The command **set listener** is used to respond to a Global Cachè sensor notify message and execute a corresponding preset.

8.6.3 Arguments

ip	IP of a GC-100-06 or GC-100-12 devices, otherwise 239.255.250.250
notify_port	4998 for GC-100-06 or GC-100-12, otherwise same as port set in device
protocol	'TCP' for GC-100-06 or GC-100-12 devices, otherwise 'UDP'
condition	'on', 'off' or 'both'
state	'true' / 'false'
device_port	Device I/O port 1 to 6 depending on the model used
preset	System preset name

8.6.4 Notes

- Supported Global Cachè devices GC-100-06, GC-100-12, GC-100-18, IP2IR, WF2IR, GCIR3, Flex Link Relay & Sensor Cable

8.6.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	listener	<space>	*success / error [message]*	<cr>

8.6.6 Command Examples

```
set listener 172.30.0.12 4998 tcp on true 1 preset1<cr>
set listener 239.255.250.250 9132 udp on true 1 preset2<cr>
set listener 172.30.0.12 4998 tcp on 1 false<cr>
set listener 239.255.250.250 9132 udp on 1 false<cr>
set listener key:abc123 172.30.0.12 4998 tcp on true 1 preset1<cr>
```

8.6.7 Return Examples

```
set listener success<cr>

set listener error [incomplete]<cr>
set listener error [preset 'preset1' not found]<cr>
```

ToC

8.7 Command set presenter

8.7.1 Command usage

set presenter **[**key:<*security_key*>**]** <*group*> <*state*><cr>

8.7.2 Description

The command **set presenter** is used to enable or disable a group of presenter buttons.

8.7.3 Arguments

group	Name of the presenting group
state	'true' / 'false'

8.7.4 Notes

- The Presenter functionality must be unlocked and the service must be enabled.

8.7.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	presenter	<space>	*success / error [message]*	<cr>

8.7.6 Command Examples

```
set presenter group1 true<cr>
set presenter group1 false<cr>
set presenter key:abc123 group1 true<cr>
```

8.7.7 Return Examples

```
set presenter success<cr>

set presenter error [incomplete]<cr>
set presenter error [group 'group11' not found]<cr>
set presenter error [function not available]<cr>
set presenter error [sevice not enabled]<cr>
```

8.8 Command set scaler

8.8.1 Command usage

set scaler **[**key:<*security_key*>**]** <encoder_*device_name*> <*width*> <*height*><cr>

8.8.2 Description

The command **set scaler** is used to set the Encoder's multiview sub stream resolution.

8.8.3 Arguments

encoder_device_name	Device name of the Encoder
width	Horizontal size of the scaled video in pixels
height	Vertical size of the scaled video in pixels

8.8.4 Notes

- The scaler size must match the window size as specified with the **layout window** command.
- The **set scaler** command is not required if specifying the **layout_name** within the **join multi** command.
- A multiview stream cannot be started without a scaler resolution being set. The scaler resolution only needs to be set once or whenever a change is required. The scaler resolution is only cleared with a factory reset.

8.8.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	scaler	<space>	*success / error [message]*	<cr>

8.8.6 Command Examples

```
set scaler Encoder1 1920 1080<cr>
set scaler key:abc123 Encoder1 1920 1080<cr>
```

8.8.7 Return Examples

```
set scaler success<cr>

set scaler error [incomplete]<cr>
set error [encoder 'Encoder1' not found]<cr>
```

8.9 Command set security

8.9.1 Command usage

set security **[**key:<*security_key*>**]** <*device*_name> <*key*><cr>

8.9.2 Description

The command **set security** is used to enable or disable encryption between an Encoder and Decoders. With this function, you can assign individual or a group of devices with a user defined key to control which endpoints in the system can exchange HDMI AV data. You can also use this function to prevent unauthorized Decoders from accessing HDMI AV data.

8.9.3 Arguments

device_name	Name of the Encoder, Decoder or *Group*
key	User defined key or 'default'

8.9.4 Notes

- To enable encryption the key must be set with an eight (8) charator ASCII string.
- To disable encryption the key must be set with the word 'default'.
- Device firmware > 3.5.2.0 and BlueRiver™ firmware > 2.14.0 required

8.9.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	security	<space>	*success / error [message]*	<cr>

8.9.6 Command Examples

```
set security Encoder1 12345678<cr>
set security Decoder1 12345678<cr>
set security group1 12345678<cr>
set security group1 default<cr>
set security key:abc123 Encoder1 12345678<cr>
```

8.9.7 Return Examples

```
set security success<cr>

set security error [incomplete]<cr>
set security error [key must be 8 charators]<cr>
set security error [device 'Encoder1' not found]<cr>
```

ToC

8.10 Command set video_compress

8.10.1 Command usage

set video_compress **[**key:<*security_key*>**]** <*encoder_device_name*> <*state*><cr>

8.10.2 Description

The command **set video_compress** is used to apply constant compression to the video signal for reduced network bandwidth. It reduces 4K30 to around 5 Gbps bandwidth, and 1080p60 to around 2.5 Gbps bandwidth.

8.10.3 Arguments

encoder_device_name	Device name of the Encoder
state	'true' / 'false'

8.10.4 Notes

- The scaled sub stream remains uncompressed

8.10.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	video_compress	<space>	*success / error [message]*	<cr>

8.10.6 Command Examples

```
set video_compress Encoder1 true<cr>
set video_compress Encoder1 false<cr>
set video_compress key:abc123 Encoder1 true<cr>
```

8.10.7 Return Examples

```
set video_compress success<cr>

set video_compress error [incomplete]<cr>
set video_compress error [encoder 'Encoder1' not found]<cr>
```

8.11 Command set video_mode

8.11.1 Command usage

set video_mode **[**key:<*security_key*>**]** <*decoder_device_name*> <*display_mode*> [<*aspect*>] **[**<*width*> <*height*> <*fps*>**]**<cr>

8.11.2 Description

The command **set video_mode** is used to change a Decoder's mode of operation, from a low latency sync or sync_scale modes to a fast switch mode and set the output aspect ratio and resolution if required.

8.11.3 Arguments

decoder_device_name	Device name of the Decoder, Group or '**all**'
display_mode	'**sync**' / '**sync_scale**' / '**fast**'
aspect	Aspect ratio keywords '**keep**', '**stretch**' and '**crop**' for '**fast**' display_mode only (optional)
width	Display resolution width for sync_scale and fast modes
height	Display resolution height for sync_scale and fast modes
fps	Display frame rate for fast mode (sync_scale uses source fps)

8.11.4 Notes

- If no source video is present when setting sync_scale then 60 fps will be used by default.

8.11.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	video_mode	<space>	*success / error [message]*	<cr>

8.11.6 Command Examples

```
set video_mode Decoder1 sync<cr>
set video_mode Decoder1 sync_scale 1920 1080<cr>
set video_mode Decoder1 fast 1920 1080 60<cr>
set video_mode Decoder1 fast keep 1920 1080 60<cr>
set video_mode Decoder1 fast crop 1920 1080 60<cr>
set video_mode Decoder1 fast stretch 1920 1080 60<cr>
```

8.11.7 Return Examples

```
set video_mode success<cr>

set video_mode error [incomplete]<cr>
set video_mode error [decoder 'Decoder1' not found]<cr>
```

ToC

8.12 Command set video_mute

8.12.1 Command usage

set video_mute **[**key:<*security_key*>**]** <*decoder_device_name*> <*state*> **[**<*colour*>**]**<cr>

8.12.2 Description

The command **set video_mute** can be used in a number of useful ways. It can be used as either a video mute to blank the video of the display and show a black screen or any select colour. It can also be useful in identifying individual monitors by displaying a full screen of colour so you can find what monitor is connected to what Decoder.

8.12.3 Arguments

decoder_device_name	Device name of the Decoder
state	'true' / 'false'
colour	Option to change the colour of the display rather than showing black screen

8.12.4 Notes

- **colour** string has a format of RRGGBB where each pair of hexadecimal numbers represent the 8-bit value
- This mute feature is only supported when the Decoder is in a display modes other than sync_scale.

8.12.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	video_mute	<space>	*success / error [message]*	<cr>

8.12.6 Command Examples

```
set video_mute Decoder1 true<cr>
set video_mute Decoder1 true 112233<cr>
set video_mute Decoder1 false<cr>
set video_mute key:abc123 Decoder1 true<cr>
```

8.12.7 Return Examples

```
set video_mute success<cr>

set video_mute error [incomplete]<cr>
set video_mute error [decoder 'Decoder1' not found]<cr>
```

ToC

8.13 Command set video_source

8.13.1 Command usage

set video_source **[**key:<*security_key*>**]** <*encoder_device_name*> <*value*><cr>

8.13.2 Description

The command **set video_source** is used to change an Encoder's input between HDMI and DisplayPort.

8.13.3 Arguments

encoder_device_name	Device name of the Encoder
value	'auto' / 'hdmi' / 'dp'

8.13.4 Notes

- '**auto**' will automatically switch between HDMI and DisplayPort when a signal is detected. '**hdmi**' is for HDMI only, while '**dp**' is for DisplayPort only.
- Use the command **get video_source** to retrieve this setting.

8.13.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
set	<space>	video_source	<space>	*success / error [message]*	<cr>

8.13.6 Command Examples

```
set video_source Encoder1 auto<cr>
set video_source Encoder1 hdmi<cr>
set video_source Encoder1 dp<cr>
set video_source key:abc123 Encoder1 auto<cr>
```

8.13.7 Return Examples

```
set video_source success<cr>

set video_source error [incomplete]<cr>
set video_source error [encoder 'Encoder1' not found]<cr>
```

ToC

9 Command get

The **get** commands are used to retrieve information from the system or Encoders and Decoders.

9.1 Command get api

9.1.1 Command usage

get api **[**key:<*security_key*>**]**<cr>

9.1.2 Description

The command **get api** is used to retrieve the BlueRiver™ API version.

9.1.3 Arguments

None

9.1.4 Notes

- Use the command **get api** in a preset to determine what commands can be used.

9.1.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	api	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.1.6 Command Example

```
get api<cr>
get api key:abc123<cr>
```

9.1.7 Return Example

```
get api success 2.14.0.0<cr>
```

ToC

9.2 Command get audio_io

9.2.1 Command usage

get audio_io **[**key:<*security_key*>**]** <*encoder_device_name*><cr>

9.2.2 Description

The command **get audio_io** is used to retrieve the input/output status of an Encoder's analog audio connector.

9.2.3 Arguments

encoder_device_name	Device name of the Encoder

9.2.4 Notes

- Returned mode is either **in** or **out.**
- Use the command **set audio_io** to change this setting.

9.2.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	audio_io	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.2.6 Command Example

```
get audio_io Encoder1<cr>
get audio_io key:abc123 Encoder1<cr>
```

9.2.7 Return Examples

```
get audio_io success in<cr>
get audio_io success out<cr>

get audio_io error [incomplete]<cr>
get audio_io error [not supported]<cr>
get audio_io error [encoder 'Encoder1' not found]<cr>
```

ToC

9.3 Command get audio_out

9.3.1 Command usage

get audio_out **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.3.2 Description

The command **get audio_out** is used to retrieve the analog audio output source on a Decoder.

9.3.3 Arguments

decoder_device_name	Device name of the Decoder

9.3.4 Notes

- Returned mode is either **hdmi** or **analog.**
- Use the command **set audio_out** to change this setting.

9.3.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	audio_io	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.3.6 Command Example

```
get audio_out Decoder1<cr>
get audio_out key:abc123 Decoder1<cr>
```

9.3.7 Return Examples

```
get audio_out success hdmi<cr>
get audio_out success analog<cr>

get audio_out error [incomplete]<cr>
get audio_out error [not supported]<cr>
get audio_out error [decoder 'Decoder1' not found]<cr>
```

ToC

9.4 Command get audio_source

9.4.1 Command usage

get audio_source **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.4.2 Description

The command **get audio_source** is used to retrieve the source for a Decoder's Analog audio output.

9.4.3 Arguments

decoder_device_name	Device name of the Decoder

9.4.4 Notes

- **hdmi_sync** and **hdmi_multi** modes cannot be set by this API. The standard conditions for the analog audio source would either be analog or hdmi.
- Returned mode is either **analog, hdmi**, **hdmi_sync** or **hdmi_multi.**
- Use the command **set audio_source** to change this setting.

9.4.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	audio_source	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.4.6 Command Example

```
get audio_source Decoder1<cr>
get audio_source key:abc123 Decoder1<cr>
```

9.4.7 Return Examples

```
get audio_source success hdmi_sync<cr>
get audio_source success analog<cr>
get audio_source success hdmi<cr>
get audio_source success hdmi_multi<cr>

get audio_source error [incomplete]<cr>
get audio_source error [not supported]<cr>
get audio_source error [decoder 'Decoder1' not found]<cr>
```

ToC

9.5 Command get bandwidth

9.5.1 Command usage

get bandwidth **[**key:<*security_key*>**]** <*device_name*> **[**<*index*>**]**<cr>

9.5.2 Description

The command **get bandwidth** is used to retrieve the network bandwidth of a given Encoder stream or Decoder subscription.

9.5.3 Arguments

device_name	Device name of either Encoder or Decoder
index	Stream/Subscription index (optional)

9.5.4 Notes

- When **index** is omitted the total bandwidth will be returned.
  For an Encoder this will be the total transmitted bandwidth of stream 0 and stream 1.
  For a Decoder this will be the total received bandwidth of all unique subscriptions.

9.5.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	bandwidth	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.5.6 Command Example

```
get bandwidth Encoder1<cr>
get bandwidth Encoder1 0<cr>
get bandwidth Encoder1 1<cr>
get bandwidth Decoder1<cr>
get bandwidth Decoder1 27<cr>
get bandwidth key:abc123 Encoder1<cr>
```

9.5.7 Return Example

```
get bandwidth success 0.0<cr>
get bandwidth success 3.33<cr>

get bandwidth error [incomplete]<cr>
get bandwidth error [not supported]<cr>
get bandwidth error [device 'Decoder1' not found]<cr>
```

ToC

9.6 Command get devices

9.6.1 Command usage

get devices **[**key:<*security_key*>**]** <*target*><cr>

9.6.2 Description

The command **get devices** is used to retrieve the name and MAC Address of available devices.

9.6.3 Arguments

target	'all' / 'all_rx' / 'all_tx'

9.6.4 Notes

- Return value <device_name> = name of device
- Return value <device_id> = device MAC Address

9.6.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	devices	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.6.6 Command Example

```
get devices all<cr>
get devices all_tx<cr>
get devices all_rx<cr>
get devices key:abc123 all<cr>
```

9.6.7 Return Examples

```
get devices success '<device_name>-<device_id>','<device_name>-<device_id>'<cr>
get devices success '<device_name>-<device_id>'<cr>

get devices error [incomplete]<cr>
```

ToC

9.7 Command get display_status

9.7.1 Command usage

get display_status **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.7.2 Description

The command **get display_status** is used to find if a Decoder has a display connected.

9.7.3 Arguments

decoder_device_name	Device name of the Decoder

9.7.4 Notes

- Returned mode is either **true** or **false.**
- Some non-compliant displays will need to be powered on before detection is possible.

9.7.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	display_status	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.7.6 Command Example

```
get display_status Decoder1<cr>
get display_status key:abc123 Decoder1<cr>
```

9.7.7 Return Examples

```
get display_status success true<cr>
get display_status success false<cr>

get display_status error [incomplete]<cr>
get display_status error [not supported]<cr>
get display_status error [decoder 'Decoder1' not found]<cr>
```

ToC

9.8 Command get edid

9.8.1 Command usage

get edid **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.8.2 Description

The command **get edid** is used to retrieve a Decoder's connected displays EDID.

9.8.3 Arguments

decoder_device_name	Device name of the Decoder

9.8.4 Notes

- A display device must be connected to the Decoder to retrieve the EDID.
- Use the command **set edid** to save EDID into an Encoder.
- 256 bytes of EDID will be returned.

9.8.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**edid_data**	**terminator**
get	<space>	edid	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.8.6 Command Example

```
get edid Decoder1<cr>
get edid key:abc123 Decoder1<cr>
```

9.8.7 Return Examples

```
get edid success
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f8180818a9500950fb3000
101d1c0023a801871382d40582c4500dc0c1100001e000000ff004155353130313230313137313700000fd00384c1e531500
0a202020202020000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131 41f230907078301000
065030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e20
6e285500dc0c1100001e8c0ad090204031200c405500dc0c1100001800000000000000000000000000000000000000000000
00000ac<cr>

get edid error [incomplete]<cr>
get edid error [decoder 'Decoder1' not found]<cr>
get edid error [decoder 'Decoder1' no EDID available]<cr>
```

ToC

9.9 Command get encoder

9.9.1 Command usage

get encoder **[**key:<*security_key*>**]** <*decoder_device_name*> <*subscription*> [<*index*>]<cr>

9.9.2 Description

The command **get encoder** is used to retrieve the Encoder name subscribed to a Decoder's subscription. When subscription is video and index is not specified, then the Encoder subscribed to the Decoder's video index 0 will be returned.

9.9.3 Arguments

decoder_device_name	Device name of a Decoder
subscription	'video' / 'audio_d' / 'audio_a'
index	0..31 (optional)

9.9.4 Notes

- **index** is only required for subscription **video**. **audio_a** and **audio_d** do not use this argument.

9.9.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	encoder	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.9.6 Command Examples

```
get encoder Decoder1 video<cr>
get encoder Decoder1 video 0<cr>
get encoder Decoder1 video 26<cr>
get encoder Decoder1 audio_d<cr>
get encoder Decoder1 audio_a<cr>
get encoder key:abc123 Decoder1 video<cr>
```

9.9.7 Return Examples

```
get encoder success Encoder1 <cr>

get encoder error [incomplete]<cr>
get encoder error [no encoder connected]<cr>
get encoder error [not supported]<cr>
get encoder error [decoder 'Decoder1' not found]<cr>
```

ToC

9.10 Command get frame_converter

9.10.1 Command usage

get **[**key:<*security_key*>**]** <*encoder_device_name*> <*stream*><cr>

9.10.2 Description

The command **get frame_converter** is used to retrieve the current frame rate converter setting for the specified video stream.

9.10.3 Arguments

encoder_device_name	Device name of the Encoder
stream	sub / main

9.10.4 Notes

9.10.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	frame_converter	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.10.6 Examples

```
get frame_converter Encoder1 sub<cr>
get frame_converter Encoder1 main<cr>
get frame_converter key:abc123 Encoder1 sub<cr>
```

9.10.7 Return Examples

```
get frame_converter success true<cr>
get frame_converter success false<cr>

get frame_converter error [incomplete]<cr>
get frame_converter error [encoder 'Encoder1' not found]<cr>
```

9.11 Command get preferred

9.11.1 Command usage

get preferred **[**key:<*security_key*>**]** <*decoder_device_name*> <*resolution*><cr>

9.11.2 Description

The command **get preferred** is used to retrieve the preferred resolution of a display connected to a Decoder.

9.11.3 Arguments

decoder_device_name	Device name of Decoder
resolution	'width' / 'height' / 'fps'

9.11.4 Notes

- A display must be connected to the Decoder for the EDID to be retrieved. Some non-complient displays may need to be switched on before the EDID can be accessed.

9.11.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	preferred	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.11.6 Command Example

```
get preferred Decoder1 width<cr>
get preferred Decoder1 height<cr>
get preferred Decoder1 fps<cr>
get preferred key:abc123 Decoder1 width<cr>
```

9.11.7 Return Example

```
get preferred success 1920<cr>
get preferred success 1080<cr>
get preferred success 60<cr>

get preferred error [incomplete]<cr>
get preferred error [not supported]<cr>
get preferred error [decoder 'Decoder1' not found]<cr>
get preferred error [no EDID available]<cr>
```

9.12 Command get presenter

9.12.1 Command usage

get presenter **[**key:<*security_key*>**]** <*group*><cr>

9.12.2 Description

The command **get present** is used to retrieve a presenting groups current state.

9.12.3 Arguments

group	Name of the Group

9.12.4 Notes

- Group names have no spaces eg 'group1'

9.12.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	presenter	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.12.6 Command Example

```
get presenter group1<cr>
get presenter key:abc123 group1<cr>
```

9.12.7 Return Example

```
get presenter success true<cr>
get presenter success false<cr>

get presenter error [incomplete]<cr>
get presenter error [group 'group11' not found]<cr>
get presenter error [function not available]<cr>
get presenter error [sevice not enabled]<cr>
```

9.13 Command get scaler

9.13.1 Command usage

get scaler **[**key:<*security_key*>**]** <*encoder_device_name*> <*option*><cr>

9.13.2 Description

The command **get scaler** is used to retrieve the scaled video resolution of an Encoder's sub stream.

9.13.3 Arguments

encoder_device_name	Device name of the Encoder
option	'all' / 'width' / 'height' / 'fps'

9.13.4 Notes

9.13.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	scaler	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.13.6 Command Examples

```
get scaler Encoder1 all<cr>
get scaler Encoder1 width<cr>
get scaler Encoder1 height<cr>
get scaler Encoder1 fps<cr>
get scaler key:abc123 Encoder1 all<cr>
```

9.13.7 Return Examples

```
get scaler success 1920 1080 60<cr>
get scaler success 1920<cr>
get scaler success 1080<cr>
get scaler success 60<cr>

get scaler error [incomplete]<cr>
get scaler error [not supported]<cr>
get scaler error [encoder 'Encoder1' not found]<cr>
```

9.14 Command get security

9.14.1 Command usage

get security **[**key:<*security_key*>**]** <*device_name*><cr>

9.14.2 Description

The command **get security** is used to retrieve current security status of an Encoder or Decoder.

9.14.3 Arguments

device_name	Device name of Encoder or Decoder

9.14.4 Notes

- Device firmware > 3.5.2.0 and BlueRiver™ firmware > 2.14.0 required

9.14.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	security	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.14.6 Command Example

```
get security Encoder1<cr>
get security key:abc123 Encoder1<cr>
```

9.14.7 Return Example

```
get security success default<cr>
get security success user_defined<cr>

get security error [incomplete]<cr>
get security error [device 'Encoder1' not found]<cr>
```

ToC

9.15 Command get status

9.15.1 Command usage

get status **[**key:<*security_key*>**]** <*device_name*> [<*stream*> [<*index*]]<cr>

9.15.2 Description

The command **get status** is used to retrieve the status of the specified device or individual Encoder device stream or Decoder subscription.

9.15.3 Arguments

device_name	Name of the Encoder or Decoder
stream	"audio_a", "audio_d" or "video"
index	To be used with "video" only. For Encoder 0 – 1 and for Decoder 0 – 31.

9.15.4 Notes

- When no stream is specified the return will be as seen on the status of the Encoder / Decoder UI Status tab, this is a general status of the device.

9.15.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	status	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.15.6 Command Example

```
get status Encoder1<cr>
get status Encoder1 audio_a<cr>
get status Encoder1 audio_d<cr>
get status Encoder1 video 0<cr>
get status Encoder1 video 1<cr>
get status key:abc123 Encoder1<cr>
```

```
get status Decoder1<cr>
get status Decoder1 audio_a<cr>
get status Decoder1 audio_d<cr>
get status Decoder1 video 0<cr>
get status Decoder1 video 31<cr>
```

9.15.7 Return Examples

```
get status success CONNECTED<cr>
get status success STOPPED<cr>
get status success TIMEOUT<cr>
get status success DISCONNECTED<cr>
get status success OUT OF RANGE<cr>

get status error [incomplete]<cr>
get status error [not supported]<cr>
get status error [invalid index]<cr>
get status error [device 'Encoder1' not found]<cr>
```

ToC

9.16 Command get temp

9.16.1 Command usage

get temp **[**key:<*security_key*>**]** <*device_name*><cr>

9.16.2 Description

The command **get temp** is used to retrieve the current processor temperature of a Decoder or Encoder.

9.16.3 Arguments

device_name	Device name of either a Decoder or Encoder

9.16.4 Notes

- The result is in degrees celsius (°C). The maximum operating temperature should not exceed 85°C (185°F).

9.16.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	temp	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.16.6 Command Examples

```
get temp Encoder1<cr>
get temp key:abc123 Encoder1<cr>
```

9.16.7 Return Examples

```
get temp success 69 <cr>

get temp error [incomplete]<cr>
get temp error [device 'Encoder1' not found]<cr>
```

ToC

9.17 Command get ver

9.17.1 Command usage

get ver **[**key:<*security_key*>**]** <*device_name*><cr>

9.17.2 Description

The command **get ver** is used to retrieve the current firmware version of a Decoder or Encoder.

9.17.3 Arguments

device_name	Device name of either a Decoder or Encoder

9.17.4 Notes

9.17.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	ver	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.17.6 Command Examples

```
get ver Encoder1<cr>
get ver key:abc123 Encoder1<cr>
```

9.17.7 Return Examples

```
get ver success 3.5.2.0 <cr>

get ver error [incomplete]<cr>
get ver error [device 'Encoder1' not found]<cr>
```

9.18 Command get video

9.18.1 Command usage

get video **[**key:<*security_key*>**]** <*encoder_device_name*> <*option*><cr>

9.18.2 Description

The command **get video** is used to retrieve the connected video information from an Encoder.

9.18.3 Arguments

encoder_device_name	Device name of the Encoder
option	'all' / 'width' / 'height' / 'fps' / 'cs' / 'bpp' / 'sm'

9.18.4 Notes

- **all** returns <*width*> <*height*> <*frames_per_second*> <*color_space*> <*bits_per_pixel*> <*scan_mode*>

9.18.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.18.6 Command Example

```
get video Encoder1 all<cr>
get video Encoder1 width<cr>
get video Encoder1 height<cr>
get video Encoder1 fps<cr>
get video Encoder1 cs<cr>
get video Encoder1 bpp<cr>
get video Encoder1 sm<cr>
get video key:abc123 Encoder1 all<cr>
```

9.18.7 Return Examples

```
get video success 1920 1080 60 YCBCR_444 8 PROGRESSIVE<cr>
get video success 1920<cr>
get video success 1080<cr>
get video success 60<cr>
get video success RGB<cr>
get video success YCBCR_444<cr>
get video success YCBCR_422<cr>
get video success YCBCR_420<cr>
get video success 8<cr> *can also be 10 or 12
get video success PROGRESSIVE<cr>
get video success INTERLACED <cr>

get video error [incomplete]<cr>
get video error [not supported]<cr>
get video error [encoder 'Encoder1' not found]<cr>
```

ToC

9.19 Command get video_compress

9.19.1 Command usage

get video_compress **[**key:<*security_key*>**]** <*encoder_device_name*><cr>

9.19.2 Description

The command **get video_compress** is used to retrieve the video compression status of the specified Encoder.

9.19.3 Arguments

encoder_device_name	Device name of the Encoder

9.19.4 Notes

- Use the command **set video_compress** to turn it on or off.

9.19.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video_compress	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.19.6 Command Example

```
get video_ compress Encoder1<cr>
get video_ compress key:abc123 Encoder1<cr>
```

9.19.7 Return Examples

```
get video_ compress success true<cr>
get video_ compress success false<cr>

get video_ compress error [incomplete]<cr>
get video_ compress error [not supported]<cr>
get video_ compress error [encoder 'Encoder1' not found]<cr>
```

ToC

9.20 Command get video_mode

9.20.1 Command usage

get video_mode **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.20.2 Description

The command **get video_mode** is used to retrieve current Decoder mode of operation.

9.20.3 Arguments

decoder_device_name	Device name of Decoder

9.20.4 Notes

9.20.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video_mode	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.20.6 Command Example

```
get video_mode Decoder1<cr>
get video_mode key:abc123 Decoder1<cr>
```

9.20.7 Return Example

```
get video_mode success Sync<cr>
get video_mode success Sync Scale<cr>
get video_mode success Fast (keep)<cr>
get video_mode success Fast (stretch)<cr>
get video_mode success Fast (crop)<cr>
get video_mode success Multiview<cr>
get video_mode success Wall (sync)<cr>
get video_mode success Wall (fast)<cr>

get video_mode error [incomplete]<cr>
get video_mode error [not supported]<cr>
get video_mode error [decoder 'Decoder1' not found]<cr>
```

ToC

9.21 Command get video_mute

9.21.1 Command usage

get video_mute **[**key:<*security_key*>**]** <*decoder_device_name*><cr>

9.21.2 Description

The command **get video_mute** is used to retrieve the video mute status of the specified Decoder.

9.21.3 Arguments

decoder_device_name	Device name of the Decoder

9.21.4 Notes

- Use the command **set video_mute** to turn it on and off or change the color of the muted display.

9.21.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video_mute	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.21.6 Command Example

```
get video_mute Decoder1<cr>
get video_mute key:abc123 Decoder1<cr>
```

9.21.7 Return Examples

```
get video_mute success true<cr>
get video_mute success false<cr>

get video_mute error [incomplete]<cr>
get video_mute error [not supported]<cr>
get video_mute error [decoder 'Decoder1' not found]<cr>
```

ToC

9.22 Command get video_source

9.22.1 Command usage

get video_source **[**key:<*security_key*>**]** <*encoder_device_name*><cr>

9.22.2 Description

The command **get video_source** is used to retrieve the video source input of an Encoder.

9.22.3 Arguments

encoder_device_name	Device name of the Encoder

9.22.4 Notes

- Returned mode is either **hdmi**, **dp** or **auto.**
- Use the command **set video_source** to change this setting.

9.22.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video_source	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.22.6 Command Example

```
get video_source Encoder1<cr>
get video_source key:abc123 Encoder1<cr>
```

9.22.7 Return Examples

```
get video_source success hdmi<cr>
get video_source success dp<cr>
get video_source success auto<cr>

get video_source error [incomplete]<cr>
get video_source error [not supported]<cr>
get video_source error [encoder 'Encoder1' not found]<cr>
```

ToC

9.23 Command get video_status

9.23.1 Command usage

get video_status **[**key:<*security_key*>**]** <*encoder_device_name*><cr>

9.23.2 Description

The command **get video_status** is used to find if an Encoder has a stable video source connected.

9.23.3 Arguments

encoder_device_name	Device name of the Encoder

9.23.4 Notes

- Returned mode is either **true** or **false.**

9.23.5 Return Value

command	<space>	**mode**	<space>	**status**	<space>	**value**	**terminator**
get	<space>	video_status	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.23.6 Command Example

```
get video_status Encoder1<cr>
get video_status key:abc123 Encoder1<cr>
```

9.23.7 Return Examples

```
get video_status success true<cr>
get video_status success false<cr>

get video_status error [incomplete]<cr>
get audio_status error [not supported]<cr>
get video_status error [encoder 'Encoder1' not found]<cr>
```

ToC

9.24 Command get window

9.24.1 Command usage

get window **[**key:<*security_key*>**]** <*layout_name*> <*index*> <*option*><cr>

9.24.2 Description

The command **get window** is used to retrieve a window size of a multiview layout.

9.24.3 Arguments

layout_name	Name of the multiview layout
index	Index of the layout window 0-31
option	'all' / 'width' / 'height'

9.24.4 Notes

- The layout must already be executed by the SDVoE Director Controller which is then held in memory.

9.24.5 Return Value

command	<space>	mode	<space>	status	<space>	value	terminator
get	<space>	window	<space>	*success data / error [message]*	<space>	<string>	<cr>

9.24.6 Command Example

```
get window MyLayout 3 all<cr>
get window MyLayout 0 width<cr>
get window MyLayout 1 height<cr>
get window key:abc123 MyLayout 0 all<cr>
```

9.24.7 Return Examples

```
get window success 1920 1080<cr>
get window success 1920<cr>
get window success 1080<cr>

get window error [incomplete]<cr>
get window error [not supported]<cr>
get window error [layout 'MyLayout' not found]<cr>
get window error [index '0' not found]<cr>
```

10 Command send

The **send** command is used to send either infrared or serial RS-232 data to any or all Encoders or Decoders from a 3rd party control system.

10.1 Command send infrared

10.1.1 Command usage

send ir **[**key:<*security_key*>**]** <*device_name*> <*data_hex*><cr>

10.1.2 Description

The command **send ir** is used to send infrared (IR) signals from a control system to Encoders and Decoders.

10.1.3 Arguments

device_name	Device name of the Decoder, Encoder or '**all**', '**all_rx**', '**all_tx**'
data_hex	String of ascii characters representing the hexadecimal Pronto infrared code

10.1.4 Notes

- The **data_hex** argument is a hexadecimal string which represent the Pronto infrared code to be sent.
- Its length must be a multiple of eight (i.e. data length must be a multiple of four bytes) and cannot exceed 256 burst pairs and a maximum length of 1032 bytes.

10.1.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
send	<space>	ir	<space>	*success / error*	<cr>

10.1.6 Command Examples

```
send ir Decoder1 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015
00150015001500400015004000150040001500150015001500150015001500150015001500150015001500150015004000150015001500150015001500400
01500400015001500150015001500400015001500150015004000150040001500150015001500150015004000150040001500150015001506
89<cr>

send ir key:abc123 all_rx 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001
500150015001500150015004000150040001500400015001500150015001500150015001500150015001500150015001500150015004000150015001500150015
00150040001500400015001500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150
01500150689<cr>
```

ToC

10.1.7 Return Examples

```
send ir success<cr>

send ir error [incomplete]<cr>
send ir error [max length exceeded]<cr>
send ir error [Length of hex data should be in multiples of 4 bytes]<cr>
send ir error [Length of infrared data must be a multiple of 32 bits]<cr>
send ir error [device 'Encoder1' not found]<cr>
send ir error [not supported]<cr>
```

ToC

10.2 Command send serial

10.2.1 Command usage

send serial **[**key:<*security_key*>**]** <*device_name*> **[**reply <*timeout*>**]** <*data_string*><cr>

10.2.2 Description

The command **send serial** is used to send serial RS-232 data from a control system to Encoders and Decoders.

10.2.3 Arguments

device_name	Device name of Encoder, Decoder, Group or '**all**', '**all_rx**', '**all_tx**'
reply	Keyword '**reply**' to wait for a reply (optional)
timeout	Timeout period of reply data to be received up to 9 (used with reply)
data_string	String of ascii characters

10.2.4 Notes

- When either 2-way communication is required and the control system is expecting a reply from the serial equipment or unsolicited serial data is expected, then the **join serial** command is required to join the device with the SDVoE Director Controller '**join serial** *Encoder1* **api**'. When unsolicited serial data is received the SDVoE Director Controller will raise a 'notify serial' event.

- When **reply** is present and **timeout** is not specified, a default 3 seconds will be applied as the maximum wait time for a reply to be received before the command returns.

- The data_string argument is a printable ASCII text string of any length. 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 null character
  - \n line feed
  - \r carriage return
  - \t horizontal tab
  - \xnn (where nn are two hexadecimal characters) hexadecimal representation of byte
  - \\ a single backslash
  - \ (space) space character

ToC

10.2.5 Return Value

command	<space>	mode	<space>	status	terminator
send	<space>	serial	<space>	*success [data] / error [message]*	<cr>

10.2.6 Command Examples

```
send serial Decoder1 reply my data string\r<cr>
send serial Decoder1 reply 2 my data string\r\n<cr>
send serial Encoder1 \x00\x01\x02\x03\x04<cr>
send serial key:abc123 Decoder1 my data string\r<cr>
```

10.2.7 Return Examples

```
send serial success [my return string]<cr>
send serial success []<cr>
send serial success<cr>

send serial error [incomplete]<cr>
send serial error [data expected]<cr>
send serial error [not supported]<cr>
send serial error [device 'Encoder1' not found]<cr>
```

ToC

10.3 Command send cec

10.3.1 Command usage

send cec **[**key:<*security_key*>**]** <*device_name*> <*data_hex*><cr>

10.3.2 Description

The command **send cec** is used to send cec data from a control system to a Decoder's connected display.

10.3.3 Arguments

device_name	Device name of the Decoder, Encoder or '**all**', '**all_tx**', '**all_rx**'
data_hex	String of ascii characters representing the hexadecimal cec code

10.3.4 Notes

- The **data_hex** argument is a hexadecimal string which represent the cec code to be sent.

10.3.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
send	<space>	cec	<space>	*success / error*	<cr>

10.3.6 Command Examples

```
send cec Decoder1 F004<cr>
send cec key:abc123 Decoder1 F004<cr>
```

10.3.7 Return Examples

```
send cec success<cr>

send cec error [incomplete]<cr>
send cec error [decoder 'Decoder1' not found]<cr>
send cec error [This device does not support CEC]<cr>
```

10.4 Command send gc

The command **send gc** provides a seamless integration with Global Cachè products. For more information on Global Cachè visit www.globalcache.com

10.4.1 Command usage

send gc **[**key:<*security_key*>**]** <*address*> <*port*> <*gc_api*><cr>

10.4.2 Description

The command send **gc** allows control over numerous Global Cachè products via the same single TCP connection used for the SDVoE Director Controller.

10.4.3 Arguments

address	Global Cachè IP address
port	Global Cachè TCP port. Usually 4998 and for serial com1: 4999 com2:5000
gc_api	Global Cachè API string

10.4.4 Notes

- The **gc_api** string uses the same standard Global Cachè control string format as found in the Global Cachè API manuals downloaded from here: www.globalcache.com/downloads/

10.4.5 Return Value

Return values will be in the standard Global Cachè format under normal conditions. An exception to this would be if a TCP connection was not possible to a device, in which case an error such as send gc error [device '172.30.1.111' not found]<cr> would be sent.

10.4.6 Examples

```
send gc 172.30.1.111 4999 a string to send<cr>
send gc 172.30.1.111 4998 setstate,1:1,1<cr>
send gc 172.30.1.111 4998 getstate,1:1<cr>
send gc 172.30.1.111 4998 sendir,1:2,4444,34500,1,1,34,48,24,12,24,960,24,12,24...<cr>
send gc key:abc123 172.30.1.111 4999 a string to send<cr>
```

10.4.7 Return Examples

```
a serial string<cr>
state,1:1,1<cr>
completeir,1:1,1<cr>

send gc error [incomplete]<cr>
send gc error [device '172.30.1.111' not found]<cr>
send gc error [device '172.30.1.111' timeout]<cr>
send gc error [invalid]<cr>
ERR_<XX><cr>
```

ToC

10.5 Command send tcp

10.5.1 Command usage

send tcp **[**key:<*security_key*>**]** <*address*> <*port*> <*command*><cr>

10.5.2 Description

The command **send tcp** provides a seamless integration with any TCP controllable device.

10.5.3 Arguments

address	TCP IP address
port	TCP port
command	Command string to send to TCP device

10.5.4 Notes

- The TCP device must be in the same range as the SDVoE Director Controller.
- To send HEX add \x before the HEX byte. \x0D for carriage return

10.5.5 Return Value

Return values will be what is returned from the TCP device.

10.5.6 Examples

```
send tcp 172.30.1.111 1000 an ascii string to send<cr>
send tcp 172.30.1.111 1000 an mixed string to send\x0D<cr>
send tcp 172.30.1.111 1000 \x00\x01\x02\x03<cr>
send tcp key:abc123 172.30.1.111 1000 an ascii string to send<cr>
```

10.5.7 Return Examples

```
send tcp error [incomplete]<cr>
send tcp error [device '172.30.1.111' not found]<cr>
```

11 Commands for Multiview

This section covers all the supported multiview commands.

11.1 Command multiview

11.1.1 Command usage

multiview **[**key:<*security_key*>**]** <*decoder_device_name*> <*name*> <*h_size*> <*v_size*> <*fps*><cr>

11.1.2 Description

The command **multiview** is used to activate the multiview layout in a specified Decoder as defined with the command **layout**. The size of the layout can be scaled to fit different display resolutions.

11.1.3 Arguments

decoder_device_name	Device name of the Decoder
name	The name given to the multiview layout
h_size	Scaled output width resolution of layout
v_size	Scaled output height resolution of layout
fps	The required refresh rate of the multiview display

11.1.4 Notes

- A layout must be created before using this command.

11.1.5 Return Value

command	<space>	**status**	**terminator**
multiview	<space>	*success / error [message]*	<cr>

11.1.6 Examples

```
multiview Decoder1 MV_e17 3840 2160 30<cr>
multiview Decoder2 MyLayout 1920 1080 60<cr>
multiview key:abc123 Decoder1 MV_e17 3840 2160 30<cr>
```

11.1.7 Return Examples

```
multiview success<cr>

multiview error [incomplete]<cr>
multiview error [layout 'MyLayout' not found]<cr>
multiview error [decoder 'Decoder1' not found]<cr>
```

ToC

11.2 Command layout

The **layout** commands are used to define a multiview layout.

11.2.1 Command layout new

11.2.1.1 Command usage

layout new **[**key:<*security_key*>**]** <*layout_name*> <*width*> <*height*><cr>

11.2.1.2 Description

The **layout new** command is used to initially create a multiview layout or reset an existing layout with the specified name.

Once the multiview layout has been created with this **layout new** command, use the **layout window** command to define the windows within the multiview layout.

11.2.1.3 Arguments

layout_name	The name given to the multiview layout
width	Total horizontal size of the layout in pixels
height	Total vertical size of the layout in pixels

11.2.1.4 Notes

- The **width** argument must be a multiple of 2 pixels

11.2.1.5 Return Value

command	<space>	**status**	**terminator**
layout	<space>	*success / error [message]*	<cr>

11.2.1.6 Command Examples

```
layout new MyLayout 3840 2160<cr>
layout new MyLayout 1920 1080<cr>
layout new key:abc123 MyLayout 3840 2160<cr>
```

11.2.1.7 Return Examples

```
layout new success<cr>

layout new error [incomplete]<cr>
```

ToC

11.2.2 Command layout window

11.2.2.1 Command usage

layout window **[**key:<*security_key*>**]** <*layout_name*> <*index*> <*horiz_position*> <*vert_position*> <*width*> <*height*> <*subscription*><cr>

11.2.2.2 Description

The **layout window** command is used to define the window size and location in the multiview display. Once the multiview layout has been created with the **layout new** command, use this **layout window** command to define the windows within the multiview layout.

11.2.2.3 Arguments

layout_name	The name given to the multiview layout
index	Index of the window, between 0 and 31
horiz_position	Horizontal position in pixels
vert_position	Vertical position in pixels
width	Horizontal size in pixels
height	Vertical size in pixels
subscription	Decoder index to be used with a join command, between 0 and 31

11.2.2.4 Notes

- The **horiz_position** and **width** arguments must be a multiple of 32 pixels.
- Multiple windows can have the same subscription therefore the same video content.
- Windows can overlap, in which case the window with the lowest **window_index** will be on top.

11.2.2.5 Return Value

command	<space>	**status**	**terminator**
layout	<space>	*success / error [message]*	<cr>

11.2.2.6 Examples

```
layout window MyLayout 1 0 0 1920 1080 0<cr>
layout window MyLayout 0 1920 1080 1920 1080 1<cr>
layout window key:abc123 MyLayout 1 0 0 960 480 0<cr>
```

11.2.2.7 Return Examples

```
layout window success<cr>

layout window error [incomplete]<cr>
layout window error [layout 'MyLayout' not found]<cr>
```

ToC

11.2.3 Command layout black

11.2.3.1 Command usage

layout black **[**key:<*security_key*>**]** <*layout_name*> <*index*><cr>

11.2.3.2 Description

The **layout black** command is used to define a window in the multiview layout as black.

11.2.3.3 Arguments

layout_name	The name given to the multiview layout
index	Index of the window, which must be between 0 and 31

11.2.3.4 Notes

11.2.3.5 Return Value

command	<space>	**status**	**terminator**
layout	<space>	*success / error [message]*	<cr>

11.2.3.6 Command Examples

```
layout black MyLayout 0<cr>
layout black key:abc123 MyLayout 0<cr>
```

11.2.3.7 Command Examples

```
layout black success<cr>

layout black error [incomplete]<cr>
layout black error [layout 'MyLayout' not found]<cr>
```

11.2.4 Command layout delete

11.2.4.1 Command usage

layout delete **[**key:<*security_key*>**]** <*layout_name*><cr>

11.2.4.2 Description

The **layout delete** command is used to delete an existing layout from the SDVoE Director Controller.

11.2.4.3 Arguments

layout_name	The name given to the multiview layout

11.2.4.4 Notes

11.2.4.5 Return Value

command	<space>	**status**	**terminator**
layout	<space>	*success / error [message]*	<cr>

11.2.4.6 Command Examples

```
layout delete MyLayout<cr>
layout delete key:abc123 MyLayout<cr>
```

11.2.4.7 Command Examples

```
layout delete success<cr>

layout delete error [incomplete]<cr>
layout delete error [layout 'MyLayout' not found]<cr>
```

ToC

12 Message notify

The **notify** messages are sent from the SDVoE Director Controller to a third party control system connected on TCP port 6980 with updated event notifications. A **notify** message will be sent on the following events:

- Serial RS-232 data received from Encoder or Decoder (when joined to the API)
- Encoder or Decoder network connectivity (always)
- Decoder display connectivity (always)
- Encoder source connectivity (always)

12.1 Message notify serial

12.1.1 Message received

notify serial <*device_name*> <*data_string*><cr>

12.1.2 Description

A **notify serial** message is sent when serial RS-232 data from an Encoder or Decoder is received.

12.1.3 Arguments

device_name	Device name of Encoder or Decoder
data_string	String of ascii characters

12.1.4 Notes

- Before **notify serial** messaged can be received, the join serial command must be used to join the device to the SDVoE Director Controller. *Refer to 4.8 'Command join serial'.*

12.1.5 Return Value

message	<space>	**mode**	<space>	**data_string**	**terminator**
notify	<space>	serial	<space>	*ascii_data*	<cr>

12.1.6 Examples

```
notify serial 'Decoder1' my data string\\r<cr>
notify serial 'Decoder1' \\x00\\x01\\x02\\x03\\x04<cr>
```

ToC

12.2 Message notify network

12.2.1 Message received

notify network <*device_name*> <*state*><cr>

12.2.2 Description

A **notify network** message is sent whenever an Encoder or Decoder is connected or disconnected from the network.

12.2.3 Arguments

device_name	Device name of Encoder or Decoder
state	'true' / 'false'

12.2.4 Notes

- A **notify network** message will be sent when the SDVoE Director Controller is unable to connect or connects with an Encoder or Decoder. For example, a false then true message will be sent during a device power cycle or reboot.

12.2.5 Return Value

message	<space>	**mode**	<space>	**status**	**terminator**
notify	<space>	network	<space>	*true / false*	<cr>

12.2.6 Examples

```
notify network 'Decoder1' false<cr>
notify network 'Decoder1' true<cr>
```

ToC

12.3 Message notify display

12.3.1 Message received

notify display <decoder_*device_name*> <*state*><cr>

12.3.2 Description

A **notify display** message is sent whenever a display is connected or disconnected from a Decoder.

12.3.3 Arguments

decoder_device_name	Device name of Decoder
state	'true' / 'false'

12.3.4 Notes

- Some non-compliant displays will require power for this event to be raised.

12.3.5 Return Value

message	<space>	**mode**	<space>	**status**	**terminator**
notify	<space>	display	<space>	*true / false*	<cr>

12.3.6 Examples

```
notify display 'Decoder1' false<cr>
notify display 'Decoder1' true<cr>
```

12.4 Message notify source

12.4.1 Message received

notify source <encoder_*device_name*> <*state*><cr>

12.4.2 Description

A **notify source** message is sent whenever a source is connected or disconnected from an Encoder.

12.4.3 Arguments

encoder_device_name	Device name of Encoder
state	'true' / 'false'

12.4.4 Notes

- The SDVoE Director Controller is looking for a stable video signal.

12.4.5 Return Value

message	<space>	**mode**	<space>	**status**	**terminator**
notify	<space>	source	<space>	*true / false*	<cr>

12.4.6 Examples

```
notify source 'Encoder1' false<cr>
notify source 'Encoder1' true<cr>
```

ToC

13 Command preset

The preset commands are used to store and apply a series of commands available from this manual.
A sequence of commands can be used to create routing tables, video wall or multiview layouts.
Refer Appendix E - Preset logic for logic that can be applied within a preset.

13.1 Command preset add

13.1.1 Command usage

preset add **[**key:<*security_key*>**]** <*preset_name*> <*preset_data*><cr>

13.1.2 Description

The command **preset add** is used to create and append commands to a specified preset.

13.1.3 Arguments

preset_name	The name defined as the preset
preset_data	A valid Control Command string

13.1.4 Notes

- The preset is executed with the **preset load** command.
- Preset commands are not allowed in a preset itself, only used to create, delete and execute presets.

13.1.5 Return Value

command	<space>	**function**	<space>	**name**	<space>	**status**	**terminator**
preset	<space>	add	<space>	*preset1*	<space>	*success / error [message]*	<cr>

13.1.6 Command Examples

```
preset add preset1 layout new MV_e1 3840 2160<cr>
preset add preset1 layout window MV_e1 0 0 0 3840 2160 0<cr>
preset add preset1 multiview Decoder1 MV_e1 30<cr>
preset add key:abc123 preset1 join multi Encoder1 Decoder1 0 false<cr>
```

13.1.7 Return Examples

```
preset add preset1 success<cr>

preset add error [incomplete]<cr>
```

ToC

13.2 Command preset load

13.2.1 Command usage

preset load **[**key:<*security_key*>**]** <*preset_name*><cr>

13.2.2 Description

The command **preset load** is used to apply stored commands within the specified preset.

13.2.3 Arguments

preset_name	The name defined as the preset

13.2.4 Notes

- The preset is created with the **preset add** command or directly via the UI.
- Presets are stored in the SDVoE Director Controller.

13.2.5 Return Value

command	<space>	**function**	<space>	**name**	<space>	**status**	**terminator**
preset	<space>	load	<space>	*preset1*	<space>	*success / error [message]*	<cr>

13.2.6 Command Examples

```
preset load preset1<cr>
preset load key:abc123 preset1<cr>
```

13.2.7 Return Examples

```
preset load preset1 success<cr>

preset load error [incomplete]<cr>
preset load error [preset 'preset1' not found]<cr>
preset load preset1 error […]<cr>
```

13.3 Command preset delete

13.3.1 Command usage

preset delete **[**key:<*security_key*>**]** <*preset_name*><cr>

13.3.2 Description

The command **preset delete** is used to delete the specified preset from the SDVoE Director Controller or directly from the UI.

13.3.3 Arguments

preset_name	The name defined as the preset

13.3.4 Notes

13.3.5 Return Value

command	<space>	**function**	<space>	**name**	<space>	**status**	**terminator**
preset	<space>	delete	<space>	*preset1*	<space>	*success / error [message]*	<cr>

13.3.6 Command Examples

```
preset delete preset1<cr>
preset delete MyMultiviewSetting<cr>
preset delete key:abc123 preset1<cr>
```

13.3.7 Return Examples

```
preset delete preset1 success<cr>

preset delete error [incomplete]<cr>
preset delete error [preset 'preset1' not found]<cr>
preset error [invalid mode]<cr>
```

ToC

13.4 Command preset delay

13.4.1 Command usage

preset delay **[**key:<*security_key*>**]** <*sec*>

13.4.2 Description

The command **preset delay** is used within a preset to add a delay between commands.

13.4.3 Arguments

sec	Delay time in milliseconds up to 9999

13.4.4 Notes

- This command can only be used within a preset.

13.4.5 Return Value

None

13.4.6 Command Examples

```
preset delay 1000
preset delay key:abc123 500
```

13.4.7 Return Example

None

ToC

13.5 Command preset wait

13.5.1 Command usage

preset wait **[**key:<*security_key*>**]**

13.5.2 Description

The command **preset wait** is used within a preset to pause the execution of commands until processing of all previous commands is completed.

13.5.3 Arguments

13.5.4 Notes

- This command can only be used within a preset.

13.5.5 Return Value

None

13.5.6 Command Example

```
preset wait
preset wait key:abc123
```

13.5.7 Return Example

None

ToC

13.6 Command preset dynamic

13.6.1 Command usage

preset dynamic **[**key:<*security_key*>**]** <*preset_name*> <*state*><cr>

13.6.2 Description

The command **preset dynamic** is intended to be used with video wall presets. When set to true the preset will be applied on a notification source true from the Encoder. So if the source changes resolution the wall preset is applied again to crop with the correct sizes to match the wall layout.

13.6.3 Arguments

preset_name	The name defined as the preset
state	'true' / 'false'

13.6.4 Notes

- When the video wall preset is no longer active, you must issue a preset dynamic <encoder> false command to prevent the wall preset from being applied each time the Encoder is connected to a source.
- All active dynamic presets for a given Encoder will automatically be disabled when any of the following commands are used on an active Encoder:
  join fast, join sync, join sync_scale, join adv, join wall, join walladv, stop video, stop av, reboot & reset

13.6.5 Return Value

command	<space>	**function**	<space>	**name**	<space>	**status**	**terminator**
preset	<space>	dynamic	<space>	*preset1*	<space>	*success / error [message]*	<cr>

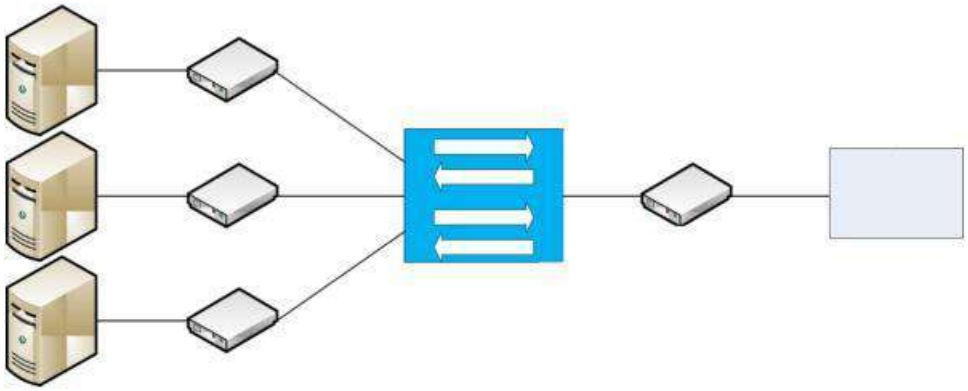
13.6.6 Command Examples

```
preset dynamic preset1 true<cr>
preset dynamic preset1 false<cr>
preset dynamic key:abc123 preset1 true<cr>
```

13.6.7 Return Examples

```
preset dynamic preset1 success<cr>

preset dynamic error [incomplete]<cr>
preset dynamic error [preset 'preset1' not found]<cr>
```

Appendix A - How to Multiview



Multiview is the ability to take multiple independent video sources and compose them onto a single output. Sources can have different resolutions and frame rates. Any multiviewer application has to scale the various source signals, place them into a predefined layout of certain dimension, compose the scene and then generate a video signal of certain resolution and frame rate that is sent out to a connected display.

The key to successfully implementing multiviewer layouts is to correctly manage network bandwidth; both at the Decoder side where multiple streams need to be received to create the multiviewer composition as well as on the Encoder side, where both the main and the multiview scaled down sub stream has to coexist.

In short, creating multiview layouts is all about managing network bandwidth and comes down to the following two rules:

- Bandwidth of all video streams needed for a particular multiview layout along with other signals going to a Decoder has to be under 10Gb/s.

- The combined bandwidth of the main stream, multiview down scaled sub stream and other signals going out of any Encoder has to be under 10Gb/s.

ToC

Appendix A - How to Multiview continued…

In multiviewer mode, the display is divided into a grid consisting of tiles. Each tile displays video from a rectangular area inside a Decoder's frame buffer. Alternatively, a tile can also be black. One or more tiles form a window displaying a source. Before going further, it is important to understand the concepts that are used to define and program multiview layouts.

- A **surface** is a rectangular area within the Decoder's frame buffer that is reserved for a specific video source. A Decoder 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 **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 contains a rectangular area of a surface or is black. Windows can overlap, in which case the content of the window with the smallest index is shown. Multiple tiles may be required to represent each window (when windows overlap) and the black areas that are not part of any window.

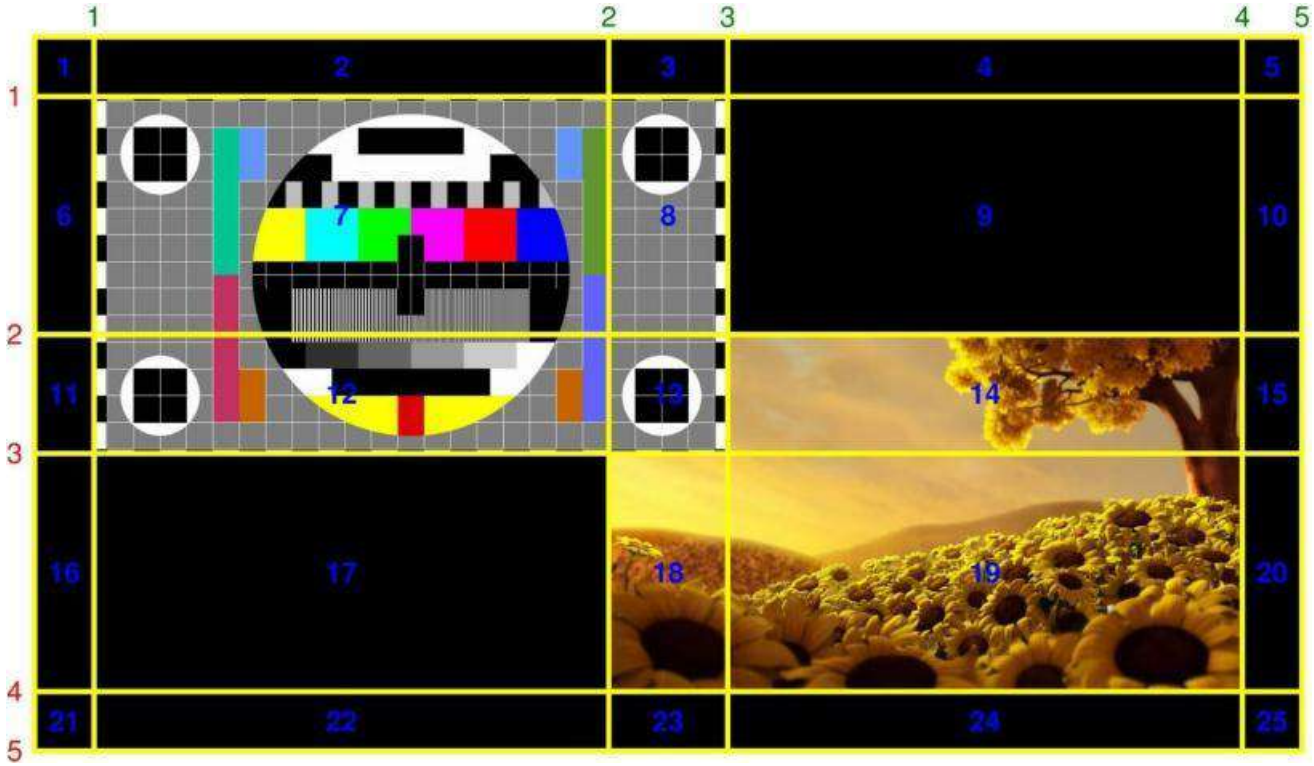
- A **layout** represents the full set of multiview configuration for a Decoder. A layout has a name that is used to refer to it by the various commands, and an output resolution. In addition, layouts defined using API commands have a set of surfaces and a set of windows.

- The following limitations apply:
    - The total number of tiles cannot exceed 240
    - Number of horizontal or vertical tile borders cannot exceed 16
    - The largest possible tile grid is either 16x15 or 15x16
    - Tile widths must be a multiply of 32 pixels
    - Surfaces have to be horizontally separated in frame buffer memory by 32 pixels margin
    - The number of distinct video sources cannot exceed 32
    - Number of windows cannot exceed 32
    - The total bandwidth sent by an individual transmitter and received by an individual receiver must never exceed 10Gbit/s and for all practical purposes should never exceed 8.5 Gb/s to allow room for Gigabit Ethernet, USB, Audio and control (IR and RS-232)
    - Video streams used by multiviewer must be progressive RGB 8 bits or YUV 4:4:4 bits

ToC

Appendix A - How to Multiview continued…

The figure below shows a layout with two windows: the 'Test Pattern' window and the 'Sun Flower' window. The window locations define the horizontal (green numbers) and vertical (red numbers) transitions in the scene which in turn define the tiles (blue numbers) This particular layout has 5x5 tile grid or 25 tiles total. Four tiles form the 'Test Pattern' window (tiles 7, 8, 12 and 13), three tiles form the visible portion of the 'Sun Flower' window (tiles 14, 18 and 19). All other tiles are black and part of the background. The 'Test Pattern' window has a lower index than the 'Sun Flower' window because it is on top (for example the 'Test Pattern' window could be window #0 while the 'Sun Flower' window could be window #1). The 'Test Pattern' window refers to a surface with the 'Test Pattern' video and the 'Sun Flower' window refers to a surface that has the 'Sun Flower' video.



Layout with two video sources.
Because of the windows overlapping, the total number of tiles used in this layout is 25; (5x5) tile grid.

ToC

Appendix A - How to Multiview continued…

To calculate approximate maximum network bandwidth for a given video resolution, the following formulas can be used:

LineFq (KHz) = Ver_res_total x fps
Network BW (Gbps) = Hor_res x Ebpp x LineFq x 1.05 / 1000000

- LineFq: Video line Frequency in Khz
- Ver_res_total: Total number of lines including vertical blanking
- Hor_Res: Horizontal pixel resolutions
- Ebpp: Effective bits per pixel. For example RGB 8 bit has 24 effective bits per pixel. Meanwhile YUV 420 8 bit has 12 effective bits per pixel while YUV 422 8 bit has 16 effective bits per pixel. Similarly, RGB or YUV 444 10 bit has 30 effective bits per pixel while YUV 420 10 bits has 15 effective bits/pixel while YUV 422 has 20 effective bits/pixel.
- 1.05: Scaling factor to take into account approximate network overhead based on packet overhead, packet payload size, packet preamble and minimum inter packet gap.
- 1,000,000: Scaling factor to bring network Bandwidth to Gbps unit

Here are two examples to calculate network bandwidth for two different resolutions:

- **Resolution 1920x1080 60Hz RGB 8 bit**
  LineFq = 1120 lines * 60 Hz / 1000 = 67.5
  NetBW = 1920 * 24 * 67.5 * 1.05 = 3.27 Gbps

- **Resolution 3840x2160 30Hz YUV 8 bit**
  LineFq = 2250 lines * 30 Hz / 1000 = 67.5
  NetBW = 3840 * 24 * 67.5 * 1.05 = 6.53 Gbps

The table below shows approximate maximum network bandwidth for a few select resolutions. To calculate approximate maximum network bandwidth for other resolutions, use the formulas above.

RESOLUTION	Hvisible	Htotal	Vvisable	Vtotal	colour format	format	colour depth	Ebpp	Line (KHz)	Frame (Hz)	BW line	Network Bandwidth
720P	1280	1648	720	750	RGB	888	24	24	45	60	1.382	**1.451**
1080i	1920	2200	1080	1125	RGB	888	24	24	33.75	30	1.555	**1.632**
1080P	1920	2200	1080	1125	RGB	888	24	24	67.5	60	3.110	**3.265**
	1920	2200	1080	1125	YUV	422	24	16	67.5	60	2.073	**2.177**
	1920	2200	1080	1125	YUV	420	24	12	67.5	60	1.555	**1.632**
UHD 24	3840	5500	2160	2250	RGB	888	24	24	67.5	24	4.976	**5.225**
UHD 30	3840	5500	2160	2250	RGB	888	24	24	67.5	30	6.220	**6.531**
	3840	5500	2160	2250	YUV	422	24	16	67.5	30	4.147	**4.354**
	3840	5500	2160	2250	YUV	420	24	12	67.5	30	3.110	**3.265**
UHD 60	3840	5500	2160	2250	RGB	888	24	24	135	60	12.441	**13.063**
	3840	5500	2160	2250	YUV	422	24	16	135	60	8.294	**8.709**
	3840	5500	2160	2250	YUV	420	24	12	135	60	6.220	**6.531**
	3840	5500	2160	2250	YUV	422	30	20	135	60	10.368	**10.886**
	3840	5500	2160	2250	YUV	420	30	15	135	60	7.776	**8.165**

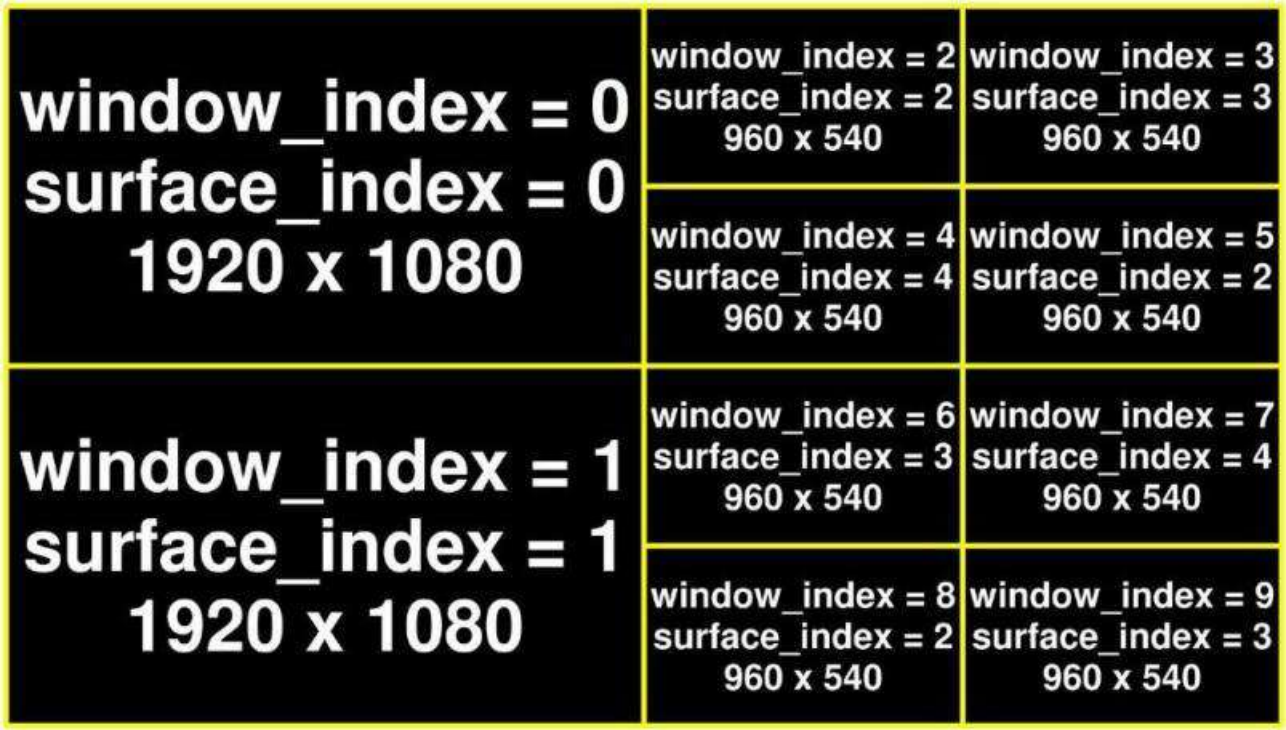
Approximate maximum network bandwidth for several common video resolutions

ToC

Appendix A - How to Multiview continued…

The table below shows available multiviewer commands:

COMMAND	DESCRIPTION	REFERENCE
set scaler	Used to set the resolution of the multiview sub stream	8.8
set frame_converter	Used to half the frame rate of the Encoder video stream	8.5
multiview	Used once to activate the layout	11.1
layout new	Used once to define the layout name and size	11.2.1
layout window	Used multiple times to define the window sizes and locations	11.2.2
layout black	Used to apply a black area to the layout	11.2.3
layout delete	Used to delete an existing layout	11.2.4
join multi	Used multiple times to join Encoders to layout windows	4.10
leave sub	Used to leave a video stream subscription	5.2
stop video	Used to stop the main stream and reduce Encoder bandwidth	6.1
stop sub	Used to stop the multiview sub stream	6.2



Examples used to create the above scene:

ToC

Example 1: (recommended)

Shows the use of specifying **layout_name** in **join multi** and not requiring the **set scaler** command.

```
layout new MyLayout 3840 2160<cr>
layout window MyLayout 0 0 0 1920 1080 0<cr>
layout window MyLayout 1 0 1080 1920 1080 1<cr>
layout window MyLayout 2 1920 0 960 540 2<cr>
layout window MyLayout 3 2880 0 960 540 3<cr>
layout window MyLayout 4 1920 540 960 540 4<cr>
layout window MyLayout 5 2880 540 960 540 2<cr>
layout window MyLayout 6 1920 1080 960 540 3<cr>
layout window MyLayout 7 2880 1080 960 540 4<cr>
layout window MyLayout 8 1920 1620 960 540 2<cr>
layout window MyLayout 9 2880 1620 960 540 3<cr>
multiview Decoder1 MyLayout 3840 2160 60<cr>
join multi Encoder1 Decoder1 0 scaled MyLayout<cr>
join multi Encoder2 Decoder1 1 scaled MyLayout<cr>
join multi Encoder3 Decoder1 2 scaled MyLayout<cr>
join multi Encoder4 Decoder1 3 scaled MyLayout<cr>
join multi Encoder5 Decoder1 4 scaled MyLayout<cr>
```

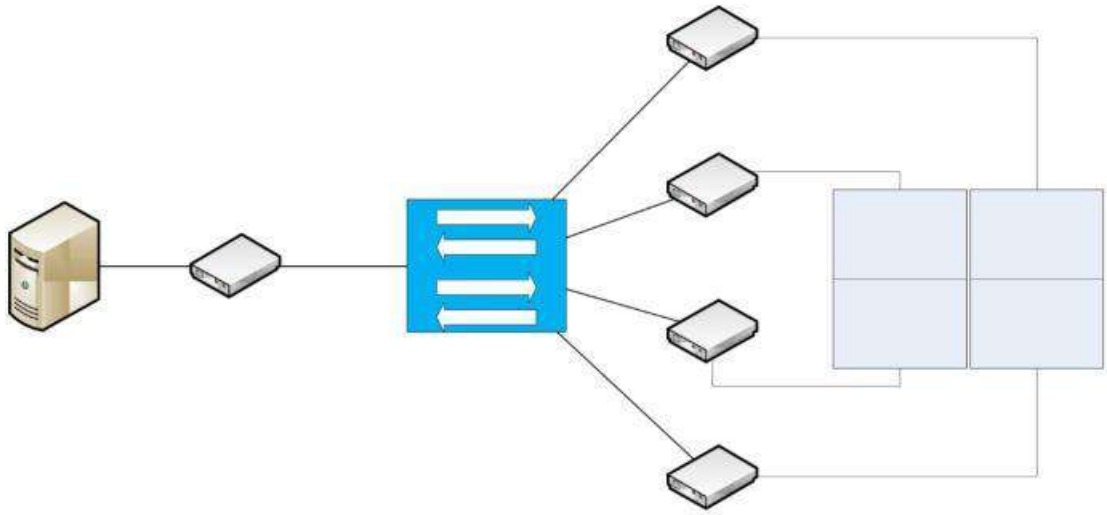
Example 2:

Shows the **layout_name** in the **join multi** command being omitted and using the **scaler** command instead.

```
layout new MyLayout 3840 2160<cr>
layout window MyLayout 0 0 0 1920 1080 0<cr>
layout window MyLayout 1 0 1080 1920 1080 1<cr>
layout window MyLayout 2 1920 0 960 540 2<cr>
layout window MyLayout 3 2880 0 960 540 3<cr>
layout window MyLayout 4 1920 540 960 540 4<cr>
layout window MyLayout 5 2880 540 960 540 2<cr>
layout window MyLayout 6 1920 1080 960 540 3<cr>
layout window MyLayout 7 2880 1080 960 540 4<cr>
layout window MyLayout 8 1920 1620 960 540 2<cr>
layout window MyLayout 9 2880 1620 960 540 3<cr>
set scaler Encoder1 1920 1080<cr>
set scaler Encoder2 1920 1080<cr>
set scaler Encoder3 1920 1080<cr>
set scaler Encoder4 1920 1080<cr>
set scaler Encoder5 1920 1080<cr>
multiview Decoder1 MyLayout 3840 2160 60<cr>
join multi Encoder1 Decoder1 0 scaled<cr>
join multi Encoder2 Decoder1 1 scaled<cr>
join multi Encoder3 Decoder1 2 scaled<cr>
join multi Encoder4 Decoder1 3 scaled<cr>
join multi Encoder5 Decoder1 4 scaled<cr>
```

ToC

Appendix B - How to Video wall



Multiple Decoders can be grouped together to form a video wall.
Each display requires a Decoder and at least one Encoder source is required.

'video wall' mode is similar to join fast mode, except the Decoder's frame buffer is now locked with multiple frame buffers of other Decoders. All the Decoders working in tandem are receiving the same video stream. Each Decoder crops the feed according to relative position within the video wall array and then scaled to a specified resolution.

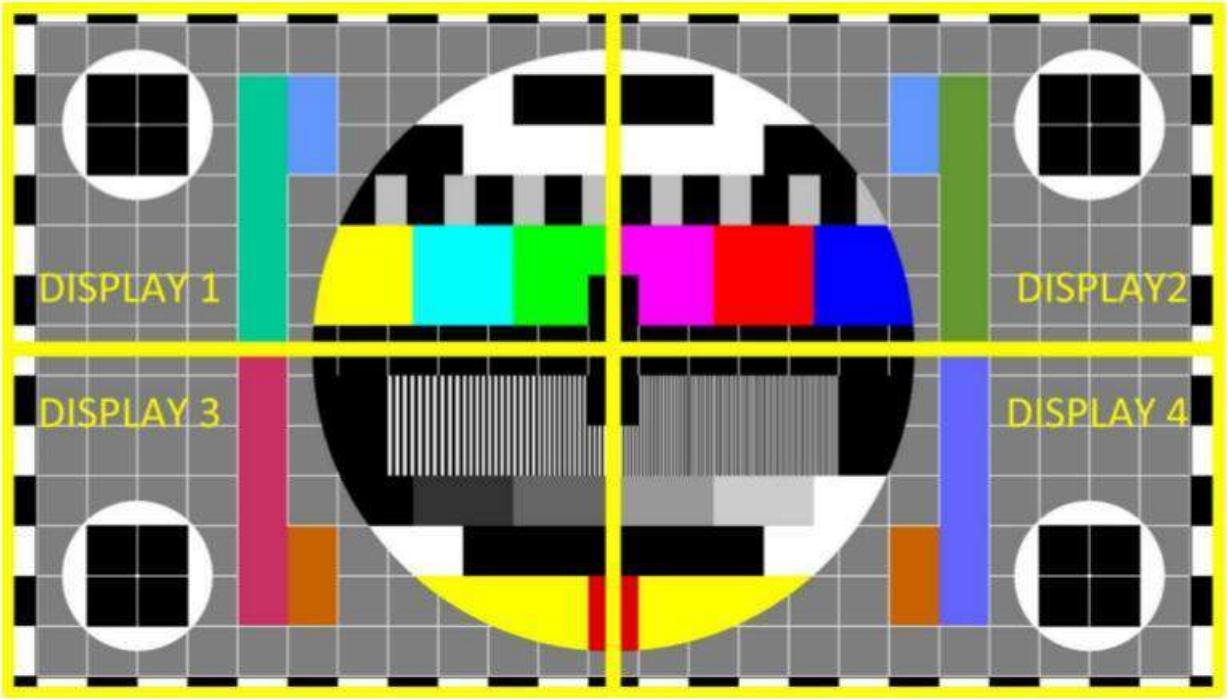
The table below shows available Video wall commands:

COMMAND	DESCRIPTION	REFERENCE
join wall	Used once for each display	4.11.1
join walladv	Used once for each display with advanced parameters	4.11.2

ToC

2x2 Video Wall example

This example can be found as a preset on the SDVoE Director Controller as 'demo_videowall_std_2x2'



Source Encoder - Encoder1 1920 x 1080 @ 60Hz, Displays width 615mm with 595mm viewable

Display Decoder1 - Decoder TopLeft
Display Decoder2 - Decoder TopRight
Display Decoder3 - Decoder BottomLeft
Display Decoder4 - Decoder BottomRight

ToC

2x2 Video Wall example continued…

DISPLAY 1

Standard wall - automatically calculated bezel compensation
join wall Encoder1 Decoder1 2x2 1 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder1 1920 1080 0 0 928 508 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 32) = 928
height = ((vertical resolution / number of vertical displays) - Bezel) = ((1080 / 2) - 32) = 508
h_offset = 0
v_offset = 0

DISPLAY 2

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder2 2x2 2 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder2 1920 1080 992 0 928 508 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 32) = 928
height = ((vertical resolution / number of vertical displays) - Bezel) = ((1080 / 2) - 32) = 508
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((1920 / 2) + 32) = 992
v_offset = 0

DISPLAY 3

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder3 2x2 3 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder3 1920 1080 0 572 928 508 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 32) = 928
height = ((vertical resolution / number of vertical displays) - Bezel) = ((1080 / 2) - 32) = 508
h_offset = 0
v_offset = ((vertical resolution / number of vertical displays) + Bezel) = ((1080 / 2) + 32) = 576

DISPLAY 4

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder4 2x2 4 size 1920 1080 60 bezel 615 595<cr>

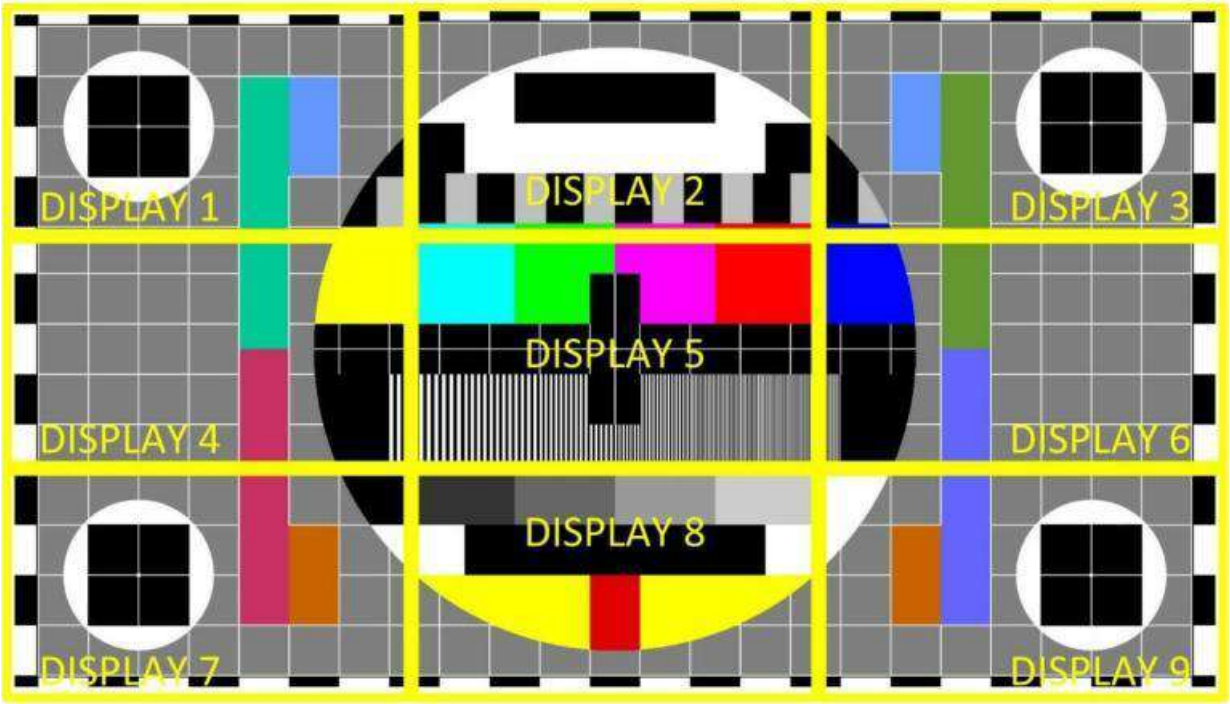
Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder4 1920 1080 992 572 928 508 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 64) = 928
height = ((vertical resolution / number of vertical displays) - Bezel) = ((1080 / 2) - 64) = 508
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((1920 / 2) + 64) = 992
v_offset = ((vertical resolution / number of vertical displays) + Bezel) = ((1080 / 2) + 64) = 576

ToC

3x3 Video Wall example

This example can be found as a preset on the SDVoE Director Controller as 'demo_videowall_std_3x3'



Source Encoder - Encoder1 3840 x 2160 @ 30Hz, Display resolution 1080p.

Display Decoder1 - Decoder TopLeft
Display Decoder2 - Decoder TopCenter
Display Decoder3 - Decoder TopRight
Display Decoder4 - Decoder MiddleLeft
Display Decoder5 - Decoder MiddleCenter
Display Decoder6 - Decoder MiddleRight
Display Decoder7 - Decoder BottomLeft
Display Decoder8 - Decoder BottomCenter
Display Decoder9 - Decoder BottomLeft

DISPLAY 1

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder1 3x3 1 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder1 1920 1080 0 0 1216 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = 0
v_offset = 0

ToC

3x3 Video Wall example continued…

DISPLAY 2

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder2 3x3 2 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder2 1920 1080 1344 0 1152 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) – (Bezel * 2)) = ((3840 / 3) – (64 * 2)) = 1152
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((3840 / 3) + 64) = 1344
v_offset = 0

DISPLAY 3

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder3 3x3 3 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder3 1920 1080 2624 0 1216 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = (((horizontal resolution / number of horizontal displays) * 2) + Bezel) = (((3840 / 3) * 2) + 64) = 2624
v_offset = 0

DISPLAY 4

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder4 3x3 4 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder4 1920 1080 0 784 1216 592 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) – (Bezel *2)) = ((2160 / 3) – (64 *2)) = 592
h_offset = 0
v_offset = ((vertical resolution / number of vertical displays) + Bezel) = ((2160 / 3) + 64) = 784

DISPLAY 5

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder5 3x3 5 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder5 1920 1080 1344 784 1152 592 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) – (Bezel * 2)) = ((3840 / 3) – (64 * 2)) = 1152
height = ((vertical resolution / number of vertical displays) – (Bezel *2)) = ((2160 / 3) – (64 *2)) = 592
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((3840 / 3) + 64) = 1344
v_offset = ((vertical resolution / number of vertical displays) + Bezel) = ((2160 / 3) + 64) = 784

ToC

3x3 Video Wall example continued…

DISPLAY 6

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder6 3x3 6 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder6 1920 1080 2624 784 1216 592 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) – (Bezel *2)) = ((2160 / 3) – (64 *2)) = 592
h_offset = (((horizontal resolution / number of horizontal displays) * 2) + Bezel) = (((3840 / 3) * 2) + 64) = 2624
v_offset = (((vertical resolution / number of vertical displays) + Bezel) = ((2160 / 3) + 64) = 784

DISPLAY 7

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder7 3x3 7 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder7 1920 1080 0 1504 1216 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = 0
v_offset = (((vertical resolution / number of vertical displays) * 2) + Bezel) = (((2160 / 3) * 2) + 64) = 1504

DISPLAY 8

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder8 3x3 8 size 1920 1080 60 bezel 615 595<cr>

Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder8 1920 1080 1344 1504 1152 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) – (Bezel * 2)) = ((3840 / 3) – (64 * 2)) = 1152
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((3840 / 3) + 64) = 1344
v_offset = (((vertical resolution / number of vertical displays) * 2) + Bezel) = (((2160 / 3) * 2) + 64) = 1504

DISPLAY 9

Standard wall - Automatically calculated bezel compensation
join wall Encoder1 Decoder9 3x3 9 size 1920 1080 60 bezel 615 595<cr>

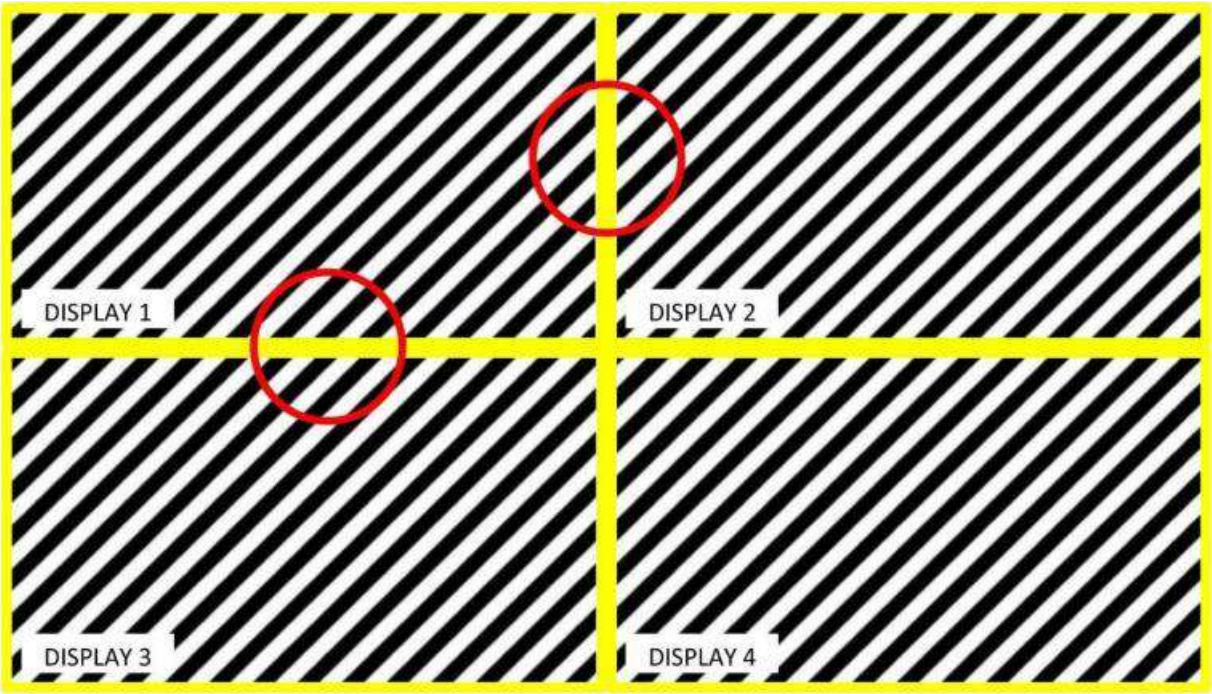
Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder9 1920 1080 2624 1504 1216 656 0 0 1920 1080 60<cr>

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((3840 / 3) - 64) = 1216
height = ((vertical resolution / number of vertical displays) - Bezel) = ((2160 / 3) - 64) = 656
h_offset = (((horizontal resolution / number of horizontal displays) * 2) + Bezel) = (((3840 / 3) * 2) + 64) = 2624
v_offset = (((vertical resolution / number of vertical displays) * 2) + Bezel) = (((2160 / 3) * 2) + 64) = 1504

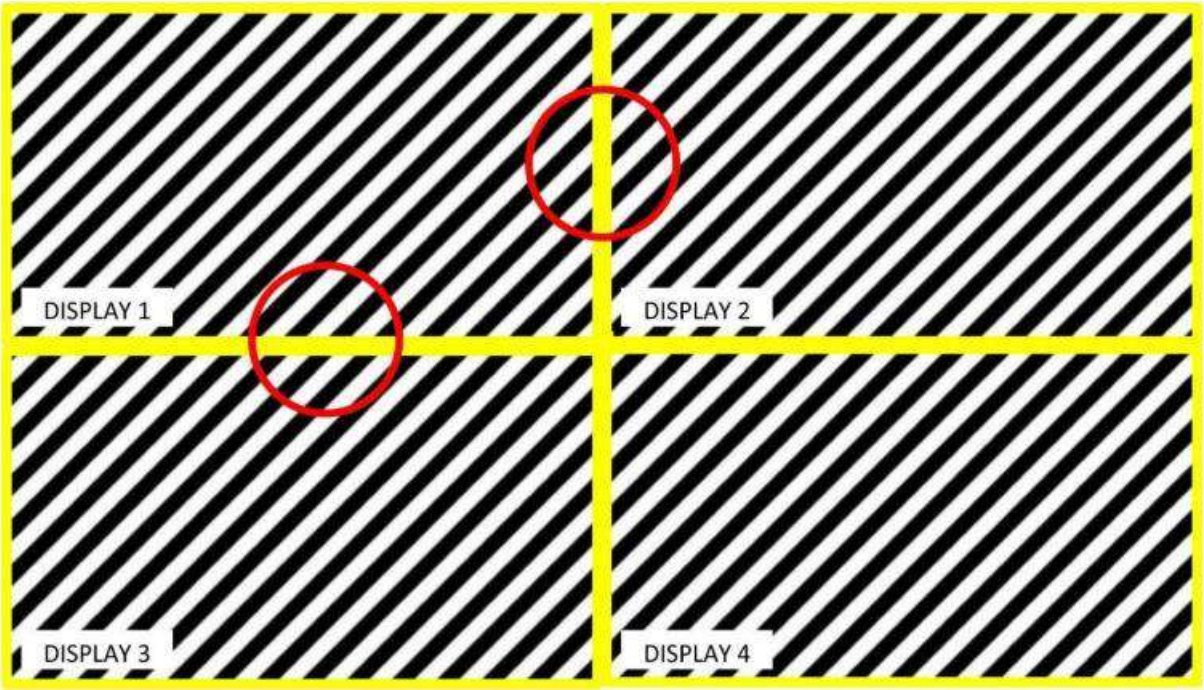
ToC

Bezel Compensation - walladv

When you display the video across multiple monitors, the monitor's bezel gaps can introduce space that was not intended to be there resulting in a disjointed image that is not continuous across all displays. With bezel compensation, part of the video is hidden behind the display bezel so that the bezel appears to be part of the video. Using bezel compensation produces a more continuous image across the displays and provides a more realistic experience. It is similar to looking through a window where the window frames block your view. Below is an example of a 2x2 with and without bezel compensation applied.



without bezel compensation



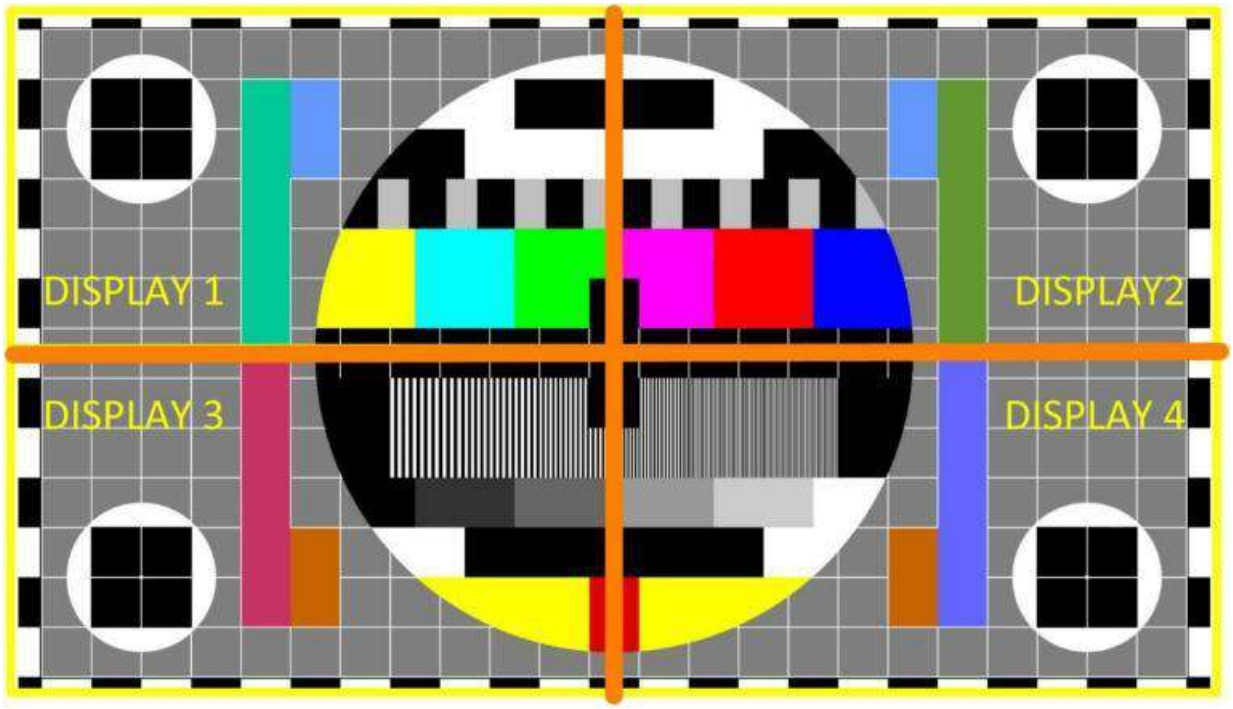
with bezel compensation

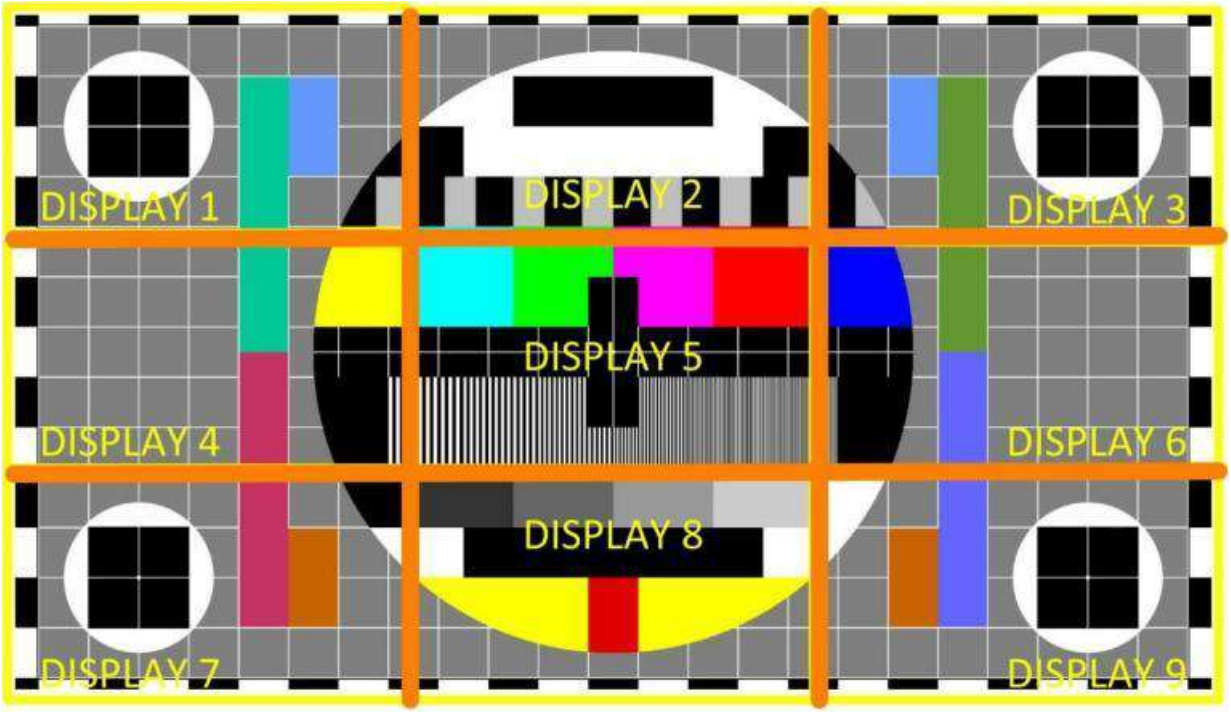
ToC

Bezel Compensation – walladv continued…

Bezel compensation can be calculated in a few different ways. If the display has a uniform bezel size then calculations can be simply performed. Calculations become a little more complex when the bezel sizes are not the same. For instance, some displays will have a larger bottom bezel than the top, or the left can vary compared to the right.

Bezel compensation only applies to edges of the video potions joining to another display. Only the orange edges of the below examples need to be compensated.





ToC

Bezel Compensation – walladv continued…

Definition of terms:

source_resolution_width = source video resolution width in pixels

source_resolution_height = source video resolution height in pixels

BW = physical width of the display bezel in millimetres when using uniform bezel size

BWL = physical width of the displays left bezel in millimetres when using different bezel sizes

BWR = physical width of the displays right bezel in millimetres when using different bezel sizes

BWT = physical width of the displays top bezel in millimetres when using different bezel sizes

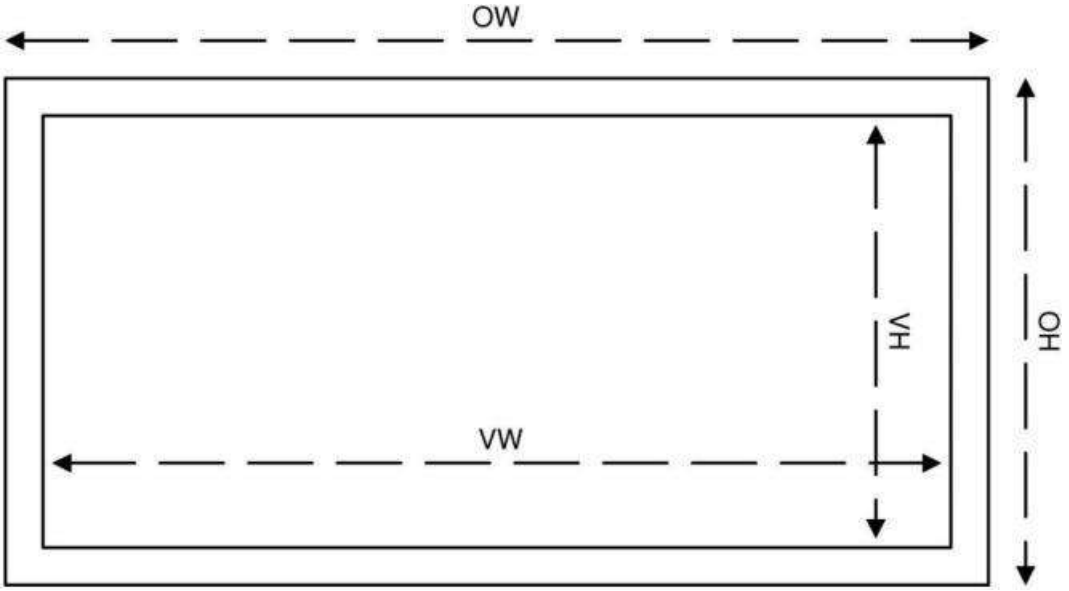
BWB = physical width of the displays bottom bezel in millimetres when using different bezel sizes

OW = physical overall width of the display in millimetres

VW = physical width of the displays viewable area in millimetres

OH = physical overall height of the display in millimetres

VH = physical height of the displays viewable area in millimetres



ToC

Bezel Compensation – walladv continued…

We are going to look at an example of a uniform bezel sized display, and one that is not.

1) Uniform bezel size

For a 2x2 video wall with uniform bezels we are going to use the following values:

- source_resolution_width = source video resolution width in pixels
- OW = physical overall width of the display in millimetres
- VW = physical width of the displays viewable area in millimetres

Firstly we need to know a few dimensions of the display in millimetres.

- physical width of the display bezel in millimetres (BW)
- physical overall width of the display in millimetres (OW)
- physical width of the displays viewable area in millimetres (VW)

BW can be physically measured, or calculated from OW and VW like below:
((OW - VW) / 2) = ((615 - 595) / 2) = 10

The last detail we need to know is the source resolution in pixels.
This example uses 1920x1080 so source_resolution_width = 1920

Now we have all the information we require, we can use the formula below to calculate the bezel compensation required in pixels.

((source_resolution_width * BW) / VW) =
((1920 * 10) / 595) =
(19200 / 595) =
32px

We can also work out the Bezel compensation by using pixel density.
(source_resolution_width / VW) = pixel_density
(1920 / 595) = 3.22 px_per_mm

((OW – VW) / 2) = bezel_size (mm)
((615 – 595) / 2) = 10 (mm)

(bezel_size x pixel_density) = bezel_compensation
10 x 3.22 = 32px

width = ((horizontal resolution / number of horizontal displays) - Bezel)

height = ((vertical resolution / number of vertical displays) - Bezel)

h_offset = ((horizontal resolution / number of horizontal displays) + Bezel)

v_offset = ((vertical resolution / number of vertical displays) + Bezel)

ToC

Bezel Compensation – walladv continued…

2) Different bezel sizes

For a 2x2 video wall with different bezels we are going to use the following values:

- source_resolution_width = source video resolution width in pixels
- source_resolution_height = source video resolution height in pixels
- OW = physical overall width of the display in millimetres
- VW = physical width of the displays viewable area in millimetres
- BWx = bezel width in millimetres

Firstly we need to know a few dimensions of the display in millimetres.

- physical width of the display left bezel in millimetres (BWL)
- physical width of the display right bezel in millimetres (BWR)
- physical width of the display top bezel in millimetres (BWT)
- physical width of the display bottom bezel in millimetres (BWB)
- physical overall width of the display in millimetres (OW)
- physical width of the displays viewable area in millimetres (VW)
- physical overall height of the display in millimetres (OH)
- physical height of the displays viewable area in millimetres (VH)

Now the bezel on each side of the display need to be physically measured.
BWL = 10mm
BWR = 10mm
BWT= 10mm
BWB = 20mm

Now we need to know how many displays we have in rows and columns.
A 2x2 wall has 2 rows and 2 columns.
So DH = 2 and DV = 2

The last detail we need to know is the source resolution in pixels.
This example uses 1920x1080 so source_resolution_width = 1920 and source_resolution_height = 1080

ToC

Bezel Compensation – walladv continued…

Now we have all the information we require, we can use the formulas below to calculate the bezel compensation required in pixels.

Left bezel compensation =
((source_resolution_width * BWL) / VW) =
((1920 * 10) / 595) =
(19200 / 595) =
32px

Right bezel compensation =
((source_resolution_width * BWR) / VW) =
((1920 * 10) / 595) =
(19200 / 595) =
32px

Top bezel compensation =
((source_resolution_width * BWT) / VH) =
((1080 * 10) / 335) =
(10800 / 335) =
32px

Bottom bezel compensation =
((source_resolution_width * BWB) / VH) =
((1080 * 20) / 335) =
(21600 / 335) =
64px

width = ((horizontal resolution / number of horizontal displays) – (Bezel Left + Bezel Right))

height = ((vertical resolution / number of vertical displays) - (Bezel Left + Bezel Right))

h_offset = ((horizontal resolution / number of horizontal displays) + Bezel Left)

v_offset = ((vertical resolution / number of vertical displays) + Bezel Top)

ToC

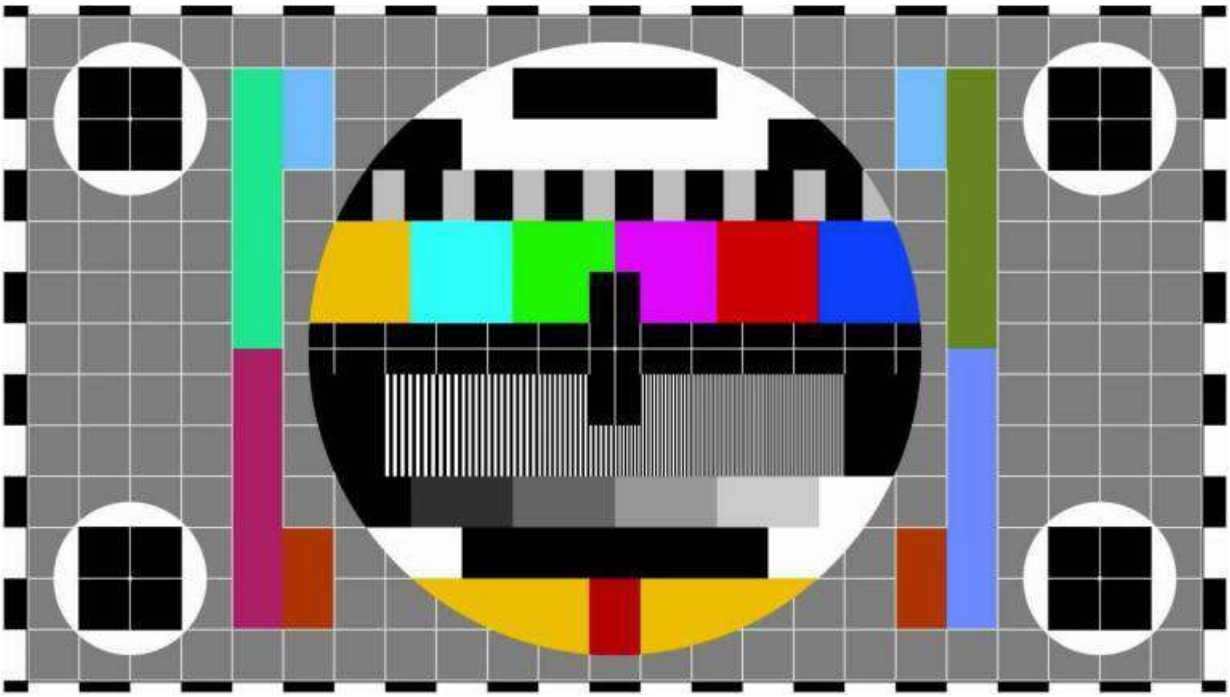
viewport - walladv

The viewport is the rectangular region of a screen where video is displayed on Decoders. Areas on the screen outside this region are black. You can use the viewport to apply black to the top and bottom, or sides of the video to maintain original aspect ratio's or the location of the image on the display.

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, with respect to the specified width.

The **viewport_width** and **viewport_height** arguments specify respectively the width and height of the viewport, in pixels, with respect to the specified height.

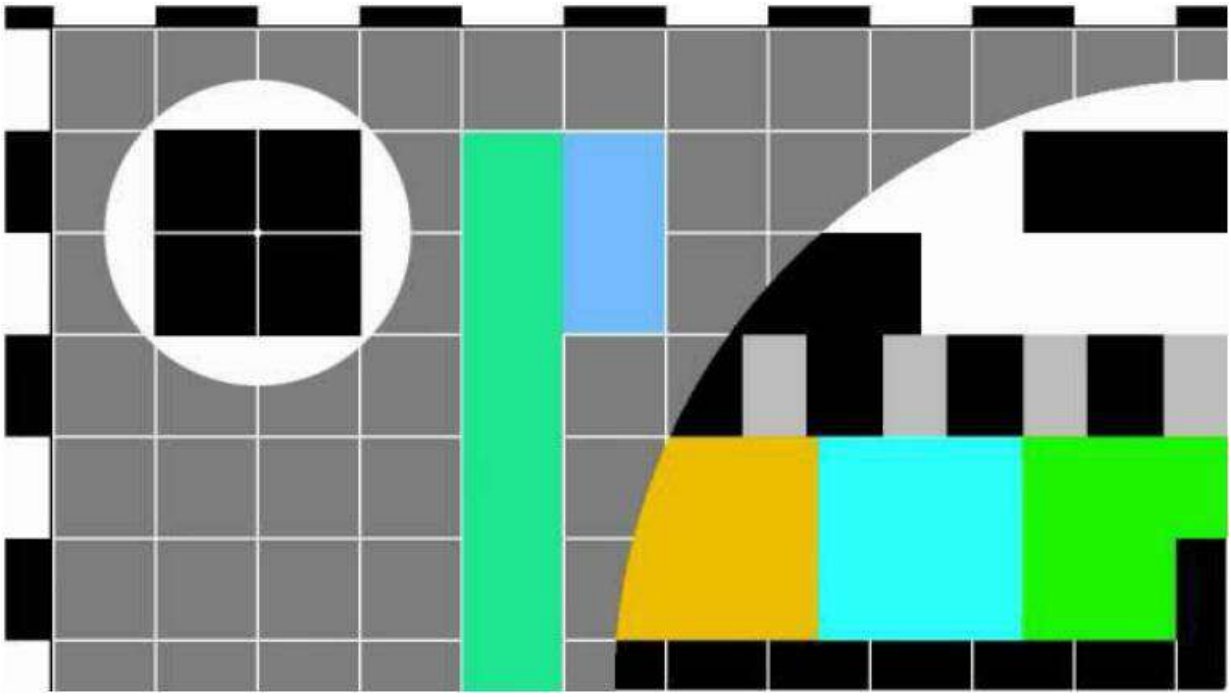
Original full screen 1920x1080 video source



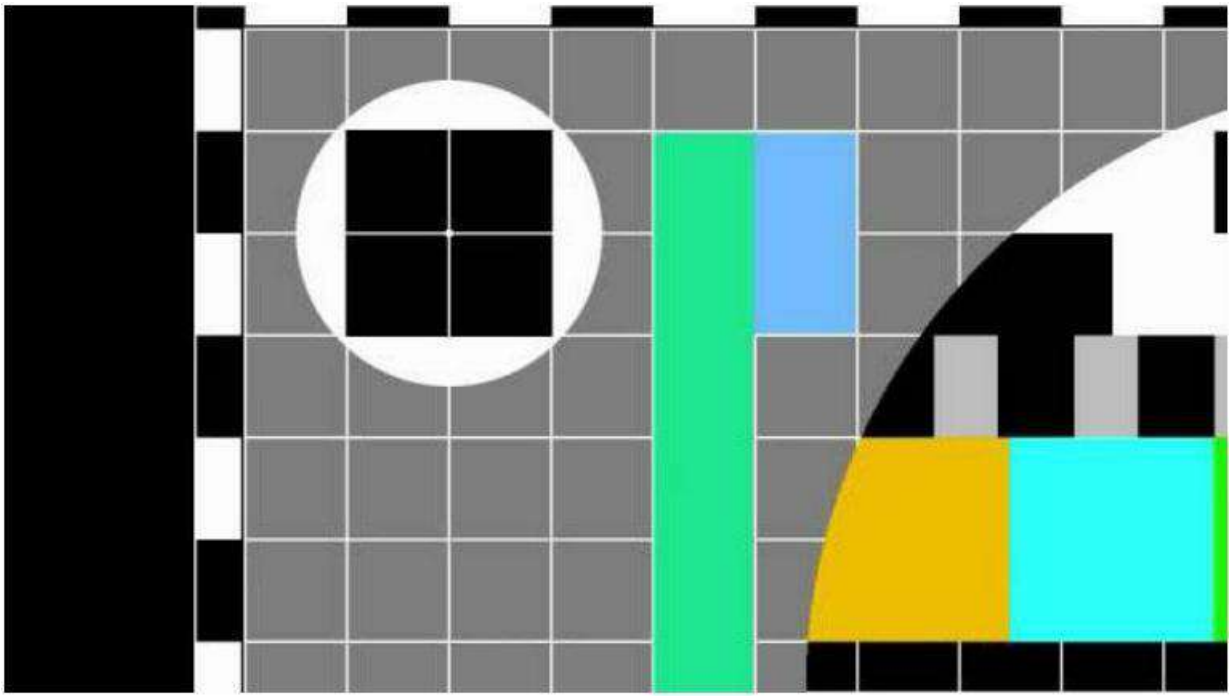
ToC

viewport – walladv continued…

Top left 960x540 portion of original video source still displayed as 1920x1080 (no bezel comp)
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 0 0 1920 1080 60<cr>



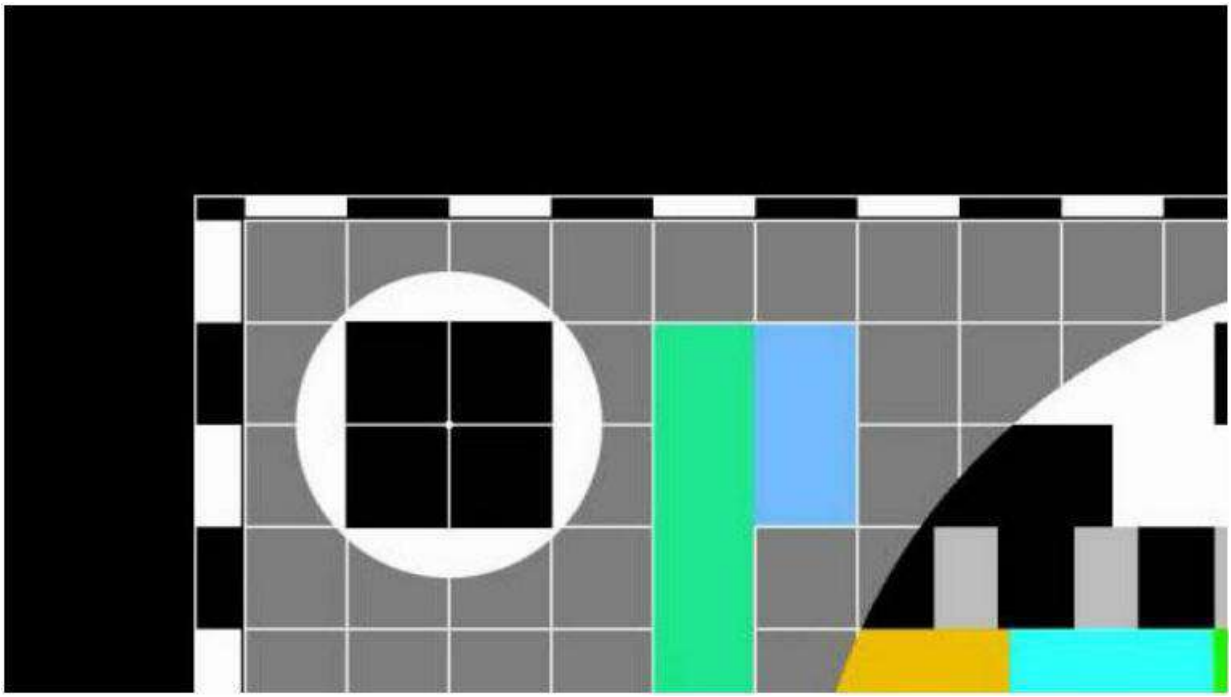
Add 100 pixels of black to the left and reduce viewport_width by 100 pixels.
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 **100** 0 **1820** 1080 60<cr>



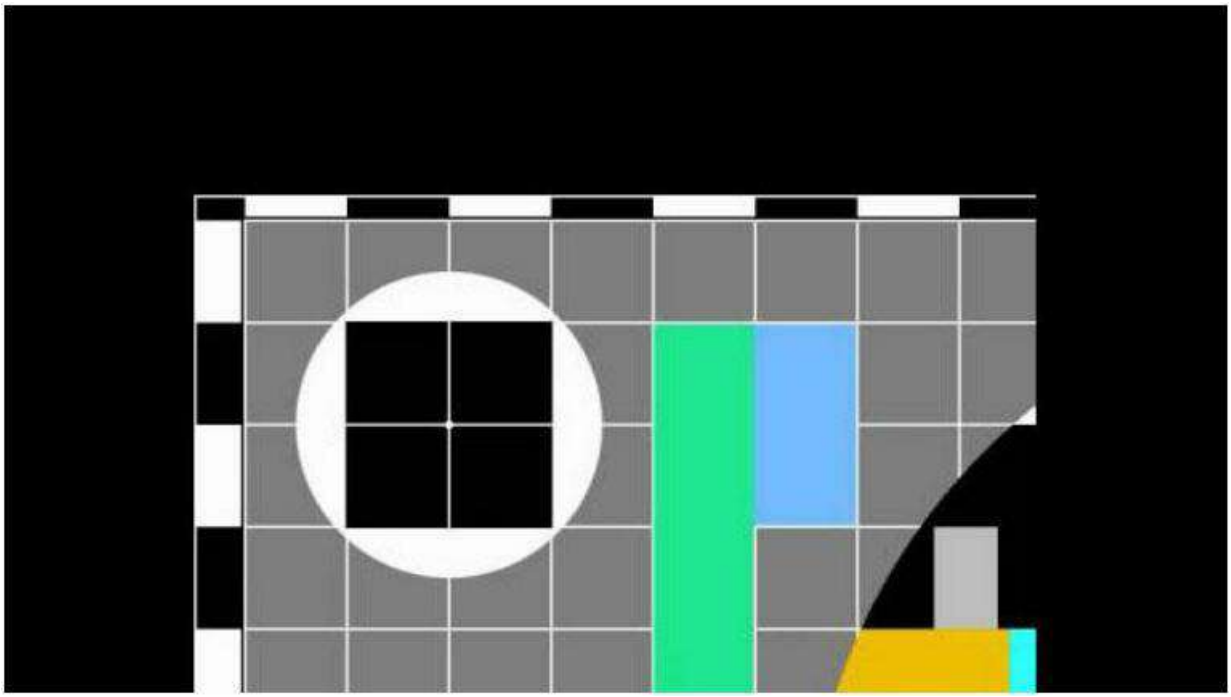
ToC

viewport – walladv continued…

Add 100 pixels of black to the top and reduce viewport_height by 100 pixels.
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 100 **100** 1820 **980** 60<cr>



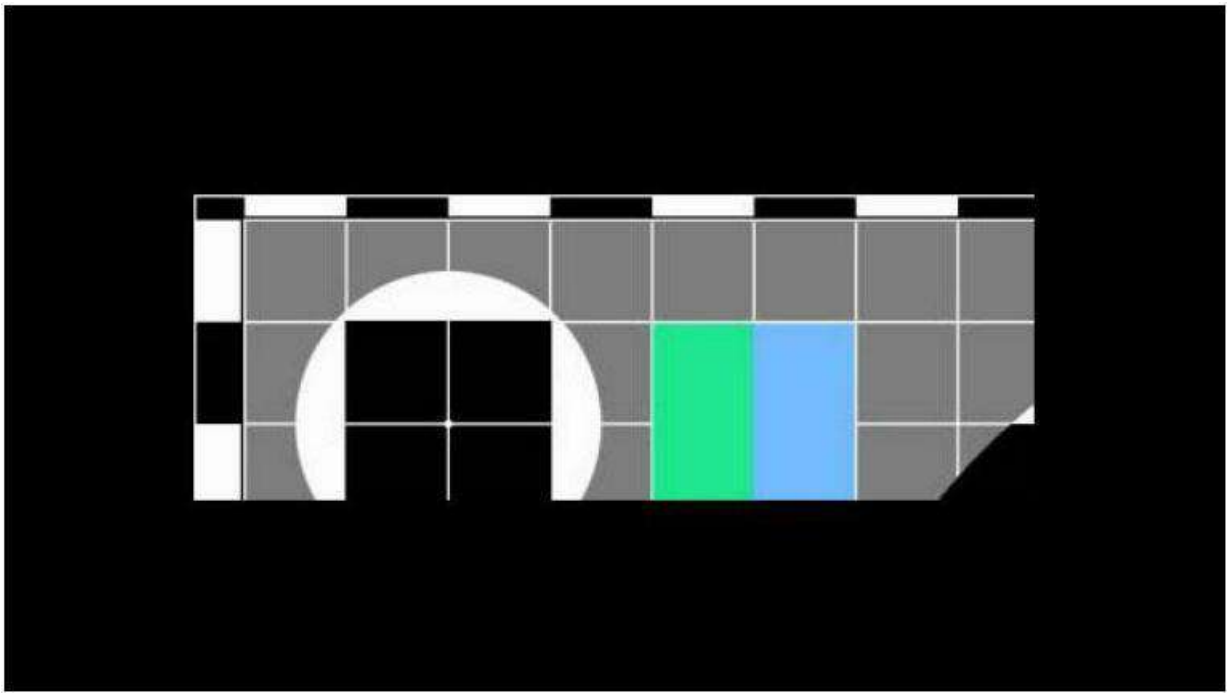
Add 100 pixels of black to the right by reducing viewport_width by 100 pixels.
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 100 100 **1720** 980 60<cr>



ToC

viewport – walladv continued…

Add 100 pixels of black to the bottom by reducing viewport_height by 100 pixels.
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 100 100 1720 **880** 60<cr>

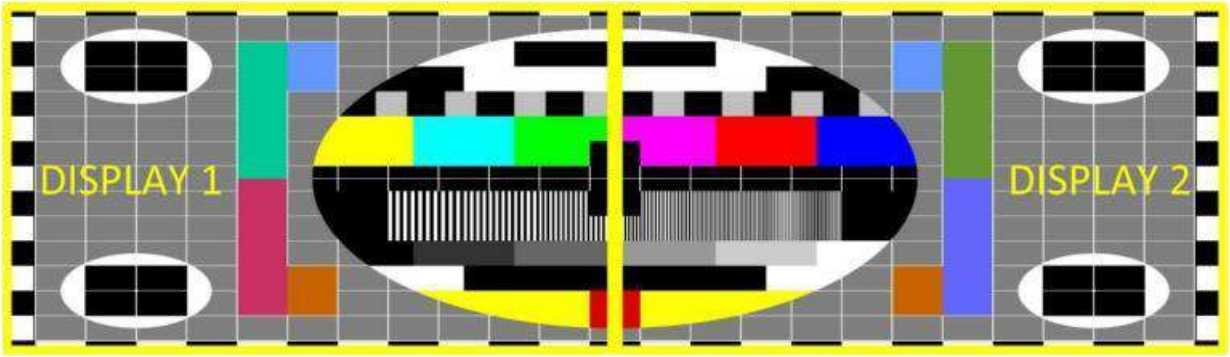


ToC

viewport – walladv continued…

This example can be found as a preset on the SDVoE Director Controller as 'demo_videowall_adv_full_2x1'.

When a 16:9 video image is displayed on a LIGHTNING 32:9 (2x1) video wall, the image width will stretch to fit the displays destroying the original aspect ratio of the image as below:



DISPLAY 1

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 16) = 944
height = 1080
h_offset = 0
v_offset = 0

join walladv Encoder1 Decoder1 1920 1080 0 0 944 1080 0 0 1920 1080 60<cr>

DISPLAY 2

width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 16) = 944
height = 1080
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((1920 / 2) + 16) = 976
v_offset = 0

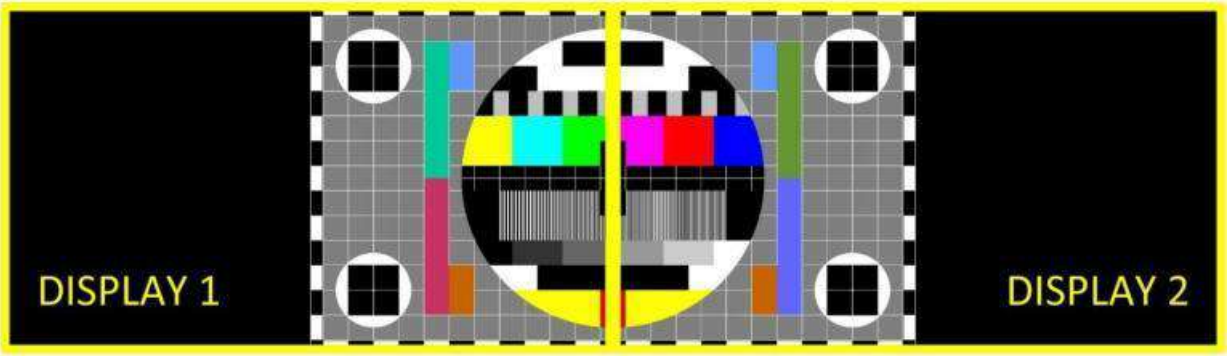
join walladv Encoder1 Decoder2 1920 1080 976 0 944 1080 0 0 1920 1080 60<cr>

ToC

viewport – walladv continued…

This example can be found as a preset on the SDVoE Director Controller as 'demo_videowall_adv_keep_2x1'.

LIGHTNING can maintain the original aspect ratio of the image, black is added to the sides of the image with the viewport as per below:



We know the image is 1920x1080, and we know the width is now 3840 (width of 2 displays 1920x1080) so we can work out the black is 960px each side.

DISPLAY 1

width = 1920
height = 1080
h_offset = 0
v_offset = 0
keep_width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 16) = 944
keep_height = 1080
viewport_horz = (((horizontal resolution * number of horizontal displays) – source resolution) / 2) = 960
viewport_vert = 0
viewport_width = (((horizontal resolution * number of horizontal displays) – source resolution) / 2) = 960
viewport_height = 1080

join walladv Encoder1 Decoder1 1920 1080 0 0 944 1080 960 0 960 1080 60<cr>

DISPLAY 2

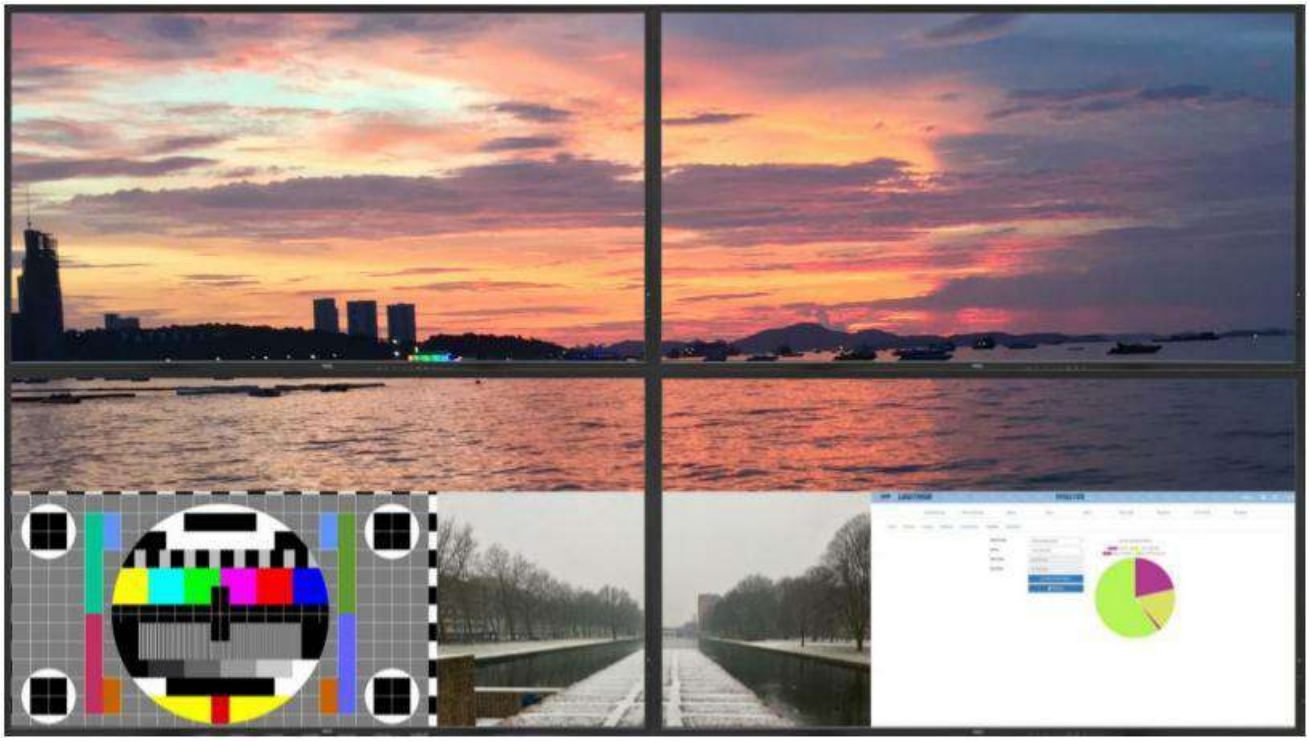
width = 1920
height = 1080
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((1920 / 2) + 16) = 976
v_offset = 0
keep_width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 16) = 944
keep_height = 1080
viewport_horz = 0
viewport_vert = 0
viewport_width = (((horizontal resolution * number of horizontal displays) – source resolution) / 2) = 960
viewport_height = 1080

join walladv Encoder1 Decoder2 1920 1080 976 0 944 1080 0 0 960 1080 60<cr>

ToC

Appendix C - How to Video Wall with Multiview

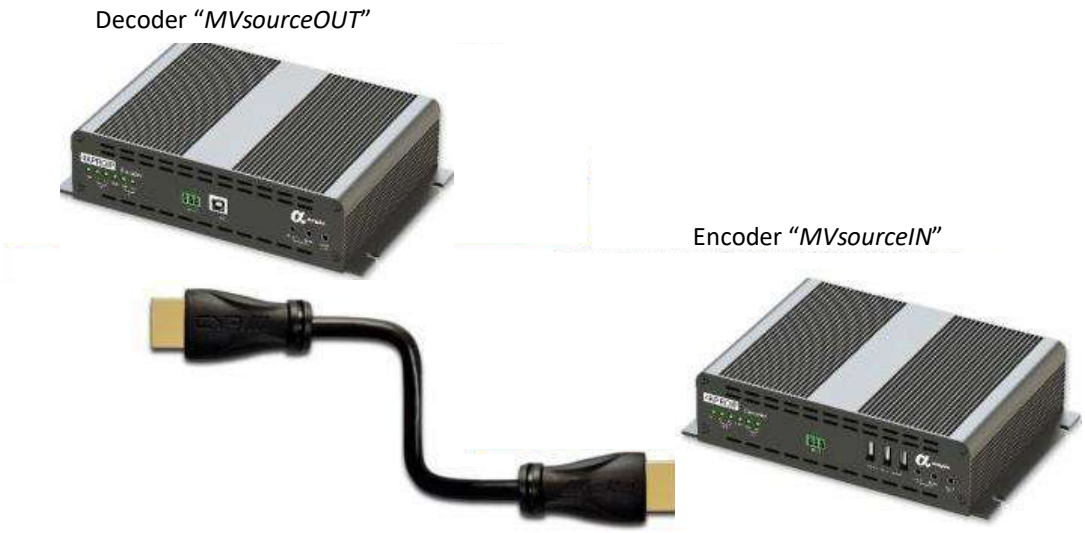
Here is an explanation on how to create a Multiview layout on a video wall. To do this there are four (4) simple steps, make a physical HDMI connection between a dedicated Decoder and Encoder, create a Video Wall preset along with Multiview presets, and lastly execute the presets. The Video Wall preset is only required to be executed once to set up the Video Wall and to select the Multiview Encoder as the video source. Multiple Multiview presets can then be executed to change the layout as required.



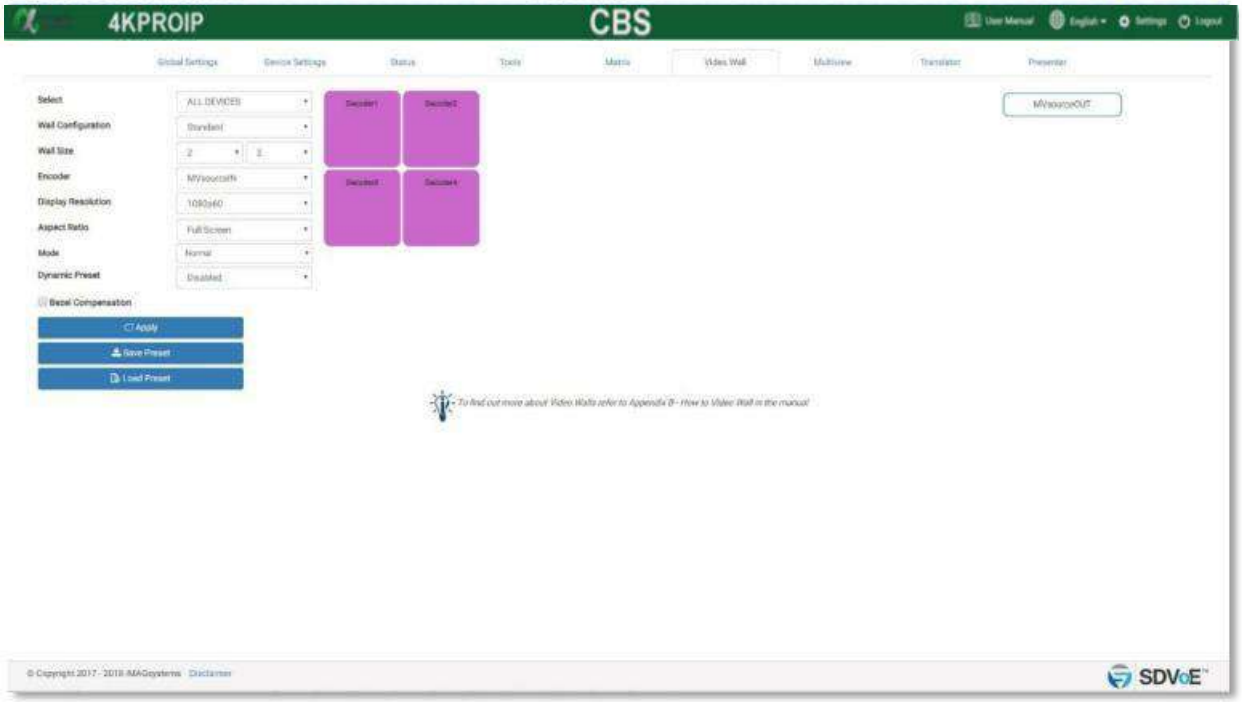
ToC

Appendix C - How to Video Wall with Multiview continued…

1. Firstly a dedicated Decoder is required to create a HDMI video signal containing the Multiview Layout. Let's name this Decoder '*MVsourceOUT*'. The HDMI output of this Decoder is connected to a dedicated Encoder's HDMI input which will then convert the Multiview layout video into a stream accessible on the network for any other Decoder to display. Let's name this Encoder '*MVsourceIN*'.



2. Now create a Video Wall preset using the required display layout. For this example a 2x2 Video Wall is used and the preset is saved as '*sample_VideoWall_2x2*'.

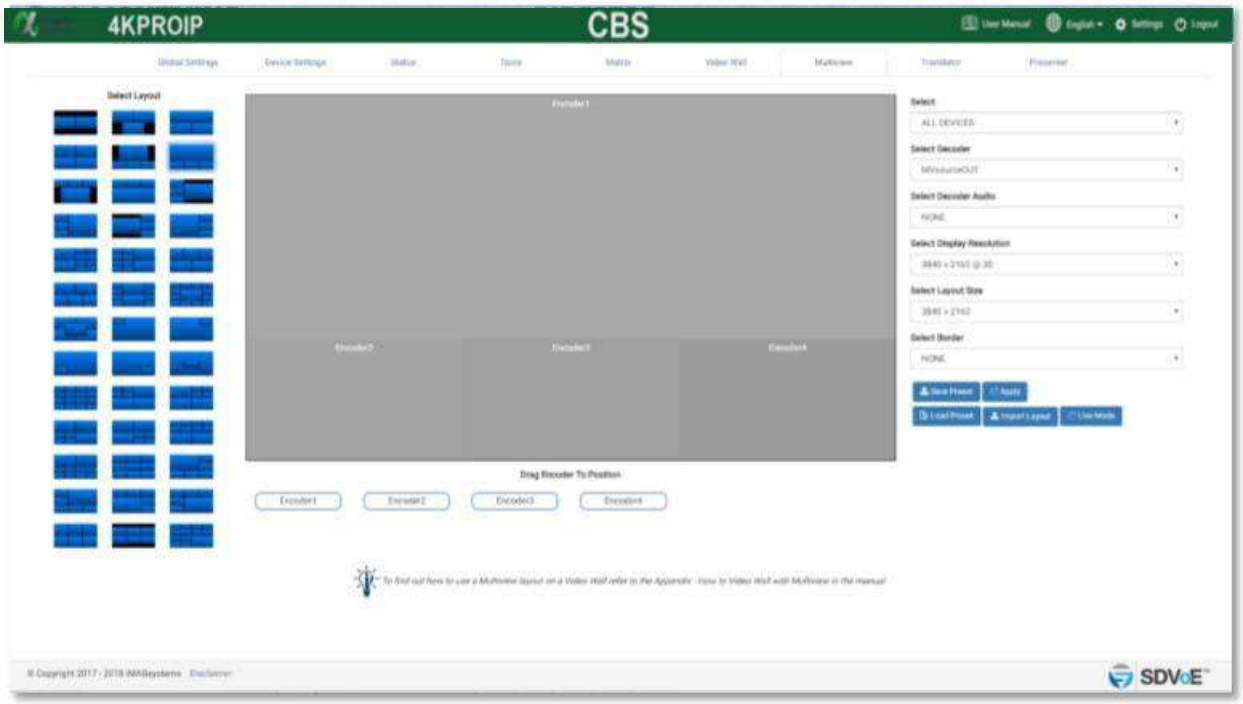


ToC

Appendix C - How to Video Wall with Multiview continued…

3. Now create a Multiview layout as required using Decoder '*MVsourceOUT*'. For this example layout #6 is used and the preset is saved as '*sample_Multiview_6*'.

4. Now these two presets just need to be executed. Whenever a change of the layout is required, only a different Multiview layout preset is required to be executed as the Video Wall configuration will remain the same.



Note: The two presets can be combined into a single preset by editing one of them and pasting the contents of the other.

ToC

Appendix D - How to HTTP request

GET = http://<*controllerURL*>/api/command/<*LIGHTNING_API_COMMAND*>/<*KEY*>

POST = http://<*controllerURL*>/api/command/{'cmd': '<*LIGHTNING_API_COMMAND*>', 'key': '<*KEY*>'}

Example 1: POST - ajax

```
<script language='JavaScript' type='text/javascript'>
        var controllerIP = '169.254.1.1'; *change this to the same IP address as the SDVoE Director controller
        var BaseURL = 'http://' + controllerIP + '/api/command/';
        var MAXIMUM_WAITING_TIME = 5000; *timeout in milliseconds
        var CheckStatusTimer;
        var key = '123xyz'; *replace this with the generated security key
        var command = '123xyz'; *replace with any LIGHTNING API command

        $.ajax({
                type: 'POST',
                crossDomain: true,
                contentType: 'application/json; charset=utf-8',
                dataType: 'text',
                url: BaseURL,
                data: "{'cmd':' + command + ','key':' + key + '}",
                timeout: MAXIMUM_WAITING_TIME,
                success: function(data, textStatus, XMLHttpRequest){
                        console.log(data);
                },
                error: function (XMLHttpRequest, textStatus, errorThrown) {
                        console.log('ERROR = ' + errorThrown);
                }
        });
</script>
```

Example 2: POST - xhr

```
<script language='JavaScript' type='text/javascript'>
        var controllerIP = '169.254.1.1'; *change this to the same IP address as the SDVoE Director controller
        var BaseURL = 'http:// ' + controllerIP + '/api/command/';
        var key = '123xyz'; *replace this with the generated security key
        var command = '123xyz'; *replace with any LIGHTNING API command
        var xmlRequest = new XMLHttpRequest();
        xmlRequest.open('POST', BaseURL, true);
        var params = "{'cmd':' +command+ ', 'key':' + key + '}";
        var MAXIMUM_WAITING_TIME = 5000;
        xmlRequest.onreadystatechange = function () {
                if (this.readyState == 4) {
                        clearTimeout(xmlTimer);
                        if(this.status == 200){
                                console.log(this.responseText);
                        }else{
                                console.log('ERROR = ' + this.status + ' ' + this.statusText);
                        }
                }else{
                        if(this.status != 200){
                                console.log('ERROR = ' + this.status + ' ' + this.statusText);
                        }
                }
        };
        xmlRequest.send(params);
        var xmlTimer = setTimeout(function() {
                xmlRequest.abort();
                console.log('ERROR = timeout');
        }, MAXIMUM_WAITING_TIME);
</script>
```

ToC

Example 3: GET - xhr

```
<script language='JavaScript' type='text/javascript'>
        var controllerIP = '172.30.0.220'; *change this to the same IP address as the SDVoE Director controller
        var BaseURL = 'http://' + controllerIP + '/api/command/';
        var key = '123xyz'; //replace this with the generated security key
        var command = '123xyz'; //replace with an LIGHTNING API command
        var MAXIMUM_WAITING_TIME = 5000;
        var btn2xhr = new XMLHttpRequest();
        btn2xhr.open('GET', BaseURL + '<command>/' + key);
        btn2xhr.onreadystatechange = function () {
                if (this.readyState == 4) {
                        clearTimeout(btn2xhrTimer);
                        if(this.status == 200){
                                document.getElementById('Text0').innerHTML = this.responseText;
                        }else{
                                document.getElementById('Text0').innerHTML = 'ERROR = ' + this.status + ' ' + this.statusText;
                        }
                } else {
                        if(this.status != 200){
                                document.getElementById('Text0').innerHTML = 'ERROR = ' + this.status + ' ' + this.statusText;
                        }
                }
        };
        btn2xhr.send(null);
        var btn2xhrTimer = setTimeout(function() {
                btn2xhr.abort();
                document.getElementById('Text0').innerHTML = 'ERROR = timeout';
        }, MAXIMUM_WAITING_TIME);
</script>
```

Example 4: IFTTT – Webhooks – POST

URL
http://<controllerIP>/api/command/

Method
POST

Content Type
application/json

Body
{'cmd':'<LIGHTNING_API_COMMAND>','key':'<KEY>'}

Example 5: IFTTT – Webhooks – GET

URL
http://<controllerIP>/api/command/<LIGHTNING_API_COMMAND>/<KEY>

Method
GET

Content Type
text/plain

ToC

Example 6: bt.tn – Auxiliary HTTP request GET

URL=<controllerIP>/api/command/<LIGHTNING_API_COMMAND>/<KEY> port=80
**Note: All spaces in the <LIGHTNING_API_COMMAND> must be url-encoded as %20.*

See Percent-encoding chart below:

!	#	$	&	'	(	)	*	+	,	/	:	;	=	?	[	]
%21	%23	%24	%26	%27	%28	%29	%2A	%2B	%2C	%2F	%3A	%3B	%3D	%3F	%5B	%5D
space	‘	%	-	.	<	>	\	^	_	`	{	|	}	~	@	
%20	%22	%25	%2D	%2E	%3C	%3E	%5C	%5E	%5F	%60	%7B	%7C	%7D	%7E	%40	

Example 7: bt.tn – HTTP POST

HTTP URL - Specify URL
http://<controllerIP>/api/command

HTTP METHOD
POST

ARGUMENTS - application/json
{'cmd':'<LIGHTNING_API_COMMAND>','key':'<KEY>'}

ToC

Appendix E – Preset Logic

Basic if else logic can be applied within a preset to allow you to build some *smarts* into your system. All **get** commands can be used as an expression.

The following syntax applies:

```
if (something) {
    do_something
    …
} else {
    do_this_instead
    …
}
```

The following **get** commands will return a **string** value that can be used with:

- **==** (equal to)
- **!=** (not equal to)

- get edid
- get video_source
- get audio_source
- get audio_io
- get audio_out
- get encoder
- get scaler <*encoder_device_name*> all
- get window <*layout_name*> <*index*> all
- get video <*encoder_device_name*> cs
- get video <*encoder_device_name*> sm
- get ver
- get devices
- get status
- get api
- get video_mode
- get security

Example 1:

```
if (get encoder Decoder1 == Encoder1) {
    join fast Encoder2 Decoder1 size 1920 1080 60
} else {
    join fast Encoder1 Decoder2 size 1920 1080 60
}
```

Example 2:

```
if (get audio_out Decoder1 != hdmi) {
    set audio_out Decoder1 hdmi
}
```

ToC

Appendix E – Preset logic continued…

The following **get** commands will return an **integer** value that can be used with:

- **==** (equal to)
- **!=** (not equal to)
- **<** (less than)
- > (greater than)

  - get temp
  - get video <*encoder_device_name*> width
  - get video <*encoder_device_name*> height
  - get video <*encoder_device_name*> fps
  - get video <*encoder_device_name*> bpp
  - get scaler <*encoder_device_name*> width
  - get scaler <*encoder_device_name*> height
  - get scaler <*encoder_device_name*> fps
  - get window <*layout_name*> <*index*> width
  - get window <*layout_name*> <*index*> height
  - get bandwidth
  - get preferred

Example:

```
if (get temp Encoder1 > 80) {
    stop av Encoder1
} else {
    join fast Encoder1 Decoder1
}
```

The following **get** commands will return a **boolean** value that can be used with:

- **not** (false)

  - get video_status
  - get display_status
  - get frame_converter
  - get video_mute
  - get video_compress
  - get presenter

Example 1:

```
if (get video_status Encoder1) {
    join fast Encoder1 Decoder1
 } else {
    if (get video_status Encoder2) {
        join fast Encoder2 Decoder1
    }
}
```

Example 2:

```
if not(get video_status Encoder1) {
    join fast Encoder2 Decoder1
}
```

ToC