DIRECTOR
COMMAND GUIDE
THUNDER

Les technologies pour voir et entendre plus loin

SAHF.TECH®

See And Hear Further TECHnologies

DIRECTOR THUNDER

COMMAND GUIDE V1.4.0.0

DIRECTOR

ASSISTANCE

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 DIRECTOR THUNDER command guide and develop client control applications for THUNDER devices.

1.1 License Requirements

The Director Controller must have a valid license key entered before use or trying to connect to the Telnet TCP control port 6980.
If no valid license is active the 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 Director Controller and issue commands using ascii strings terminated with a carriage return <cr> 0x0D. This allows any Telnet client to be used with the system.

The 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 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 system 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/<*DIRECTOR_API_COMMAND*>/<*KEY*>
POST = http://<controllerURL>/api/command/{'cmd':'<*DIRECTOR_API_COMMAND*>','key':'<*KEY*>'}

Refer Appendix A - How to HTTP request

ToC

2 Command Overview

Commands are in a simple ASCII text format. For each command, the 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. The API is to be used synchronous communication.

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

  - join video encoder1 decoder1<cr>
  - join video encoder1 all<cr>
  - join video encoder1 MyGroup<cr>

  - stop encoder1<cr>
  - start encoder1<cr>

  - send serial decoder1 "my data string\x0D"<cr>

ToC

3 System Commands

3.1 Command reboot

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

3.1.1 Command usage

reboot **[**key:<*security_key*>**]** <*device_name*> / <*group_name*> / all / all_rx / all_tx<cr>

3.1.2 Description

Causes the target device to restart and become unavailable for a short period while restarting.

3.1.3 Arguments

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

3.1.4 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.

3.1.5 Return Value

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

3.1.6 Command Examples

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

3.1.7 Return Examples

```
reboot success<cr>

reboot error [incomplete]<cr>
reboot error [invalid response]<cr>
reboot error [device '<device_name>' not found]<cr>
reboot error [device '<device_name>' disconnected]<cr>
```

ToC

4 Command join

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

join video command provides independent routing of the video.

join audio command provides independent routing of the audio.

join ir 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 all command provides combined routing of all signals.

join av command provides combined routing of the video and audio together.

join kvm command provides combined routing of the video, audio and USB together.

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

join usb_ext command provides independent routing of external USB Extenders.

Commands missing **mode** return:

```
join error [incomplete]<cr>
```

Commands with invalided **mode** return:

```
join error [invalid mode]<cr>
```

ToC

4. Command join video

4.1.1 Command usage

join video **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]** **[**<original> | <auto> | **[**size <*video_mode*>**]]**<cr>

4.1.2 Description

The command **join video** is used for independent video routing.

4.1.3 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)
original	Keyword '**original**' will set the Decoder resolution as pass-though to match the Encoder video resolution. (optional)
auto	Keyword '**auto**' is only available when the Decoder has a monitor connected with valid EDID. The Decoders resolution will be set to the connected displays preferred resolution. (optional)
size	Keyword '**size**' is followed by the video mode (optional)

4.1.4 Notes

- **all** can replace *decoder_device_name* when the video is required to be connected to all Decoders.
- Only use **'original'** or **'auto'** or **'size'**.
- The name of a group can replace *decoder_device_name* when the video is required to be connected to all Decoders in a group.
- '**auto**' = Decoder's connected displays preferred resolution
- '**original**' = Encoder source resolution
- Valid video modes:
  - 2160p60 * on compatible devices only
  - 2160p50 * on compatible devices only
  - 2160p30
  - 1080p60
  - 1080p50
  - 720p60
  - 720p50

ToC

4 Notes1 .5 Return Value

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

4 Notes1 .6 Command Examples

```
join video Encoder1 Decoder1<cr>
join video Encoder1 all<cr>
join video Encoder1 MyGroup<cr>
join video Encoder1 Decoder1 exclusive<cr>
join video Encoder1 Decoder1 original<cr>
join video Encoder1 Decoder1 auto<cr>
join video Encoder1 Decoder1 size 1080p60<cr>
join video key:abc123 Encoder1 Decoder1<cr>
```

4.6.7 Return Examples

```
join video success<cr>

join video error [incomplete]<cr>
join video error [join not permitted]<cr>
join video error [invalid parameter]<cr>
join video error [invalid video_mode]<cr>
join video error [monitor not detected]<cr>
join video error [unsupported monitor]<cr>
join video error [invalid response]<cr>
join video error [encoder '<encoder_device_name>' not found]<cr>
join video error [decoder '<decoder_device_name>' not found]<cr>
join video error [device '<device_name>' disconnected]<cr>
join video error [group devices not found]<cr>
```

4. Command join audio

4.2 Command

join audio **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]**<cr>

4.2 Example

The command **join audio** is used for independent audio routing.

4.2

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)

4.2.4 Notes

- **all** can replace *decoder_device_name* when the audio is required to be connected to all Decoders.
- The name of a group can replace *decoder_device_name* when the audio is required to be connected to all Decoders in a group.

4.2.5 Return Value

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

4.2.6 Command join audio 2

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

4.2.7 Return Example2

```
join audio success<cr>

join audio error [incomplete]<cr>
join audio error [join not permitted]<cr>
join audio error [invalid parameter]<cr>
join audio error [invalid response]<cr>
join audio error [encoder '<encoder_device_name>' not found]<cr>
join audio error [decoder '<decoder_device_name>' not found]<cr>
join audio error [device '<device_name>' disconnected]<cr>
join audio error [group devices not found]<cr>
```

ToC

4.3 Command join ir

4.3Command

join ir **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> [<exclusive>]<cr>

4.3.2 Description

The command **join ir** is used for independent infrared routing.

4.3

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder, Group or '**all**'
exclusive	Keyword '**exclusive**' will remove all other joins (optional)

4.3.4 Notes

- **all** can replace *decoder_device_name* when the infrared is required to be connected to all Decoders.
- The name of a group can replace *decoder_device_name* when the infrared is required to be connected to all Decoders in a group.

4.3.5 Return Value

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

4.3.6 Command joinaudio 2

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

4.3.7 Return Example2

```
join ir success<cr>

join ir error [incomplete]<cr>
join ir error [join not permitted]<cr>
join ir error [invalid response]<cr>
join ir error [encoder '<encoder_device_name>' not found]<cr>
join ir error [decoder '<decoder_device_name>' not found]<cr>
join ir error [device '<device_name>' disconnected]<cr>
join ir error [group devices not found]<cr>
```

ToC

4.4 serial

4.4Command

join serial **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]**<cr>

4.4.2 Description

The command **join serial** is used for independent serial routing.

4.4

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)

4.4.4 Notes

- **all** can replace *decoder_device_name* when the serial RS232 is required to be connected to all Decoders.
- The name of a group can replace *decoder_device_name* when the serial RS232 is required to be connected to all Decoders in a group.
- Device must be in Matrix mode to allow joins.

4.4.5 Return Value

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

4.4.6 Command joinaudio 2

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

4.4.7 Return Example2

```
join serial success<cr>

join serial error [incomplete]<cr>
join serial error [not supported]<cr>
join serial error [join not permitted]<cr>
join serial error [invalid parameter]<cr>
join serial error [invalid response]<cr>
join serial error [encoder '<encoder_device_name>' not found]<cr>
join serial error [decoder '<decoder_device_name>' not found]<cr>
join serial error [device '<device_name>' disconnected]<cr>
join serial error [group devices not found]<cr>
```

ToC

4.5 usb

4.5 Command

join usb **[**key:<*security_key*>**]** <*encoder_device_name*> <decoder_device_name> / <*group_name*> / <all> **[**<exclusive>**]**<cr>

4.5.2 Description

The command **join usb** is used for independent USB routing.

4.5

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder
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 replace *decoder_device_name* when the USB is required to be connected to all Decoders.
- The name of a group can replace *decoder_device_name* when the USB is required to be connected to all Decoders in a group.

4.5.5 Return Value

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

4.5.6 Command join audio 2

```
join usb Encoder1 Decoder1<cr>
join usb Encoder1 all<cr>
join usb Encoder1 Mygroup<cr>
join usb Encoder1 Decoder1 exclusive<cr>
join usb Encoder1 Mygroup exclusive<cr>
join usb key:abc123 Encoder1 Decoder1<cr>
```

4.5.7 5 usb 2

```
join usb success<cr>

join usb error [incomplete]<cr>
join usb error [join not permitted]<cr>
join usb error [invalid parameter]<cr>
join usb error [invalid response]<cr>
join usb error [encoder '<encoder_device_name>' not found]<cr>
join usb error [decoder '<decoder_device_name>' not found]<cr>
join usb error [device '<device_name>' disconnected]<cr>
join usb error [group devices not found]<cr>
```

ToC

4.6 all

4.6 Command

join all **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]** **[**<original> | <auto> | **[**size <*video_mode*>**]]**<cr>

4.6.2 Description

The command **join all** is used for combined routing.

4.6

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)
original	Keyword '**original**' will set the Decoder resolution as pass-though to match the Encoder video resolution. (optional)
auto	Keyword '**auto**' is only available when the Decoder has a monitor connected with valid EDID. The Decoders resolution will be set to the connected display's preferred resolution. (optional)
size	Keyword '**size**' is followed by the video mode (optional)

4.6.4 Notes

- **all** can replace *decoder_device_name* when all signals are required to be connected to all Decoders.
- Only use **'original'** or **'auto'** or **'size'**.
- The name of a group can replace *decoder_device_name* when all signals are required to be connected to all Decoders in a group.
- '**auto**' = Decoder's connected displays preferred resolution
- '**original**' = Encoder source resolution
- Valid video modes:
  - 2160p60 * on compatible devices only
  - 2160p50 * on compatible devices only
  - 2160p30
  - 1080p60
  - 1080p50
  - 720p60
  - 720p50

ToC

4.6.5 Return Value

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

4.6.6 Examples

```
join all Encoder1 Decoder1<cr>
join all Encoder1 all<cr>
join all Encoder1 MyGroup<cr>
join all Encoder1 MyGroup exclusive<cr>
join all Encoder1 Decoder1 original<cr>
join all Encoder1 Decoder1 auto<cr>
join all Encoder1 Decoder1 size 1080p60<cr>
join all key:abc123 Encoder1 Decoder1<cr>
```

4.6.7 5 usb 2

```
join all success<cr>

join all error [incomplete]<cr>
join all error [join not permitted]<cr>
join all error [invalid parameter]<cr>
join all error [invalid video_mode]<cr>
join all error [monitor not detected]<cr>
join all error [unsupported monitor]<cr>
join all error [invalid response]<cr>
join all error [encoder '<encoder_device_name>' not found]<cr>
join all error [decoder '<decoder_device_name>' not found]<cr>
join all error [device '<device_name>' disconnected]<cr>
join all error [group devices not found]<cr>
```

ToC

4.7 [illegible] av

4.7 Command

join av **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]** **[**<original> | <auto> | **[**size <*video_mode*>**]]**<cr>

4.7.2 Description

The command **join av** is used for combined audio and video routing.

4.7 [illegible]

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)
original	Keyword '**original**' will set the Decoder resolution as pass-though to match the Encoder video resolution. (optional)
auto	Keyword '**auto**' is only available when the Decoder has a monitor connected with valid EDID. The Decoders resolution will be set to the connected display's preferred resolution. (optional)
size	Keyword '**size**' is followed by the video mode (optional)

4.7.4 Notes

- **all** can replace *decoder_device_name* when the audio and video are required to be connected to all Decoders.
- Only use **'original'** or **'auto'** or **'size'**.
- The name of a group can replace *decoder_device_name* when the audio and video are required to be connected to all Decoders in a group.
- '**auto**' = Decoder's connected displays preferred resolution
- '**original**' = Encoder source resolution
- Valid video modes: :
  - 2160p60 * on compatible devices only
  - 2160p50 * on compatible devices only
  - 2160p30
  - 1080p60
  - 1080p50
  - 720p60
  - 720p50

ToC

4.7.5 Return Value

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

4.7.6 Command joiExample2

```
join av Encoder1 Decoder1<cr>
join av Encoder1 all<cr>
join av Decoder1 MyGroup<cr>
join av Encoder1 MyGroup exclusive<cr>
join av Encoder1 Decoder1 original<cr>
join av Encoder1 Decoder1 auto<cr>
join av Encoder1 Decoder1 size 1080p60<cr>
join av key:abc123 Encoder1 Decoder1<cr>
```

4.7.7 5 usb 2

```
join av success<cr>

join av error [incomplete]<cr>
join av error [join not permitted]<cr>
join av error [invalid parameter]<cr>
join av error [invalid video_mode]<cr>
join av error [monitor not detected]<cr>
join av error [unsupported monitor]<cr>
join av error [invalid response]<cr>
join av error [encoder '<encoder_device_name>' not found]<cr>
join av error [decoder '<decoder_device_name>' not found]<cr>
join av error [device '<device_name>' disconnected]<cr>
join av error [group devices not found]<cr>
```

ToC

4.8 kvm

4.8 Command

join kvm **[**key:<*security_key*>**]** <*encoder_device_name*> <*decoder_device_name*> / <*group_name*> / <all> **[**<exclusive>**]** **[**<original> | <auto> | **[**size <*video_mode*>**]]**<cr>

4.8.2 Description

The command **join kvm** is used for combined audio, video and USB routing.

4.8

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)
original	Keyword '**original**' will set the Decoder resolution as pass-though to match the Encoder video resolution. (optional)
auto	Keyword '**auto**' is only available when the Decoder has a monitor connected with valid EDID. The Decoders resolution will be set to the connected display's preferred resolution. (optional)
size	Keyword '**size**' is followed by the video mode (optional)

4.8.4 Notes

- **all** can replace *decoder_device_name* when kvm is required to be connected to all Decoders.
- Only use **'original'** or **'auto'** or **'size'**.
- The name of a group can replace *decoder_device_name* when kvm is required to be connected to all Decoders in a group.
- '**auto**' = Decoder's connected displays preferred resolution
- '**original**' = Encoder source resolution
- Valid video modes:
  - 2160p60 * on compatible devices only
  - 2160p50 * on compatible devices only
  - 2160p30
  - 1080p60
  - 1080p50
  - 720p60
  - 720p50

ToC

4.8.5 Return Value

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

4.8.6 Command joinExample2

```
join kvm Encoder1 Decoder1<cr>
join kvm Encoder1 all<cr>
join kvm Encoder1 MyGroup<cr>
join kvm Encoder1 MyGroup exclusive<cr>
join kvm Encoder1 Decoder1 original<cr>
join kvm Encoder1 Decoder1 auto<cr>
join kvm Encoder1 Decoder1 size 1080p60<cr>
join kvm key:abc123 Encoder1 Decoder1<cr>
```

4.8.7 5 usb 2

```
join kvm success<cr>

join kvm error [incomplete]<cr>
join kvm error [join not permitted]<cr>
join kvm error [invalid parameter]<cr>
join kvm error [invalid video_mode]<cr>
join kvm error [monitor not detected]<cr>
join kvm error [unsupported monitor]<cr>
join kvm error [invalid response]<cr>
join kvm error [encoder '<encoder_device_name>' not found]<cr>
join kvm error [decoder '<decoder_device_name>' not found]<cr>
join kvm error [device '<device_name>' disconnected]<cr>
join kvm error [group devices not found]<cr>
```

ToC

4.9 wall

4.9.1 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*> <*display_height*> <*viewable_height*>]<cr>

4.9.2 Description

The command **join wall** is used to join an Encoder to a Decoder within a video wall layout.

4.9

encoder_device_name	Device name of the Encoder
decoder_device_name	Device name of the Decoder
wall_type	Define the video wall configuration as columns x rows
display_position	Define the display position from the top left
size (optional)	Define the display resolution
width	Display resolution horizontal in px (used with optional **size**)
height	Display resolution vertical in px (used with optional **size**)
fps	Display frame rate (used with optional **size**)
bezel (optional)	Define the size of the display bezel
display_width	Define the overall width of the display in mm (used with optional **bezel**)
viewable_width	Define the viewable width of the display in mm (used with optional **bezel**)
display_height	Define the overall height of the display in mm (used with optional **bezel**)
viewable_height	Define the viewable height of the display in mm (used with optional **bezel**)

4.9.4 Notes

- Up to 16x16 can be used for wall_type

4.9.5 Return Value

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

4.9.6 Command joinExample2

```
join wall Encoder1 Decoder1 3x3 1<cr>
join wall Encoder1 Decoder1 3x3 1 size 1920 1080 60<cr>
join wall Encoder1 Decoder1 3x3 1 bezel 1000 980 800 780<cr>
join wall Encoder1 Decoder1 3x3 1 size 1920 1080 60 bezel 1000 980 800 780<cr>
```

ToC

4.9.7 5 usb 2

```
join wall success<cr>

join wall error [incomplete]<cr>
join wall error [join not permitted]<cr>
join wall error [invalid display_position]<cr>
join wall error [invalid parameter]<cr>
join wall error [invalid size]<cr>
join wall error [invalid response]<cr>
join wall error [encoder '<encoder_device_name>' not found]<cr>
join wall error [decoder '<decoder_device_name>' not found]<cr>
join wall error [device '<device_name>' disconnected]<cr>
```

ToC

4.10 Command join usb_ext

4.10.1 Command usage

join usb_ext **[**key:<*security_key*>**]** <*lex_device_name*> <*rex_device_name*> **[**<exclusive>**]**<cr>

4.10.2 Description

The command **join usb_ext** is used for independent external USB Extender routing.

4.10

lex_device_name	Device name of Host
rex_device_name	Device name of Client
exclusive	Keyword '**exclusive**' allows for only the specified Host to be joined with the Client. All other Clients joined with the Host will be removed. (optional)

4.10.4 Notes

- Standalone Icron USB extender LEX can join with up to 7 USB REX devices.

4.10.5 Return Value

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

4.10.6 Command Example2

```
join usb_ext USBLEX1 USBREX1<cr>
join usb_ext USBLEX1 USBREX1 exclusive<cr>
join usb_ext key:abc123 USBLEX1 USBREX1<cr>
```

4.10.7 5 Example2

```
join usb_ext success<cr>

join usb_ext error [incomplete]<cr>
join usb_ext error [invalid parameter]<cr>
join usb_ext error [join failed]<cr>
join usb_ext error [lex '<lex_device_name>' not found]<cr>
join usb_ext error [rex '<rex_device_name>' not found]<cr>
join usb_ext error [device '<device_name>' disconnected]<cr>
```

ToC

5 Command leave

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

Commands missing **mode** return:

```
leave error [incomplete]<cr>
```

Commands with invalid **mode** return:

```
leave error [invalid mode]<cr>
```

5 Command leave video

5.6 Command

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

5.6.2 Description

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

5.6

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

5.6.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave video subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave video subscriptions.

5.6.5 Return Value

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

5.6.6 Command join Example2

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

5.6 Return Example2

```
leave video success<cr>

leave video error [incomplete]<cr>
leave video error [invalid response]<cr>
leave video error [decoder '<decoder_device_name>' not found]<cr>
leave video error [device '<decoder_device_name>' disconnected]<cr>
leave video error [group devices not found]<cr>
```

ToC

5.2 leave

5.2 Command

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

5.2.2 Description

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

5.2

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

5.2.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave audio subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave audio subscriptions.

5.2.5 Return Value

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

5.2.6 Command join Example2

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

5.2.Return Example2

```
leave audio success<cr>

leave audio error [incomplete]<cr>
leave audio error [invalid response]<cr>
leave audio error [decoder '<decoder_device_name>' not found]<cr>
leave audio error [device '<decoder_device_name>' disconnected]<cr>
leave audio error [group devices not found]<cr>
```

ToC

5.3. leave ir

5.1 Command

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

5.2 Description

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

5.3 Parameters

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

5.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave infrared subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave infrared subscriptions.

5.5 Return Value

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

5.6 Command join Example2

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

5.7 Return Example2

```
leave ir success<cr>

leave ir error [incomplete]<cr>
leave ir error [invalid response]<cr>
leave ir error [decoder '<decoder_device_name>' not found]<cr>
leave ir error [device '<decoder_device_name>' disconnected]<cr>
leave ir error [group devices not found]<cr>
```

ToC

5.4 Command leave serial

5.4 Command

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

5.4.2 Description

The command **leave serial** is used to disconnect a Decoder from the serial RS232 stream it is receiving.

5.4

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

5.4.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave serial subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave serial subscriptions.
- Device must be in Matrix mode to allow joins.

5.4.5 Return Value

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

5.4.6 Command join Example2

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

5.4 Return Example2

```
leave serial success<cr>

leave serial error [incomplete]<cr>
leave serial error [not supported]<cr>
leave serial error [invalid response]<cr>
leave serial error [decoder '<decoder_device_name>' not found]<cr>
leave serial error [device '<decoder_device_name>' disconnected]<cr>
leave serial error [group devices not found]<cr>
```

ToC

5.5 leave

5.5 Command

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

5.5.2 Description

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

5.5

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

5.5.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave USB subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave USB subscriptions.

5.5.5 Return Value

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

5.5.6 Command join Example

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

5.5.Return 5 Example2

```
leave usb success<cr>

leave usb error [incomplete]<cr>
leave usb error [invalid response]<cr>
leave usb error [decoder '<decoder_device_name>' not found]<cr>
leave usb error [device '<decoder_device_name>' disconnected]<cr>
leave usb error [group devices not found]<cr>
```

ToC

5.6 Command leave all

5.6Command

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

5.6.2

The command **leave all** is used to disconnect a Decoder from all the streams it is receiving.

5.6

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

5.6.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave all subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group 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 joinExample

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

5.6.Return5 Example2

```
leave all success<cr>

leave all error [incomplete]<cr>
leave all error [invalid response]<cr>
leave all error [decoder '<decoder_device_name>' not found]<cr>
leave all error [device '<decoder_device_name>' disconnected]<cr>
leave all error [group devices not found]<cr>
```

ToC

5.6 Command leave all

5.1 Command

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

5.2 Description

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

5.3

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

5.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave audio and video subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave audio and video subscriptions.

5.5 Return Value

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

5.6 Command join Example

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

5.7 Return Example2

```
leave av success<cr>

leave av error [incomplete]<cr>
leave av error [invalid response]<cr>
leave av error [decoder '<decoder_device_name>' not found]<cr>
leave av error [device '<decoder_device_name>' disconnected]<cr>
leave av error [group devices not found]<cr>
```

ToC

5 leave kvm

5.1 Command

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

5.2 Description

The command **leave kvm** is used to disconnect a Decoder from the audio, video and USB streams it is receiving.

5.3

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

5.4 Notes

- **all** can replace *decoder_device_name* when all the Decoders are required to leave audio, video and USB subscriptions.
- The name of a group can replace *decoder_device_name* when all Decoders in a group are required to leave audio, video and USB subscriptions.

5.5 Return Value

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

5.6 Command join Example

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

5.7 Return Example2

```
leave kvm success<cr>

leave kvm error [incomplete]<cr>
leave kvm error [invalid response]<cr>
leave kvm error [decoder '<decoder_device_name>' not found]<cr>
leave kvm error [device '<decoder_device_name>' disconnected]<cr>
leave kvm error [group devices not found]<cr>
```

ToC

5.9 leave usb_ext

5.9 Command

leave usb_ext **[**key:<*security_key*>**]** <*rex_device_name*><cr>

5.9.2 Description

The command **leave usb_ext** is used to disconnect an external USB Extender Client from the USB Host.

5.9

rex_device_name	Name of the Client

5.9.4 Notes

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

5.9.5 Return Value

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

5.9.6 Command join Example

```
leave usb_ext USBCLIENT1<cr>
leave usb key:abc123 USBCLIENT1<cr>
```

5.9 Return Example2

```
leave usb_ext success<cr>

leave usb_ext error [incomplete]<cr>
leave usb_ext error [leave failed]<cr>
leave usb_ext error [rex '<rex_device_name>' not found]<cr>
leave usb_ext error [device '<rex_device_name>' disconnected]<cr>
```

ToC

6 Command stop

The **stop** command is used to stop all Encoder streams from being sent on the network. Joins are maintained between Encoder and Decoders but no data is sent from the Encoder.

Commands missing **mode** return:

```
stop error [incomplete]<cr>
```

Commands with invalid **mode** return:

```
stop error [invalid mode]<cr>
```

ToC

6 Section stop

6.6 Command

stop **[**key:<*security_key*>**]** <*mode*> <*encoder_device_name*> / <*group_name*> / <all_tx><cr>

6.6.2 Description

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

6.6 S

mode	Always '**all**'
encoder_device_name	Device name of the Encoder or Group or '**all_tx**'

6.6.4 Notes

- The name of a group can replace *encoder_device_name* when all Encoders in a group are required to stop.
- All stream types are started together.
- A streaming connection can be made again with the command **join** or **start**.

6.6.5 Return Value

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

6.6.6 Command join Examples

```
stop all Encoder1<cr>
stop all MyGroup<cr>
stop all all_tx<cr>
stop key:abc123 all Encoder1<cr>
```

6.6.Return 5 Examples

```
stop all success<cr>

stop all error [incomplete]<cr>
stop all error [invalid response]<cr>
stop all error [encoder '<encoder_device_name>' not found]<cr>
stop all error [device '<encoder_device_name>' disconnected]<cr>
stop all error [group devices not found]<cr>
```

ToC

7 Command start

The **start** command is used to start all Encoder streams.

Commands missing **mode** return:

```
start error [incomplete]<cr>
```

Commands with invalid **mode** return:

```
start error [invalid mode]<cr>
```

ToC

7 Direction start

6 Command

start **[**key:<*security_key*>**]** <*mode*> <*encoder_device_name*> / <*group_name*> / <all_tx><cr>

6.2 Description

The command **start** is used to start all Encoder streams.

6

mode	Always 'all'
encoder_device_name	Device name of the Encoder or Group or '**all_tx**'

6.4 Notes

- A stream cannot be joined unless it has started, so a **join** command will automatically send the **start** command if the Encoder streams are stopped.
- All stream types are started together.
- The name of a group can replace *encoder_device_name* when all Encoders in a group are required to start all streams.

6.5 Return Value

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

6.6 Command join Examples

```
start all Encoder1<cr>
start all MyGroup<cr>
start all all_tx<cr>
start key:abc123 all Encoder1<cr>
```

6.5 Examples

```
start all success<cr>

start all error [incomplete]<cr>
start all error [invalid response]<cr>
start all error [encoder '<encoder_device_name>' not found]<cr>
start all error [device '<encoder_device_name>' disconnected]<cr>
start all error [group devices not found]<cr>
```

ToC

8 Command set

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

Commands missing **mode** return:

```
set error [incomplete]<cr>
```

Commands with invalid **mode** return:

```
set error [invalid mode]<cr>
```

ToC

8.1 Command set for devices

This sections contains set commands for devices such as Encoders and Decoders.

1 Command set audio_source

1.6.1 Command usage

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

1.6.2 Description

The command **set audio_source** is used to change a Encoders audio stream from HDMI audio or analog audio.

1.6

encoder_device_name	Device name of the Encoder
value	hdmi | analog | auto

1.6.4 Notes

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

1.6

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

1.6.6 Command Examples

```
set audio_source Encoder1 hdmi<cr>
set audio_source Encoder1 analog<cr>
set audio_source key:abc123 Encoder1 auto<cr>
```

1.6. Examples

```
set audio_source success<cr>

set audio_source error [incomplete]<cr>
set audio_source error [invalid value '<value>']<cr>
set audio_source error [invalid response]<cr>
set audio_source error [encoder '<encoder_device_name>' not found]<cr>
set audio_source error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

1 Command 2 Command set edid

101.2.1 Command usage

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

101.2.2 Description

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

101.2 S

encoder_device_name	Device name of the Encoder or Group or '**all_tx**'
edid_data	String which represents the binary EDID data

101.2.4 Notes

- **all_tx** 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.

101.2leave

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

101.2.6 Command Examples

```
set edid Encoder1
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818
0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000
00fd00384c1e5315000a20202020202000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131
41f2309070783010000065030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c110000
18011d007251d01e206e285500dc0c1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000
000000000000000000000000ac<cr>

set edid key:abc123 MyGroup
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818
0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000
00fd00384c1e5315000a20202020202000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131
41f2309070783010000065030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c110000
18011d007251d01e206e285500dc0c1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000
000000000000000000000000ac <cr>


set edid all_tx
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818
0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000
00fd00384c1e5315000a20202020202000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131
41f2309070783010000065030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c110000
18011d007251d01e206e285500dc0c1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000
000000000000000000000000ac<cr>
```

ToC

10.1.2. Return Examples

```
set edid success<cr>

set edid error [incomplete]<cr>
set edid error [invalid edid_data]<cr>
set edid error [invalid response]<cr>
set edid error [encoder '<encoder_device_name>' not found]<cr>
set edid error [device '<encoder_device_name>' disconnected]<cr>
set edid error [group devices not found]<cr>
```

ToC

3. Command set frame_converter

101.all1 Command usage

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

101.all2 Description

The command **set frame_converter** is used to change the Encoders video frame rate.

101.all

encoder_device_name	Device name of the Encoder
value	0..59

101.all4 Notes

- Value 0 is used to set the original source frame rate.
- Use the command **get frame_converter** to retrieve this setting.

101.all

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

101.all6 Command Examples

```
set frame_converter Encoder1 0<cr>
set frame_converter key:abc123 Encoder1 30<cr>
```

101.all Examples

```
set frame_converter success<cr>

set frame_converter error [incomplete]<cr>
set frame_converter error [invalid value '<value>']<cr>
set frame_converter error [invalid response]<cr>
set frame_converter error [encoder '<encoder_device_name>' not found]<cr>
set frame_converter error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

101.4 Command set rotation

101.4.1 Command usage

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

105.2 Description

The command **set rotation** is used to rotate the video for displays that have been rotated.

101.4

decoder_device_name	Device name of the Decoder
value	0..7

101.4.4 Notes

- Valid values:
  - 0 = none
  - 1 = vertical mirror
  - 2 = horizontal mirror
  - 3 = 180º rotation
  - 4 = 90º clockwise rotation + horizontal mirror
  - 5 = 90º clockwise rotation
  - 6 = 270º clockwise rotation
  - 7 = 270º clockwise rotation + horizontal mirror

101.4

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

101.4.6 Command Examples

```
set rotation Decoder1 0<cr>
set rotation key:abc123 Decoder1 5<cr>
```

101.4. Examples

```
set rotation success<cr>

set rotation error [incomplete]<cr>
set rotation error [invalid value '<value>']<cr>
set rotation error [invalid response]<cr>
set rotation error [decoder '<decoder_device_name>' not found]<cr>
set rotation error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

101.5 Command set scaler

101.5.1 Command usage

set scaler **[**key:<*security_key*>**]** <*decoder_device_name*> <*mode*><cr>

101.5.2 Description

The command **set scaler** is used to change the Decoders video output resolution.

101.5 Set

decoder_device_name	Device name of the Decoder
mode	auto | original | 2160p60 .. 720p50

101.5.4 Notes

- Valid mode values:
  - auto = Decoder's connected displays preferred resolution
  - original = Encoder source video resolution
  - 2160p60 * on compatible devices only
  - 2160p50 * on compatible devices only
  - 2160p30
  - 1080p60
  - 1080p50
  - 720p60
  - 720p50

101.5 Leave

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

101.5.6 Command Examples

```
set scaler Decoder1 auto<cr>
set scaler Decoder1 original<cr>
set scaler Decoder1 1080p60<cr>
set scaler key:abc123 Decoder1 2160p30<cr>
```

ToC

10.1.5. Return Examples

```
set scaler success<cr>

set scaler error [incomplete]<cr>
set scaler error [invalid mode '<mode>']<cr>
set scaler error [monitor not detected]<cr>
set scaler error [invalid response]<cr>
set scaler error [decoder '<decoder_device_name>' not found]<cr>
set scaler error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

101.6 set scalerset video_mute

101.6.1 Command usage

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

101.6.2 Description

The command **set video_mute** is used to disable the Decoders video output and leave a black screen.

101.6

decoder_device_name	Device name of the Decoder
state	true | false

101.6.4 Notes

101.6leave

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

101.6.6 Command Examples

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

101.6.Return Examples

```
set video_mute success<cr>

set video_mute error [incomplete]<cr>
set video_mute error [invalid state '<state>']<cr>
set video_mute error [invalid response]<cr>
set video_mute error [decoder '<decoder_device_name>' not found]<cr>
set video_mute error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

101.7 set scalerset video_quality

101.7.1 Command usage

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

101.7.2 Description

The command **set video_quality** is used to manage network bandwidth by changing the video stream quality.

101.7

encoder_device_name	Device name of the Encoder
value	-1..5

101.7.4 Notes

- Valid values:
  - -1 = auto
  - 0 = minimum
  - 1
  - 2
  - 3
  - 4
  - 5 = maximum

101.7leave

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

101.7.6 Command Examples

```
set video_quality Encoder1 -1<cr>
set video_quality key:abc123 Encoder1 5<cr>
```

101.7.Return 5 Examples

```
set video_quality success<cr>

set video_quality error [incomplete]<cr>
set video_quality error [invalid value '<value>']<cr>
set video_quality error [invalid response]<cr>
set video_quality error [encoder '<encoder_device_name>' not found]<cr>
set video_quality error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

10.8 set scaler set volume

10.8.1 Command usage

set volume **[**key:<*security_key*>**]** <*device_name*> <*value*><cr>

10.8.2 Description

The command **set volume** is used to set the volume level of the analog audio.

10.8.3 Syntax

device_name	Device name of the Encoder or Decoder
value	0..100

10.8.4 Notes

10.8.5 Return Value

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

10.8.6 Command Examples

```
set volume Encoder1 0<cr>
set volume key:abc123 Decoder1 5<cr>
```

10.8.Return 5 Examples

```
set volume success<cr>

set volume error [incomplete]<cr>
set volume error [invalid value '<value>']<cr>
set volume error [invalid response]<cr>
set volume error [device '<device_name>' not found]<cr>
set volume error [device '<device_name>' disconnected]<cr>
```

ToC

8.2 Command set for system

This sections contains set commands for the system.

8.2.1 set scalerset events

8.2.1.1 Command usage

set events [key:<security_key>] <*event_name*> <*function*> <*value*><cr>

8.2.1.2 Description

The command **set events** is used to enable or disable events created on the UI's Scheduler and Events tab.

8.2.1.3

event_name	Name of the event
function	Current only "**state**" supported
value	"**enabled**" or "**disabled**"

8.2.1.4 Notes

- Event must be created on UI's Scheduler or Events tab before using command.

8.2.1.5

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

8.2.1.6 Command Examples

```
set events MyEvent enabled<cr>
set events MyEvent disabled<cr>
```

8.2.1.7 Examples

```
set events success<cr>

set events error [incomplete]<cr>
set events error [event '<event_name>' not found]<cr>
set events error [invalid parameter]<cr>
```

ToC

102.2 set scalerset listener

102.2.1 Command usage

set listener **[**key:<*security_key*>**]** <*notify_ip*> <*notify_port*> <*protocol*> **[**<*device_ip*>**]** <*condition*> <*state*> <*device_io*> <*preset_name*> **[***delay***]**<cr>

102.2.2 Description

The command **set listener** utilises Global Caché device sensor notify functionality. This enables a preset to be triggered from a sensor notify beacon sent when a devices input status changes from a switch or PIR.

102.2.3 Arguments

notify_ip	239.255.250.250
notify_port	9160
protocol	UDP
device_ip	IP Address of the device
condition	'on' | 'off' | 'any'
state	'enabled' | 'disabled'
device_io	Physical Global Caché device input sensor port 1 to 6
preset_name	Name of the preset to be executed
delay (optional)	Optional delay time of preset execution in minutes

102.2.4 Notes

- Supported Global Caché devices: IP2IR, WF2IR, GCIR3, Flex Link Relay & Sensor Cable.
- "***condition***" is the input as '**on**' (closed contact) or '**off**' (open contact) or '**any**' (contact closing or opening)
- "***state***" is the running state of the set listener service as '**enabled**' or '**disabled**'.
- Optional "***delay***" is used to delay the preset for the specified amount of time in minutes. The delay is reset each time the command is used.

102.2leave

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

102.2.6 Command Examples

```
set listener 239.255.250.250 9160 udp 172.30.10.222 on enabled 1 preset2 30<cr>
set listener 239.255.250.250 9160 udp 172.30.10.222 on disabled 1<cr>
set listener key:abc123 239.255.250.250 9160 udp 172.30.10.222 on enabled 1 preset1<cr>
```

ToC

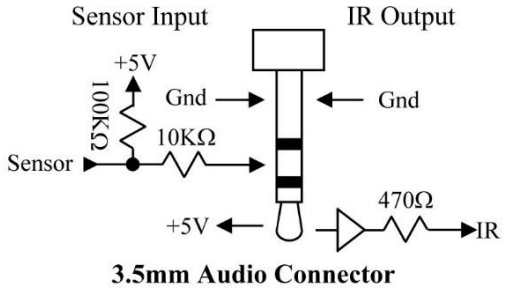
102.2.Return 5 Examples

```
set listener success<cr>

set listener error [incomplete]<cr>
set listener error [duplicate]<cr>
set listener error [no active listener found]<cr>
set listener error [invalid notify ip]<cr>
set listener error [invalid notify port]<cr>
set listener error [reserved notify port]<cr>
set listener error [invalid protocol]<cr>
set listener error [invalid device ip]<cr>
set listener error [invalid condition]<cr>
set listener error [invalid state]<cr>
set listener error [invalid device io]<cr>
set listener error [invalid delay]<cr>
set listener error [preset 'preset' not found]<cr>
```

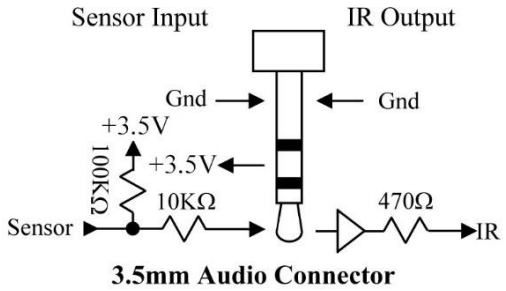
ToC

102.2.10 GC-100 Series IR/Sensor Connector Wiring



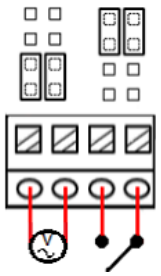
Note: Use connector RING and SLEEVE for closed contact detection

102.2.9 iTach and GlobalConnect Series IR/Sensor Connector Wiring



Note: Use connector TIP and SLEEVE for closed contact detection

102.2.10 Flex Link Relay & Sensor Cable Wiring



Note: Check jumper positions for either voltage or closed contact detection

ToC

102.2.11 Technical Notes

Global Cache sensor cables

GC-100 series

- GC-SV1 Video Out Sensor
- GC-SP1 AC/DC Voltage Sensor
- GC-SC1 Contact Closure Sensor *

IP2IR, WF2IR and GCIR3

- IT-SV1 Video Out Sensor
- IT-SP1 AC/DC Voltage Sensor
- IT-SC1 Contact Closure Sensor *

Flex series

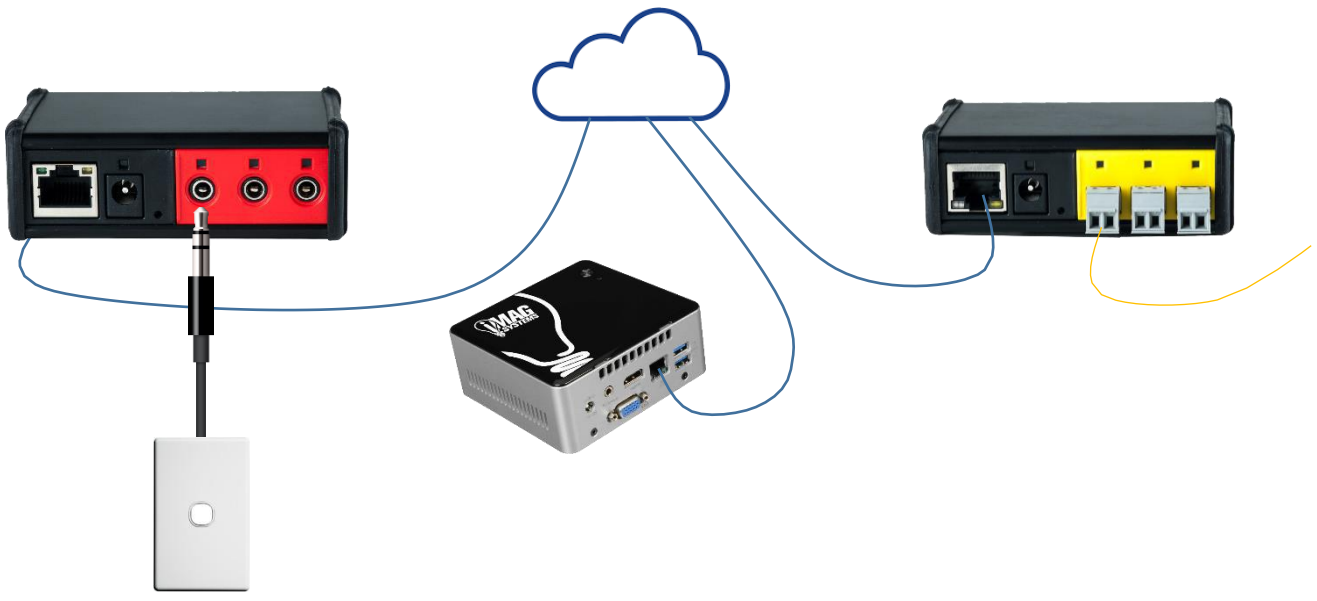
- Not required

* For closed contact detection a GC-SC1 or IT-SC1 Contact Closure Sensor cable is not really required if you can make your own. With a 3.5mm stereo phono connector wired with tip and sleeve or ring and sleeve depending on the model as described above in 3.1.8 and 3.1.9.

IR outputs and sensor inputs share a common connector and LED indicator. Each 3.5mm audio connector is independently configured using the assistant. Each connector has three contacts configured as either an infrared (IR) output or sensor input.

When configured as an output, the indicator will blink as an IR command is transmitted. When functioning as a sensor, the indicator is "on" when a positive input or no connection is present. The maximum sensor input voltage is ±24V, with an "on" indication for voltages greater than 2.5V and "off" when less than 0.8V with an input impedance of ~100KΩ.

Here is a simple example of a system using an iTach IP2IR connected to a push button to activate an iTach IP2CC relay.



ToC

2.3 set scaleset var

2.3.1 Command usage

set var **[**key:<*security_key*>**]** <*var_name*> <*value*> [delete]<cr>

2.3.2 Description

The command **set var** is used to store any user defined variable values.

2.3 Arguments

var_name	Name of the variable
value	String or boolean true false
[delete]	Keyword used to delete the variable from the system

2.3.4 Notes

- *var_name* and value can be any string up to 256 characters.
- When [delete] is used as a value the variable will be deleted.
- String values with spaces are to be wrapped in "quotation marks".

2.3 leave

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

2.3.6 Command Examples

```
set var MyVar true<cr>
set var MyVar "any string up to 256"<cr>
set var MyVar [delete]<cr>
set var key:abc123 MyVar false<cr>
```

2.3.5 Return Examples

```
set var success<cr>

set var error [incomplete]<cr>
set var error [var_name max 256 characters]<cr>
set var error [value max 256 characters]<cr>
```

ToC

8.3 Command set for User Interfaces

This sections contains set commands for interaction with User Interfaces.

After controller start-up, User Interface services are not enabled until devices have been discovered. This ensures devices are available for initial preset and UI use. To force the UI service to run without devices, the UI service firstly needs to be disabled then enabled again.

ToC

3.1 set scalerset ui

3.1.1 Command usage

set ui [key:<security_key>] <*ui_name*> <*service*> [timeout <*timeout*>] [clients <*clients*>] [login <*pin*>]<cr>

3.1.2 Description

The command **set ui** is used to enable and disable a User Interface service.

3.1

ui_name	name of the User Interface
service	'enabled' | 'disabled' | 'logout'
clients	Optional with enabled service to set the maximum client limit from 1 to 100
pin	Optional with enabled service to set a fixed or random 4 digit login pin code

3.1.4 Notes

- Control UI must be enabled as a feature for command to be used.
- Service logout is the same as disabled then enabled.

3.1

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

3.1.6 Command Examples

```
set ui myUI disabled<cr>
set ui myUI logout<cr>
set ui myUI enabled<cr>
set ui myUI enabled clients 10 login 1234<cr>
set ui key:abc123 myUI enabled clients 10 login random<cr>
```

3.1. Examples

```
set ui success<cr>

set ui error [incomplete]<cr>
set ui error [ui '<ui_name>' not found]<cr>
set ui error [invalid clients '<value>']<cr>
set ui error [invalid pin '<value>']<cr>
set ui error [invalid parameter]<cr>
set ui error [service disabled]<cr>
```

ToC

10.3.2 set scalerset ui_button

10.3.2.1 Command usage

set ui_button [key:<security_key>] <*ui_name*> <*button_name / button_group*> <*function*> [<*button_position*>] [<*value*>]<cr>

10.11Example

The command **set ui_button** is used to control a button within an active User Interface.

10.3.2

ui_name	Name of the User Interface
button_name / button_group	Name of the button or preset logic <<button_name>> Or name of a button group
function	position | state | text | press
button_position	up | down | both
value	up | down, enabled | disabled or text string depending on the function

10.3.2.4 Notes

- Control UI must be enabled as a feature for command to be used.
- The UI service must be enabled.
- Used with split buttons, <<encoder>> and <<decoder>> can be used for button text <value>.
- Used within a preset, <<button_name>> can be used in place of <button_name>.
- For button groups, only **position** and **state** functions are available. Radio Toggle buttons can only use **position up**.
- **function** > **position** and **text** uses **up** |**down** | **both**
- **function** > **position** is only valid for Toggle, Radio Toggle and Split button types.
- **function** > **state** uses **enabled** | **disabled**
- **button_position** is only required for function text for Toggle and Radio Toggle and Split button types otherwise 'up' can only be used.

- **Momentary**
  - state
  - text
  - press
- **Toggle**
  - position
  - state
  - text
  - press
- **Radio Toggle**
  - position
  - state
  - text
  - press
- **Split**
  - state
  - text
  - press
- **Repeat**
  - state
  - text
  - press

10.3.2leave

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

10.3.2.6 Command Examples

```
set ui_button myUI myButton position up<cr>
set ui_button myUI myButton position down<cr>
set ui_button myUI myButton state enabled<cr>
set ui_button myUI myButton state disabled<cr>
set ui_button myUI myButton text up "MyText"<cr>
set ui_button myUI MyButtonGroup position up<cr>
set ui_button myUI MyButtonGroup state disabled<cr>
set ui_button myUI myButton text down <<encoder>><cr>
set ui_button myUI <<button_name>> text both <<encoder>><cr>
set ui_button key:abc123 myUI myButton press<cr>
```

ToC

103.2.Return 5 Examples

```
set ui_button success<cr>

set ui_button error [incomplete]<cr>
set ui_button error [ui '<ui_name>' not found]<cr>
set ui_button error [button '<button_name / button_group>' not found]<cr>
set ui_button error [invalid parameter]<cr>
set ui_button error [invalid button_position]<cr>
set ui_button error [service disabled]<cr>
set ui_button error [invalid button type]<cr>
```

ToC

103.3 set scalerset ui_image

103.3.1 Command usage

set ui_image [key:<security_key>] <*ui_name*> <*image_name*> <*function*> <*value*><cr>

103.3.2 Description

The command **set ui_image** is used to control an image within an active User Interface.

103.3

ui_name	Name of the User Interface
image_name	Name of the image
function	visibility
value	true | false

103.3.4 Notes

- Control UI must be enabled as a feature for command to be used.
- The UI service must be enabled.

103.3leave

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

103.3.6 Command Examples

```
set ui_image myUI myImage visibility false<cr>
set ui_image key:abc123 myUI myImage visibility true<cr>
```

103.3.Return 5 Examples

```
set ui_image success<cr>

set ui_image error [incomplete]<cr>
set ui_image error [ui '<ui_name>' not found]<cr>
set ui_image error [image '<image_name>' not found]<cr>
set ui_image error [invalid parameter]<cr>
set ui_image error [service disabled]<cr>
```

ToC

8.3.4 Command set ui_indicator

8.3.4.1 Command usage

set ui_indicator [key:<security_key>] <*ui_name*> <*indicator_name*> <*function*> <*value*><cr>

8.3.4..2 Description

The command **set ui_indicator** is used to control a level indicator within an active User Interface.

8.3.4..3 Arguments

ui_name	Name of the User Interface or preset logic <<ui_name>>
indicator_name	Name of the indicator
function	value
value	-1000 ~ 1000 set by the min & max values from UI setting or preset logic <<slider_value>>

8.3.4..4 Notes

- The UI service must be enabled.
- A combined level slider will be update at the same time.
- Used within a preset, <<ui_name>> can be used in place of <ui_name>.
- Used within a preset, <<slider_value>> can be used in place of <value>.

8.3.4..5 Return Value

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

8.3.4..6 Command Examples

```
set ui_indicator myUI myIndicator value 0<cr>
set ui_indicator key:abc123 myUI myIndicator value 100<cr>
set ui_indicator <<ui_name>> myIndicator value <<slider_value>>
```

8.3.4..Return5 Example2

```
set ui_indicator value success<cr>

set ui_indicator error [incomplete]<cr>
set ui_indicator error [invalid parameter]<cr>
set ui_indicator error [invalid value '<value>']<cr>
set ui_indicator error [service disabled]<cr>
set ui_indicator error [ui '<ui_name>' not found]<cr>
set ui_indicator error [indicator '<indicator_name>' not found]<cr>
```

ToC

8.3.5 Command set ui_label

103.5.1 Command usage

set ui_label [key:<security_key>] <*ui_name*> <*label_name*> <*function*> <*value*><cr>

103.5.2 Description

The command **set ui_label** is used to control a label within an active User Interface.

103.5 S

ui_name	Name of the User Interface
label_name	Name of the label
function	color | visibility | text
value	depending on the function

103.5.4 Notes

- Control UI must be enabled as a feature for command to be used.
- The UI service must be enabled.
- When used with split buttons, <<encoder>> and <<decoder>> can be used for label text.
- function color uses HEX RGB color code from 000000 to FFFFFF
- function visibility uses true | false

103.5leave

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

103.5.6 Command Example2

```
set ui_label myUI myLabel color 000000<cr>
set ui_label myUI myLabel visibility false<cr>
set ui_label myUI myLabel visibility true<cr>
set ui_label myUI myLabel text "MyText"<cr>
set ui_label key:abc123 myUI myLabel text <<encoder>><cr>
```

103.5.Return5 Example2

```
set ui_label success<cr>

set ui_label error [incomplete]<cr>
set ui_label error [ui '<ui_name>' not found]<cr>
set ui_label error [label '<label_name>' not found]<cr>
set ui_label error [invalid parameter]<cr>
set ui_label error [service disabled]<cr>
```

ToC

8.3.5 Command set ui_label

103.6.1 Command usage

set ui_page [key:<security_key>] <*ui_name*> <*page_name*><cr>

103.6.2 Description

The command **set ui_page** is used to change the displayed page in active User Interface.

103.6 S

ui_name	Name of the User Interface
page_name	Name of the page

103.6.4 Notes

- Control UI must be enabled as a feature for command to be used.
- The UI service must be enabled.

103.6leave

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

103.6.6 Command Example2

```
set ui_page myUI myPage<cr>
set ui_page key:abc123 myUI Page2<cr>
```

103.6.Return 5 Example2

```
set ui_page success<cr>

set ui_page error [incomplete]<cr>
set ui_page error [ui '<ui_name>' not found]<cr>
set ui_page error [page '<page_name>' not found]<cr>
set ui_page error [service disabled]<cr>
```

ToC

8.3.7 Command set ui_redirect

8.3.7.1 Command usage

set ui_redirect [key:<security_key>] <*current_ui_name*> <*redirected_ui_name*> [<*page_name*>]<cr>

8.3.7.2 Description

The command **set ui_redirect** is used to change a clients current User Interface. Ideal for combining rooms where two separate User Interfaces are used, but once the rooms are combined a common User Interface is required.

8.3.7 S

current_ui_name	Name of the current User Interface
redirect_ui_name	Name of the replacement User Interface
page_name	Optional page selection of the replacement User Interface

8.3.7.4 Notes

- The UI service must be enabled.
- Refer command set ui_revert.

8.3.7leave

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

8.3.7.6 Command Example2

```
set ui_redirect myUI myNewUi<cr>
set ui_redirect key:abc123 myUI myNewUi page1<cr>
```

8.3.7.Return5 Example2

```
set ui_redirect success<cr>

set ui_redirect error [incomplete]<cr>
set ui_redirect error [duplicate]<cr>
set ui_redirect error [service disabled]<cr>
set ui_redirect error [ui '<ui_name>' not found]<cr>
set ui_redirect error [page '<page_name>' not found]<cr>
```

ToC

8.3.8 Command set ui_revert

8.3.8.1 Command usage

set ui_revert [key:<security_key>] <*original_ui_name*><cr>

8.3.8.2 Description

The command **set ui_revert** is used to revert a clients User Interface to the original, before a redirect.

8.3.8 Syntax

original_ui_name	Name of the original redirected User Interface

8.3.8.4 Notes

- The UI service must be enabled.
- Refer command set ui_redirect.

8.3.8 Leave

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

8.3.8.6 Command Example2

```
set ui_revert myUI<cr>
set ui_revert key:abc123 myUi<cr>
```

8.3.8. Return Example2

```
set ui_revert success<cr>

set ui_revert error [incomplete]<cr>
set ui_revert error [service disabled]<cr>
set ui_revert error [ui '<original_ui_name>' not found]<cr>
```

ToC

8.3.9 Command set ui_slider

8.3.9.1 Command usage

set ui_slider [key:<security_key>] <*ui_name*> <*slider_name*> <*function*> <*value*><cr>

8.3.9.2 Description

The command **set ui_slider** is used to control a level slider within an active User Interface.

8.3.9 S

ui_name	Name of the User Interface or preset logic <<ui_name>>
indicator_name	Name of the slider
function	value | state
value	-1000 ~ 1000 (set by the min & max values from UI setting) or preset logic <<slider_value>>
state	enabled | disabled

8.3.9.4 Notes

- The UI service must be enabled.
- A combined level indicator will be update at the same time.
- Used within a preset, <<ui_name>> can be used in place of <ui_name>.
- Used within a preset, <<slider_value>> can be used in place of <value>.

8.3.9leave

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

8.3.96 Command Example2

```
set ui_slider myUI mySlider value 0<cr>
set ui_slider key:abc123 myUI mySlider state disabled<cr>
set ui_slider <<ui_name>> mySlider value <<slider_value>>
```

8.3.9Return 5 Example2

```
set ui_slider value success<cr>
set ui_slider state success<cr>

set ui_slider error [incomplete]<cr>
set ui_slider error [invalid value 'xxx']<cr>
set ui_slider error [invalid parameter]<cr>
set ui_slider error [service disabled]<cr>
set ui_slider error [ui '<ui_name>' not found]<cr>
set ui_slider error [slider '<slider_name>' not found]<cr>
```

ToC

9 Command get

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

Commands missing **mode** return:

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

Commands with invalid **mode** return:

```
get error [invalid mode]<cr>
```

ToC

9.1 Command get for devices

This sections contains get commands for devices such as Encoders and Decoders.

9.1 Command get audio_source

9.1.1 Command usage

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

9.1.2 Description

The command **get audio_source** is used to retrieve the source of an Encoder's audio stream.

9.1.3

encoder_device_name	Device name of the Encoder

9.1.4 Notes

- Returned value is either **hdmi**, **analog** or **auto.**
- Use the command **set audio_souce** to change this setting.

9.1.5

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

9.1.6 Command Example

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

9.1.7 Return Examples

```
get audio_source success hdmi<cr>
get audio_source success analog<cr>
get audio_source success auto<cr>

get audio_source error [incomplete]<cr>
get audio_source error [encoder '<encoder_device_name>' not found]<cr>
get audio_source error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

9.1.2 Command get devices

9.1.2.1 Command usage

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

9.1.2.2 Description

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

9.1.2.3

target	all | all_rx | all_tx | all_rex | all_lex

9.1.2.4

- Return value <device_name> = name of device
- Return value <device_id> = device MAC Address
- all_rex and all_lex are for USB Extenders
- Success will return a json formatted string

9.1.2.5 leave

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

9.1.2.6 Command Example

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

9.1.2.7 Return Examples

```
get devices success {"<device_name>":"<device_id>","<device_name>":"<device_id>"}<cr>
get devices success {"<device_name>":"<device_id>"}<cr>
get devices success {}<cr>

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

ToC

Command 1 to 3 set scalerget display_status

9.1 all1 Command usage

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

9.1 all2 Description

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

9.1 all3

decoder_device_name	Device name of the Decoder

9.1 all7

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

9.1 all eave

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

9.1 all6 Command Example

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

9.1 allReturn5 Example2

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

get display_status error [incomplete]<cr>
get display_status error [decoder '<decoder_device_name>' not found]<cr>
get display_status error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

Command part 4 Command get edid

9.14.1 Command usage

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

9.14.2 Description

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

9.14 S

decoder_device_name	Device name of the Decoder

9.14 7

- 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.

9.14leave

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

9.14.6 Command Example

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

9.14.7 Return Examples

```
get edid success
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f8180818a9500950fb3000
101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731373137000000fd00384c1e531500
0a20202020202000000fc005068696c69707320323231450a01b602031ef04b10050102030406121314 1f230907078301000
065030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e20
6e285500dc0c1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000000000000000000
00000ac<cr>

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

ToC

9.15 Command get joins

9.15.1 Command usage

get joins **[**key:<*security_key*>**]** <*device_name*> <*subscription*><cr>

9.15.2 Description

The command **get joins** is used to retrieve the Encoder name subscribed to a Decoder's subscription, or USB Extender LEX paired with USB Extender REX.

9.15

device_name	Device name of a Decoder or USB Extender REX
subscription	video | audio | serial | ir | usb | usb_ext

9.15

- No join will return **null**.
- Success will return a json formatted string.

9.15leave

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

9.15.6 Command Example2

```
get joins Decoder1 video<cr>
get joins Decoder1 audio<cr>
get joins Decoder1 serial<cr>
get joins Decoder1 ir<cr>
get joins Decoder1 usb<cr>
get joins Decoder1 usb_ext<cr>
get joins key:abc123 Decoder1 video<cr>
```

9.15Return Example2

```
get joins success {"<device_name>":"<encoder>"}<cr>
get joins success {"Decoder1":"Encoder1"}<cr>
get joins success {"Decoder1":"null"}<cr>

get joins error [incomplete]<cr>
get joins error [invalid subscription '<subscription>']<cr>
get joins error [device '<device_name>' not found]<cr>
get joins error [device '<device_name>' disconnected]<cr>
```

ToC

Command 1 to 6 set scalerget frame_converter

9.1 to 6.1 Command usage

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

9.1 to 6.2 Description

The command **get frame_converter** is used to retrieve the Encoder video stream frame rate.

9.1 to 6 S

encoder_device_name	Device name of a Encoder

9.1 to 6 7

- Use the command **set frame_converter** to change this setting.
- Returned values between 0 and 59.

9.1 to 6leave

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

9.1 to 6 to 6 Command Example2

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

9.1 to 6 to Return 5 Example2

```
get frame_converter success 30<cr>

get frame_converter error [incomplete]<cr>
get frame_converter error [encoder '<encoder_device_name>' not found]<cr>
get frame_converter error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

9.17 set scalerget preferred

9.11 Command usage

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

9.12 Description

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

9.15

decoder_device_name	Device name of Decoder
option	width | height | fps

9.17

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

9.1leave

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

9.16 Command Example

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

9.15 Example

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

get preferred error [incomplete]<cr>
get preferred error [invalid option '<option>']<cr>
get preferred error [no EDID available]<cr>
get preferred error [decoder '<decoder_device_name>' not found]<cr>
get preferred error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

Command part 1 Introduction set scaler get rotation

9.1 Part 10 1 Command usage

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

9.1 Part 10 2 Description

The command **get rotation** is used to retrieve the video output rotation status of the specified Decoder.

9.1 Part 10 S

decoder_device_name	Device name of the Decoder

9.1 Part 10 7

- Return value will be between 0 and 7.
- Use the command **set rotation** to change this setting.

9.1 Part 10 Leave

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

9.1 Part 10 6 Command Example

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

9.1 Part 10 Return 5 Example2

```
get rotation success 0<cr>
get rotation success 7<cr>

get rotation error [incomplete]<cr>
get rotation error [decoder '<decoder_device_name>' not found]<cr>
get rotation error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

9 set scaler get scaler

9.1 9.1 Command usage

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

9.1 9.2 Description

The command **get scaler** is used to retrieve the scaled video resolution of the Decoder's HDMI video.

9.1 9 S

decoder_device_name	Device name of the Decoder
option	all | width | height | fps

9.1 9 7

- If the Decoder has been set to 'original' then the command return will be 'original' for any option specified.

9.1 9 leave

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

9.1 9 6 Command Example2

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

9.1 9 Return 5 Example2

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

get scaler error [incomplete]<cr>
get scaler error [invalid option '<option>']<cr>
get scaler error [decoder '<decoder_device_name>' not found]<cr>
get scaler error [device '<decoder_device_name>' disconnected]<cr>
```

ToC

9.10 Command get status

9.10.1 Command usage

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

9.10.2 Example

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

9.10.3 Arguments

device_name	Name of the Encoder or Decoder
stream	video | audio | usb | serial | ir

9.10.4

- 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.

9.10.5 leave

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

9.10.6 Command Example

```
get status Encoder1<cr>
get status Encoder1 audio<cr>
get status Encoder1 video<cr>
get status Encoder1 usb<cr>
get status Encoder1 serial<cr>
get status key:abc123 Encoder1 ir<cr>
```

```
get status Decoder1<cr>
get status Decoder1 audio<cr>
get status Decoder1 video<cr>
get status Decoder1 usb<cr>
get status Decoder1 serial<cr>
get status Decoder1 ir<cr>
```

9.10.7 Return Example2

```
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 [invalid stream '<stream>']<cr>
get status error [device '<device_name>' not found]<cr>
```

ToC

Command2.3 set scalerget ver

9.1.66.1 Command usage

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

9.1.66Example

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

9.1.66.3 Arguments

device_name	Device name of either a Decoder or Encoder

9.1.66 7

9.1.66leave

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

9.1.66.6 Command Examples

```
get ver Encoder1<cr>
get ver key:abc123 Decoder1<cr>
```

9.1.66Return5 Example2

```
get ver success 1.1.2<cr>

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

ToC

Command get video

9.1.62.1 Command usage

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

9.1.62.2 Description

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

9.1.62.3 Arguments

encoder_device_name	Device name of the Encoder
option	all | width | height | fps | sm

9.1.62.4 Notes

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

9.1.62.5 Return Value

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

9.1.62.6 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 sm<cr>
get video key:abc123 Encoder1 all<cr>
```

9.1.62.7 Return Example2

```
get video success 1920 1080 60 PROGRESSIVE<cr>
get video success 1920<cr>
get video success 1080<cr>
get video success 60<cr>
get video success PROGRESSIVE<cr>
get video success INTERLACED <cr>

get video error [incomplete]<cr>
get video error [invalid option '<option>']<cr>
get video error [encoder '<encoder_device_name>' not found]<cr>
get video error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

Command get video_mute

Command usage

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

Example

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

Arguments

decoder_device_name	Device name of the Decoder

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

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

Command Example

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

Example2

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

get video_mute error [incomplete]<cr>
get video_mute error [decoder '<decoder_device_name>' not found]<cr>
get video_mute error [device '<decoder_device_name>' disconnected]<cr>
```

Command get video_quality

ToC

64.1 Command usage

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

64 Example

The command **get video_quality** is used to retrieve an Encoder's video stream quality.

64.3 Arguments

encoder_device_name	Device name of the Encoder

64 7

- Returned value in the range of -1 to 5**.**
- Use the command **set video_quality** to change this setting.

64 leave

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

64 6 Command Example

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

64 Return 5 Example2

```
get video_status success -1<cr>
get video_status success 5<cr>

get video_quality error [incomplete]<cr>
get video_quality error [encoder '<encoder_device_name>' not found]<cr>
get video_quality error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

Command get video_status

65.1 Command usage

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

Example

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

65.3 Arguments

encoder_device_name	Device name of the Encoder

65 7

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

leave

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

6 Command Example

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

Example2

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

get video_status error [incomplete]<cr>
get video_status error [encoder '<encoder_device_name>' not found]<cr>
get video_status error [device '<encoder_device_name>' disconnected]<cr>
```

ToC

66 Command get volume

66.1 Command usage

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

66 Example

The command **get volume** is used to retrieve the current analog audio volume level.

66.3 Arguments

device_name	Device name of either a Decoder or Encoder

66 7

- Use the command **set volume** to change this setting.

66 leave

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

66 6 Command Examples

```
get volume Encoder1<cr>
get volume key:abc123 Decoder1<cr>
```

66 Return 5 Example2

```
get volume success 0<cr>
get volume success 100<cr>

get volume error [incomplete]<cr>
get volume error [device '<device_name>' not found]<cr>
get volume error [device '<device_name>' disconnected]<cr>
```

ToC

9.2 Command get for system

This sections contains get commands for the system.

9.2.1 Command get events

9.2.1.1 Command usage

get events **[**key:<*security_key*>**]** <*event_name*> <*function*><cr>

9.2.1.2 Description

The command **get events** is used to retrieve the state of events created on the UI's Scheduler or Events tabs.

9.2.1.3

event_name	Name of the event
function	Current only "**state**" supported

9.2.1.4

- Event must be created on UI's Scheduler or Events tab before using command.

9.2.1.5

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

9.2.1.6 Command Example

```
get events state MyEvent<cr>
```

9.2.1.7 Example

```
get events success enabled<cr>
get events success disabled<cr>

get events error [incomplete]<cr>
get events error [event '<event_name>' not found]<cr>
get events error [invalid parameter]<cr>
```

ToC

2.2 set scalerget matrix

2.2.1 Command usage

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

2.2.2 Description

The command **get matrix** is used to retrieve connected Encoders and Decoders by stream type. The result is a json string of all device connections.

2.2.3

stream	audio | video | serial | ir | usb | usb_ext

2.2.4

- null will be the result when device is not joined with specified stream type.
- Success will return a json formatted string.

2.2.5leave

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

2.2.6 Command Example

```
get matrix audio<cr>
get matrix video<cr>
get matrix serial<cr>
get matrix ir<cr>
get matrix usb<cr>
get matrix usb_ext<cr>
```

2.2.7Return Return .5 Return 2

```
get matrix success {"<decoder_device_name>":"<encoder_device_name>","<decoder_device_name>":"<encoder_device_name>",…}<cr>
get matrix success {"<decoder_device_name>":"null","<decoder_device_name>":"<encoder_device_name>"}<cr>
get matrix success {"<rex_device_name>":"null","<rex_device_name>":"<lex_device_name>"}<cr>


get matrix error [incomplete]<cr>
get matrix error [invalid stream]<cr>
```

ToC

9.2.3 set scalerget var

9.3.3.1 Command usage

get var **[**key:<*security_key*>**]** <*var_name*><cr>

9.2.3.2 Description

The command **get var** is used to retrieve the value of the specified user defined variable.

9.2.3

var_name	Name of the variable

9.2.3 7

9.2.3leave

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

9.2.3 6 Command Example

```
get var MyVar<cr>
```

9.2.3 Return .5 Return 2

```
get var success <value><cr>

get var error [incomplete]<cr>
get var error [var '<var_name>' not found]<cr>
```

ToC

9.3 Command get for User Interfaces

This sections contains get commands for interaction with User Interfaces.

After controller start-up, User Interface services are not enabled until devices have been discovered. This ensures devices are available for initial preset and UI use. To force the UI service to run without devices, the UI service firstly needs to be disabled then enabled again.

9.3.4 get ui

9.3.4.1 Command usage

get ui **[**key:<*security_key*>**]** <*ui_name*><cr>

9.3.4.2 Description

The command **get ui** is used to retrieve the User Interface status.

9.3.4.3 Arguments

ui_name	Name of the User Interface

9.3.4.4 Arguments

- User Interface must be enabled as a feature for command to be used.
- Use the command **set ui_button** to change this setting.

9.3.4.5 Return

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

9.3.4.6 Command Example

```
get ui MyUI<cr>
```

9.3.4.7 Return Example

```
get ui disabled<cr>
get ui enabled<cr>
get ui enabled timeout 240<cr>
get ui enabled clients 1<cr> *number of users 1 to 100
get ui enabled login 1234<cr> *0000 to 9999
get ui enabled timeout 240 clients 1 login 1234<cr>

get ui error [ui '<ui_name>' not found]<cr>
```

ToC

8.2 Command get ui_button

8.2.1 Command usage

get ui_button **[**key:<*security_key*>**]** <*ui_name*> <*button_name*> <*function*><cr>

8.2.2 Description

The command **get ui_button** is used to retrieve the current state of a User Interface button.

8.2

ui_name	Name of the User Interface
button_name	Name of the button in the User Interface or preset logic <<button_name>>
function	position | state

8.2

- Control UI must be enabled as a feature for command to be used.
- Use the command **set ui_button** to change this setting.
- **function** > **position** uses **up** |**down**
- **function** > **state** uses **enabled** | **disabled**

- **Momentary**
  - state
- **Toggle**
  - position
  - state
- **Radio Toggle**
  - position
  - state
- **Split**
  - position
  - state
- **Repeat**
  - state

8.2

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

8.2 Command Example2

```
get ui_button MyUI MyButton position<cr>
get ui_button MyUI <<button_name>> position<cr>
get ui_button key:abc123 MyUI MyButton state<cr>
```

8.2 Return .5 Return 2

```
get ui_button position success up<cr>
get ui_button position success down<cr>
get ui_button state success enabled<cr>
get ui_button state success disabled<cr>

get ui_button error [incomplete]<cr>
get ui_button error [ui '<ui_name>' not found]<cr>
get ui_button error [button '<button_name>' not found]<cr>
get ui_button error [invalid parameter]<cr>
get ui_button error [service disabled]<cr>
```

ToC

9.3.3 Command get ui_indicator

9.3.3.1 Command usage

get ui_indicator **[**key:<*security_key*>**]** <*ui_name*> <*indicator_name*> <*function*><cr>

9.3.3.2 Description

The command **get ui_indicator** is used to retrieve the current state of a User Interface level indicator.

9.3.3

ui_name	Name of the User Interface or preset logic <<ui_name>>
indicator_name	Name of the level indicator in the User Interface
function	value

9.3.3

- The UI service must be enabled.
- Use the command **set ui_indicator** to change this setting.
- Used within a preset, <<ui_name>> can be used in place of <ui_name>.

9.3.3leave

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

9.3.36 Command Example2

```
get ui_indicator myUI myIndicator value<cr>
get ui_indicator key:abc123 myUI myIndicator value<cr>
get ui_indicator <<ui_name>> myIndicator value
```

9.3.3ReturnReturn .5 Return 2

```
get ui_indicator value success 0<cr>
get ui_indicator value success 100<cr>

get ui_indicator error [incomplete]<cr>
get ui_indicator error [invalid parameter]<cr>
get ui_indicator error [service disabled]<cr>
get ui_indicator error [ui '<ui_name>' not found]<cr>
get ui_indicator error [indicator '<indicator_name>' not found]<cr>
```

ToC

9.3.4 Command get ui_slider

9.3.4.1 Command usage

get ui_slider **[**key:<*security_key*>**]** <*ui_name*> <*slider_name*> <*function*><cr>

9.3.4.2 Description

The command **get ui_slider** is used to retrieve the current value or state of a User Interface level slider.

9.3.4 S

ui_name	Name of the User Interface or preset logic <<ui_name>>
slider_name	Name of the level slider in the User Interface
function	value | state

9.3.4 7

- The UI service must be enabled.
- Use the command **set ui_slider** to change this setting.
- Used within a preset, <<ui_name>> can be used in place of <ui_name>.

9.3.4leave

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

9.3.46 Command Example2

```
get ui_slider myUI mySlider value<cr>
get ui_slider key:abc123 myUI mySlider state<cr>
get ui_slider <<ui_name>> mySlider value
```

9.3.4ReturnReturn .5 Return2

```
get ui_slider value success 0<cr>
get ui_slider value success 100<cr>
get ui_slider state success enabled<cr>

get ui_slider error [incomplete]<cr>
get ui_slider error [invalid parameter]<cr>
get ui_slider error [service disabled]<cr>
get ui_slider error [ui '<ui_name>' not found]<cr>
get ui_slider error [slider '<slider_name>' not found]<cr>
```

ToC

10 Command send

The **send** commands are used to send either infrared or serial RS-232 data to any or all Encoders or Decoders from a 3rd party control system. Any TCP controllable device or Global Caché device can be controlled.

Commands missing **mode** return:

```
send error [incomplete]<cr>
```

Commands with invalid **mode** return:

```
send error [invalid mode]<cr>
```

ToC

10.1 Command send cec

10.6.1 Command usage

send cec [key:<security_key>] <*device_name*> / <*group_name*> / <all> / <all_tx> / <all_rx> <*data_hex*><cr>

10.6Example

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

10.6

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

10.6

- The **data_hex** argument is a hexadecimal string which represent the cec code to be sent.
- The command will return with an error when using a single device if no source is connected on an Encoder or no display connected on a Decoder.
- ***group_name*** is used as a destination when all Encoders and Decoders in a group are required to send CEC.

10.6leave

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

10.6.6 Command Examples

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

10.6Example

```
send cec success<cr>

send cec error [incomplete]<cr>
send cec error [HDMI disconnected]<cr>
send cec error [invalid HEX data]<cr>
send cec error [invalid response]<cr>
send cec error [device '<device_name>' not found]<cr>
send cec error [device '<device_name>' disconnected]<cr>
send cec error [group devices not found]<cr>
```

ToC

10.2 Command send cec_off

10.2.1 Command usage

send cec_off [key:<security_key>] <*decoder_device_name*> / <*group_name*> / <all_rx><cr>

10.2Example

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

10.2

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

10.2

- ***group_name*** is used as a destination when all Decoders in a group are required to send CEC.

10.2leave

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

10.2.6 Command Examples

```
send cec_off Decoder1<cr>
send cec_off MyGroup<cr>
send cec_off key:abc123 all_rx<cr>
```

10.2Example

```
send cec_off success<cr>

send cec_off error [incomplete]<cr>
send cec_off error [invalid response]<cr>
send cec_off error [decoder '<device_name>' not found]<cr>
send cec_off error [device '<device_name>' disconnected]<cr>
send cec_off error [group devices not found]<cr>
```

ToC

10.3 Command send cec_on

10.all1 Command usage

send cec_on [key:<security_key>] <*decoder_device_name*> / <*group_name*> / <all_rx><cr>

10.alExample

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

10.all

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

10.all

- ***group_name*** is used as a destination when all Decoders in a group are required to send CEC.

10.alleave

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

10.all6 Command Examples

```
send cec_on Decoder1<cr>
send cec_on MyGroup<cr>
send cec_on key:abc123 all_rx<cr>
```

10.alExample

```
send cec_on success<cr>

send cec_on error [incomplete]<cr>
send cec_on error [invalid response]<cr>
send cec_on error [decoder '<device_name>' not found]<cr>
send cec_on error [device '<device_name>' disconnected]<cr>
send cec_on error [group devices not found]<cr>
```

ToC

10.4 Command send gc

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

10.4.1.6 Command usage

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

10.4 Example

The **send gc** command allows control of all Global Caché products via TCP.

10.4

address	Global Caché IP address
port	Global Caché TCP port. Usually 4998 and for serial com1: 4999 com2: 5000
gc_api	Global Caché API string
feedback	Keywords **reply**, **equals** or **contains** (optional)
feedback_string	String used with equals or contains to compare with the feedback string

10.4

- The **gc_api** string uses the same standard Global Caché control string format as found in the Global Caché API manuals downloaded from: www.globalcache.com/downloads/
- Use keyword "**[disconnect]**" in place of ***gc_api*** string to terminate the connection.
- Strings sent to port 4999 or 5000 must be wrapped in quotations "my string".
- The ***feedback*** option uses keywords **reply**, **equals** or **contains** to set the type of feedback.
  To receive a string only, use **reply**.
  For comparison with the specified ***feedback_string*** use **equals** for an exact match, or **contains** for a match within the string.

10.4.5 Return Value

Return values from port 4998 will be in the standard Global Caché 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.

Return values from either port 4999 or 5000 will be the return serial string unless a keyword is used like "reply", "contains" or "equals" in which case "reply" will return with the received serial string and "contains" and "equals" will return with a success or error as a result of comparing the serial return string.

```
send gc 172.30.1.111 4999 "a string to send"<cr> = send gc success<cr>
send gc 172.30.1.111 4999 "a string to send" reply<cr> = <feeback_string>
send gc 172.30.1.111 4999 "a string to send" contains "string"<cr> = send gc success<cr>
send gc 172.30.1.111 4999 "a string to send" contains "oops"<cr> = send gc error [<feeback_string>]
send gc 172.30.1.111 4999 "a string to send" equals "string"<cr> = send gc success<cr>
send gc 172.30.1.111 4999 "a string to send" equals "oops"<cr> = send gc error [<feeback_string>]
```

ToC

10.4.6 Example2

```
send gc 172.30.1.111 4999 "a string to send"<cr>
send gc 172.30.1.111 4999 "a string to send" reply<cr>
send gc 172.30.1.111 4999 "a string to send" contains "a string to find"<cr>
send gc 172.30.1.111 4999 "a string to send" equals "a string to find"<cr>
send gc 172.30.1.111 4998 setstate,1:1,1<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 172.30.1.111 4998 [disconnect]<cr>
send gc key:abc123 172.30.1.111 4999 "\x00\x01"<cr>
```

10.4.7 Return .5 Return 2

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

send gc error [invalid]<cr>
send gc error [invalid format]<cr>
send gc error [incomplete]<cr>
send gc error [device '<address>' not found]<cr>
send gc error [device '<address>' timeout]<cr>
send gc error [<string received>]<cr>
ERR_<XX><cr>
```

ToC

10.5 set scalersend ir

10.av1 Command usage

send ir [key:<security_key>] <*device_name*> / <*group_name*> / <all> / <all_tx> / <all_rx> <*data_hex*><cr>

10.av2 Description

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

10.avs

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

10.av7

- 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.
- ***group_name*** is used as a destination when all Encoders and Decoders in a group are required to send IR.

10.avleave

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

10.av6 Command Examples

```
send ir Decoder1 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015001500400015004000150040001500
150015001500150015001500150015001500150015004000150015001500150015004000150040001500150015001500150040001500150015004000150040001500150015
015001500400015004000150015001500150689<cr>

send ir MyGroup 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015001500400015004000150040001500
150015001500150015001500150015001500150015004000150040001500150015001500150040001500150015004000150040001500150015015001500400015004000150
015001500400015004000150015001500150689<cr>

send ir key:abc123 all_rx 000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015001500400015004000150040001500
400015001500150015001500150015001500150015001500400015001500150015001500400015004000150015001500150015001500400015001500150040001500400015015
015001500150015004000150040001500150015006 89<cr>
```

10.av7 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 [invalid HEX data]<cr>
send ir error [invalid response]<cr>
send ir error [device '<device_name>' not found]<cr>
send ir error [device '<device_name>' disconnected]<cr>
send ir error [group devices not found]<cr>
```

ToC

10.6 set scalersend serial

10.6.1 Command usage

send serial [key:<security_key>] <*device_name*> / <*group_name*> / <all> / <all_tx> / <all_rx> "<*data_string*>" [<*feedback*> ["<*feedback_string*>"]]<cr>

10.6 Example

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

10.6

device_name	Device name of Encoder, Decoder, Group or '**all**' | '**all_rx**' | '**all_tx**'
data_string	String of ASCII characters
feedback	Keywords **reply**, **equals** or **contains** (optional)
feedback_string	String used with **equals** or **contains** to compare with the feedback string

10.6

- 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 device must be set for **control** mode in the RS232 serial port settings. When unsolicited serial data is received the Controller will raise a 'notify serial' event.

- The **data_string** and **feedback_string** arguments are ASCII text strings when set to ASCII mode.
  When the device is in ASCII mode it will only be possible to send escaped \x0D carriage return and / or \x0A line feed.
  - abcdefghijklmnopqrstuvwxyz0123456789\x0D
  - \x0D\x0A

- The **data_string** and **feedback_string** arguments are ASCII text strings of bytes when set to HEX mode.
  - 00FF
  - \x00\xFF

- The ***feedback*** option uses keywords **reply**, **equals** or **contains** to set the type of feedback.
  To receive a string only, use **reply**.
  For comparison with the specified ***feedback_string*** use **equals** for an exact match, or **contains** for a match within the string. Can only be used when ***device_name*** is a single Encoder or Decoder.

- ***feedback_string*** contains the expected device's feedback string result.

- ***group_name*** is used as a destination when all Encoders and Decoders in a group are required to send.

ToC

10.6 leave

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

10.6.6 Command Example2

```
send serial Decoder1 "my data string\x0D"<cr>
send serial Encoder1 "\x00\x01\x02\x03\x04"<cr>
send serial Decoder1 "my data string" reply<cr>
send serial Decoder1 "my data string\x0D" contains "\x0D"<cr>
send serial Decoder1 "my data string\x0D\x0A" equals "OK"<cr>
send serial key:abc123 Decoder1 "000102FF"<cr>
send serial MyGroup "my data string\x0D"<cr>
```

10.6.7 Return .5 Return 2

```
send serial success [my return string]<cr>
send serial success [00FF0D]<cr> * Received HEX in HEX mode
send serial success [\x00\xFF\x0D]<cr> * Received HEX in ASCII mode
send serial success []<cr>
send serial success<cr>

send serial error [incomplete]<cr>
send serial error [invalid format]<cr>
send serial error [invalid HEX data]<cr>
send serial error [invalid HEX feedback]<cr>
send serial error [invalid parameter]<cr>
send serial error [invalid response]<cr>
send serial error [device '<device_name>' not found]<cr>
send serial error [device '<device_name>' disconnected]<cr>
send serial error [group devices not found]<cr>
```

ToC

10.7 Command send tcp

10.7.1 Command usage

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

10.7.2 Example

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

10.7.3 Syntax

address	TCP IP address
port	TCP port
command	Command string to send to TCP device
feedback	Keyword **reply**, **equals** or **contains** (optional)
feedback_string	Expected feedback string used with **equals** or **contains**

10.7.4 Remarks

- The TCP device must be in the same range as the Controller.
- To send HEX add \x before the HEX byte. \x0D for carriage return
- The feedback option uses keywords reply, equals or contains to set the type of feedback.
  To receive a string only, use reply.
  For comparison with the specified feedback_string use **equals** for an exact match, or **contains** for a match within the string. Can only be used when device_name is a single Encoder or Decoder.
- ***feedback_string*** contains the expected device's feedback string result.
- Use keyword "**[disconnect]**" in place of ***command*** string to terminate the connection.

10.7.5 Response

A success return value will contain what is returned from the TCP device.

10.7.6 Example2

```
send tcp 172.30.1.111 1000 "ascii string"<cr>
send tcp 172.30.1.111 1000 "an mixed string\x0D"<cr>
send tcp 172.30.1.111 1000 "\x00\x01\x02\x03"<cr>
send tcp 172.30.1.111 1000 "ascii string" contains "feedback string"<cr>
send tcp 172.30.1.111 1000 "ascii string" equals "feedback string"<cr>
send tcp 172.30.1.111 1000 "ascii string" reply<cr>
send tcp 172.30.1.111 1000 [disconnect]<cr>
send tcp key:abc123 172.30.1.111 1000 "ascii string"<cr>
```

ToC

10 Return Return Return .5 Return 2

```
send tcp succuss<cr>
send tcp succuss [<feedback>]<cr>
send tcp success [disconnected]<cr>

send tcp error [incomplete]<cr>
send tcp error [invalid format]<cr>
send tcp error [invalid parameter]<cr>
send tcp error [invalid address]<cr>
send tcp error [invalid port]<cr>
send tcp error [device '<address>' not found]<cr>
send tcp error [<feedback>]<cr>
```

ToC

11 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 or video wall.
Refer Appendix B - Preset logic for logic that can be applied within a preset.

Commands missing **mode** return:

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

Commands with invalid **mode** return:

```
preset error [invalid mode]<cr>
```

11.1 Command preset add

66.1.1 Command usage

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

66.1.2 Description

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

66.1.3 Arguments

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

66.1.4 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.

66.1.5 Return Value

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

66.1.6 Command Examples

```
preset add preset1 join all Encoder1 Decoder1<cr>
preset add key:abc123 preset1 join all Encoder1 Decoder1<cr>
```

66.1.7 Return Examples

```
preset add preset1 success<cr>

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

ToC

11.2 Command preset delay

66.2.1 Command usage

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

66.2 Example

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

66.2 S

milliseconds	Delay time in milliseconds up to 9999

66.2 7

- This command can only be used within a preset.

66.2 leave

None

66.2.1.2 Description

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

66.2.7 Return Example

```
No return is given for a valid command

preset delay error [incomplete]<cr>
preset delay error [invalid milliseconds]<cr>
```

ToC

11 8 Command preset delete

8.3 al Command

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

8.3 al Example

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

8.3 all

preset_name	The name defined as the preset

8.3 all 7

8.3 all eave

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

8.3 all 6 Command Example2

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

8.3 all Return .5 Return 2

```
preset delete preset1 success<cr>

preset delete error [incomplete]<cr>
preset delete error [preset '<preset_name>' not found]<cr>
```

ToC

11.4 Command preset load

66.4.1 Command

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

66.4.1 Example

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

66.4.1 Syntax

preset_name	The name defined as the preset
delay	Delay in minutes before preset is applied (optional)

66.4.1 Notes

- The preset is created with the **preset add** command or directly via the UI.
- Optional "***delay***" is used to delay a preset for the specified amount of time in minutes. The delay is reset each time the command is used and -1 will terminate the command.

66.4.1 Response

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

66.4.1.2 Description

```
preset load preset1<cr>
preset load preset1 30<cr>
preset load preset1 -1<cr>
preset load key:abc123 preset1<cr>
```

66.4.1 Example

```
preset load preset1 success<cr>
preset load preset1 delayed<cr>

preset load error [incomplete]<cr>
preset load error [invalid delay]<cr>
preset load error [preset '<preset_name>' not found]<cr>
preset load <preset_name> error […]<cr>
```

ToC

1 Message notify

The **notify** messages are sent from the Controller to a third party control system connected on Telnet 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
- Encoder or Decoder network connectivity
- Decoder display connectivity
- Encoder source connectivity

12.1 Message notify serial

12.1.1 Message received

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

12.1.2 Example

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

12.1.3 Syntax

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 device must be set to **control** mode.

12.1.5 Example

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

12.1.6 Example2

```
notify serial '<device_name>' my data string\x0D<cr>
notify serial '<device_name>' \x00\x01\x02\x03\xFF<cr>
notify serial '<device_name>' 0001020304<cr>
```

ToC

12.2 Message notify network

62.21.6Message received

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

62.2Example

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

62.2

device_name	Device name of Encoder or Decoder
state	true | false

62.2 7

- A **notify network** message will be sent when the 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.

62.2leave

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

62.2 Example2

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

ToC

12.3 Message notify display

62.1.6 Message received

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

62.1 Example

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

62.1 Syntax

decoder_device_name	Device name of Decoder
state	true | false

62.1 Notes

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

62.1 Response

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

62.1 Example2

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

ToC

12.4 Message notify source

62.4.1.6 Message received

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

62.4 Example

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

62.4 S

encoder_device_name	Device name of Encoder
state	true | false

62.4 7

- The Controller is looking for a stable valid video signal.

62.4 leave

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

62.4 Example2

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

ToC

Appendix A - How to HTTP request

GET = http://<*controllerURL*>/api/command/< *API_COMMAND*>/<*KEY*>

POST = http://<*controllerURL*>/api/command{"cmd": "< *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 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 Director 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 controller
	var BaseURL = 'http:// ' + controllerIP + '/api/command';
	var key = '123xyz'; *replace this with the generated security key
	var command = '123xyz'; *replace with any DIRECTOR 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 DIRECTOR 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":"<DIRECTOR_API_COMMAND>","key":"<KEY>"}

Example 5: IFTTT – Webhooks – GET

URL
http://<controllerIP>/api/command/<DIRECTOR_API_COMMAND>/<KEY>

Method
GET

Content Type
text/plain

ToC

Example 6: bt.tn – Auxiliary HTTP request GET

URL=<controllerIP>/api/command/<DIRECTOR_API_COMMAND>/<KEY> port=80
**Note: All spaces in the <DIRECTOR_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":"< DIRECTOR_API_COMMAND>","key":"<KEY>"}

ToC

bt.tn – B – Preset Logic

Basic if else logic can be applied within a preset to allow you to build some *smarts* into your system. All commands can be used as an expression.

The following syntax applies:

```
if (something) {
    <do_this>
    …
} elseif (something_else) {
    <do_this>
    …
} else {
    <do_this_instead>
    …
}
```

Within presets only, the following commands can be used to manipulate strings:

- (substr(<string>,<startIndex>,<optional_numOfCharacters>))
- (substring(<string>, <startIndex>, <optional_endIndex>))
- (instr(<string>,<searchString>,<offset>))
- (trim (<string>))
- inc(<variable>,<optional_amount>)
- dec(<variable>,<optional_amount>)
- &

substr() will extract a string from a string using a starting index and length.

substring() will extract a string from a string using a starting and ending index.

instr() Boolean use returns as false when <string> does not contain <searchString> and returns true when <string> does contain <searchString>.
Integer use returns as 0 when <string> does not contain <searchString> and returns the start position of <searchString> within <string> from 1.
Optional use of <offset> can be used to add or subtracted from the resulting index.

trim() will remove any HEX after the last ASCII character, like terminators <cr> and <lf> from the end of the string.

inc() is used to increment the integer value of <variable> by <optional_amount>, otherwise 1 used as default.

dec() is used to decrement the integer value of <variable> by <optional_amount>, otherwise 1 used as default.

& is used to append strings together.

Anything that needs to be resolved in a command string must be wrapped in brackets (). Brackets are resolved in a command string from the middle out.
So in the following example, (get var myVar) is firstly replaced with the variable value. Next the variable value will be split with substr().

```
set ui_label myUI lbl01 text (substr((get var myVar),1,4))
```

1st resolved set of brackets for get var = set ui_label myUI lbl01 text (substr("testing123",1,4))
2nd resolved set of brackets for substr = set ui_label myUI lbl01 text "test"

Appendix E – Preset Logic continued…

Variables used within preset are rather unique in that they are global and can be created, set or deleted at any time with API commands 'set var' and 'get var'. The variable is global so it can be accessed from any other preset. It will remain on the system until deleted or power is removed from the controller or rebooted.

A variable can be treated as either **Boolean** for true / false, **String** for text or **Integer** for numbers.

Boolean example:

```
set var myVar false
set var myVar true

if (get var myVar) {
   send tcp 172.30.10.105 1234 "true"
} else {
   send tcp 172.30.10.105 1234 "false"
}

if not (get var myVar) {
   send tcp 172.30.10.105 1234 "false"
}

set var myVar [delete]
```

String example:

```
set var myVar "abcdef"

if (get var myVar) {
   send tcp 172.30.10.105 1234 "true"
}

if (get var myVar == "abcdef") {
   send tcp 172.30.10.105 1234 "true"
} else {
   send tcp 172.30.10.105 1234 "false"
}

set var myVar [delete]
```

Integer example:

```
set var myVar 10

if (get var myVar) {
   send tcp 172.30.10.105 1234 "true"
}

if (get var myVar > 9) {
   send tcp 172.30.10.105 1234 "true"
} else {
   send tcp 172.30.10.105 1234 "false"
}

set var myVar [delete]
```

Appendix E – Preset Logic continued…

Strings can be extracted with command **substr()** as follows:

```
substr(<string>,<startIndex>,<optional_numOfCharacters>)

set var myVar "LED_LIGHTING,1:3,25,50"

set ui_label myUI lbl01 text (substr((get var myVar),1))
    lbl01 text = "LED_LIGHTING,1:3,25,50"

set ui_label myUI lbl01 text (substr((get var myVar),18))
    lbl01 text = " 25,50"

set ui_label myUI lbl01 text (substr((get var myVar),18,2))
    lbl01 text = "25"
```

Strings can be extracted with command **substring()** as follows:

```
substring(<string>,<startIndex>,<optional_ endIndex>)

set var myVar "LED_LIGHTING,1:3,25,50"

set ui_label myUI lbl01 text (substring((get var myVar),1))
    lbl01 text = "  LED_LIGHTING,1:3,25,50"

set ui_label myUI lbl01 text (substring((get var myVar),18,20))
    lbl01 text = "25"
```

Instance of a string or string starting index can be found with command **instr()** as follows:

```
instr(<string>,<searchString>,<offset>)

set var myVar "LED_LIGHTING,1:3,25,50"

if (instr((get var myVar),"LIGHTING")) {
    // myVar contains "LIGHTING"
} else {
    // myVar does not contain "LIGHTING"
}

set var myVarIndex (instr((get var myVar),"LIGHTING"))
    myVarIndex = 5

set var myVarIndex (instr((get var myVar),"ERROR"))
    myVarIndex = 0

set var myVarIndex (instr((get var myVar),"1:3",4))
    myVarIndex = 18
```

Combining **str()** and **instr()** to extract a value from a return string as follows:

```
substr(<string>,<startIndex>,<optional_numOfCharacters>)
instr(<string>,<searchString>,<offset>)

set var LevelTmp (trim(send gc 10.1.1.208 4998 "get_LED_LIGHTING,1:3\x0D" reply))
    LevelTmp = LED_LIGHTING,1:3,25,50

set var LevelTmp (substr((get var LevelTmp),18))
    LevelTmp = 25,50

set var LevelTmp (substr((get var LevelTmp),0,(instr((get var LevelTmp),",",-1))))
    LevelTmp = 25

set ui_slider LightsUI sld1 value (get var PixieLevel)

if ((get var LevelTmp) > 0) {
    set ui_slider LightsUI sld1 state enabled
    set var LightLevel (get var LevelTmp)
}

set var LevelTmp [delete]
```

bt.tn — E – Preset logic continued…

Strings can be trimmed of trailing hexadecimal with a **trim()** command as follows:

```
if (trim(send tcp 172.30.10.141 4998 "setstate,1:1,1\x0d" reply) == "state,1:1,1") {
    set ui_label AT01 lbl01 text "good"
} else {
    set ui_label AT01 lbl01 text "bad"
}

set var myVar (send tcp 172.30.10.141 4998 "setstate,1:1,1\x0d" reply)
if (trim(get var myVar) == "state,1:1,1") {
    set ui_label AT01 lbl01 text "good"
} elseif (trim(get var myVar) == "state,1:1,0") {
    set ui_label AT01 lbl01 text "bad"
} else {
    set ui_label AT01 lbl01 text "error"
}

set var myVar (trim(send tcp 172.30.10.141 4998 "setstate,1:1,1\x0d" reply))
if (get var myVar == "state,1:1,1") {
    set ui_label AT01 lbl01 text "good"
} elseif (get var myVar == "state,1:1,0") {
    set ui_label AT01 lbl01 text "bad"
} else {
    set ui_label AT01 lbl01 text "error"
}
```

Variable integer values can be incremented with **inc()** command as follows:

```
inc(<variable>,<optional_amount>)

set var myVar 10
inc(myVar)
    myVar = 11
inc(myVar,2)
    myVar = 12
```

Variable integer values can be decremented with **dec()** command as follows:

```
dec(<variable>,<optional_amount>)

set var myVar 10
dec(myVar)
    myVar = 9
dec(myVar,2)
    myVar = 8
```

Using **&** to append strings together as follows:

```
set var myvar (send tcp 172.16.10.91 1234 "\x00\x01" reply)
    myVar = "OK"
send tcp 172.16.10.91 1234 ("RX: " & (get var myvar) & "\x0D")
    string sent = "RX: OK\x0D"
```

ToC

bt.tn – E – Preset logic continued…

Within button presets only, the following variable can be used to identify the pressed button:

- **<<button_name>>** which identifies the button pressed by name.
  Note: If the button does not have a name then "<<button_name>>" will be the result.

Within slider presets only, the following variable can be used to obtain the sliders current value:

- **<<slider_value>>** which returns the active slider value.

Within any UI preset, the following variable can be used to obtain the UI name:

- **<<ui_name>>** which identifies the UI by name.

Here are some examples of use from a sliders preset:

Example 1: Set another slider (slider2) in same UI with same value:

```
set ui_slider <<ui_name>> slider2 value <<slider_value>>
```

Example 2: Set a variable with the slider value:

```
set var sliderValue <<slider_value>>
```

Example 3: Send a command string containing slider value:

```
send tcp 172.30.10.105 1234 ("SETL 1 FDRLVL TCTX 1," & <<slider_value>> & "\x0D")
```

ToC

Appendix E – Preset logic continued…

The following **get** commands will return a **string** value that can be used with:

- **==** (equal to)
- **!=** (not equal to)

  - get var
  - get ui_button

Example 1:

```
if ((get var myVar) == "<string>") {
    <do_this>
} else {
    <do_this_instead>
}
```

Example 2:

```
if ((get var myVar) != "<string>") {
    <do_this>
} else {
    <do_this_instead>
}
```

Example 3:

```
if (get ui_button UIexample btn01 position == "down") {
    send gc 172.30.20.129 4998 setstate,1:1,1
}
```

The following **send** commands will return a **string** value that can be used with:

- **==** (equal to)
- **!=** (not equal to)

  - send tcp > feedback = reply
  - send gc > (port 4999 / 5000) feedback = reply
  - send gc > (port 4998) command get

Example 1:

```
if (send tcp 169.254.1.70 4998 "setstate,1:1,1\x0D" reply == "state,1:1,1\x0D") {
    <do_this>
} else {
    <do_this_instead>
}

if ((trim(send tcp 169.254.1.70 4998 "setstate,1:1,1\x0D" reply) == "state,1:1,1") {
    <do_this>
} else {
    <do_this_instead>
}
```

Example 2:

```
if (send gc 169.254.1.70 4999 "My String" reply == "This String") {
    set ui_button UIexample btn01 position down
}
```

Example 3:

```
if (send gc 169.254.1.70 4998 getstate,1:1 == "state,1:1,1\x0D") {
    set ui_button UIexample btn01 position down
}

if (trim(send gc 169.254.1.70 4998 getstate,1:1) == "state,1:1,1") {
    set ui_button UIexample btn01 position down
}
```

ToC

Appendix E – Preset logic continued…

The following **get** command will return a **Boolean** value that can be used with:

- not
  - get var

Example 1:

```
if (get var myVar) {
    <do something>
} elseif (get var myVar2) {
    <do something else>
}
```

Example 2:

```
if not (get var myVar) {
    <do something>
}
```

The following **send** commands will return a **Boolean** value that can be used with:

- not
  - send tcp > feedback = none, equals or contains
  - send gc > feedback = none, equals or contains

Example 1:

```
if (send tcp 172.30.20.129 4998 "setstate,1:1,1\x0D" contains "setstate,1:1,1") {
    set ui_button UIexample btn01 position down
} else {
    set ui_button UIexample btn01 position up
}
```

Example 2:

```
if not (send gc 172.30.20.129 4998 setstate,1:1,1) {
    <do something>
} else {
    <do something else>
}
```

ToC

Appendix E – Preset logic continued…

Multiple conditional statements can be included using "&&" (and) as well as "||" (or).

Example 1:

```
if (get ui_button UIexample btn01 position == down && get ui_button UIexample btn01 position == down) {
    send gc 172.30.20.129 4998 setstate,1:1,1
}
```

Example 2:

```
if (get ui_button UIexample btn01 position == down || get ui_button UIexample btn01 position == down) {
    send gc 172.30.20.129 4998 setstate,1:1,1
}
```

Using a variable as string to store a send tcp/serial command result string:

```
set var myVar send tcp 10.1.1.10 6970 "{status}\x0D" reply

if ((get var myVar) == "option1") {
    <do something>
} elseif ((get var myVar) == "option2") {
    <do something>
} elseif ((get var myVar) == "option3") {
    <do something>
} else {
    <do something>
}

set ui_label myControl lbl01 text (get var myVar)
```

Using a variable as Boolean to set/store a button state:

```
set var myVar false

if (get var myVar) {
    set ui_button myControl btn01 position up
} else {
    set ui_button myControl btn01 position down
}
```

ToC

bt.tn — E – Preset logic continued…

Single preset UI example which looks for the <<button_name>> pressed

```
set var myIP4998 "xxx.xxx.xxx.xxx"
set var myIP4999 "xxx.xxx.xxx.xxx"
set var myIP6980 "xxx.xxx.xxx.xxx"

if (<<button_name>> == "Btn01") {
    send tcp (get var myIP4998) 4998 "setstate,1:1,0\x0D" contains "state,1:1,0"

} elseif (<<button_name>> == "Btn02") {
    send tcp (get var myIP4998) 4998 "setstate,1:1,1\x0D" contains "state,1:1,1"


} elseif (<<button_name>> == "Btn03") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
       send tcp (get var myIP4998) 4998 "set_LED_LIGHTING,1:1,100,0\x0D" contains "LED_LIGHTING,1:1,100,100"
       set ui_slider <<ui_name>> LightsLevel value 100
    }else{
        send tcp (get var myIP4998) 4998 "set_LED_LIGHTING,1:1,10,0\x0D" contains "LED_LIGHTING,1:1,10,10"
        set ui_slider <<ui_name>> LightsLevel value 10
    }


} elseif (<<button_name>> == "Btn04") {
    send tcp (get var myIP4998) 4998 "set_LED_LIGHTING,1:1,100,10\x0D" contains "LED_LIGHTING,1:1"
    set ui_slider <<ui_name>> LightsLevel value 100

} elseif (<<button_name>> == "Btn05") {
    send tcp (get var myIP4998) 4998 "set_LED_LIGHTING,1:1,0,10\x0D" contains "LED_LIGHTING,1:1"
    set ui_slider <<ui_name>> LightsLevel value 10

} elseif (<<button_name>> == "Btn06") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
       send tcp (get var myIP6980) 6980 "send serial decoder2 ""PON""\x0D" contains "send serial success"
    }else{
        send tcp (get var myIP6980) 6980 "send serial decoder2 ""POFF""\x0D" contains "send serial success"
    }

} elseif (<<button_name>> == "Btn07") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
       send tcp (get var myIP4999) 4999 "PON\x0D" contains "PON OK"
    }else{
        send tcp (get var myIP4999) 4999 "POFF\x0D" contains "POFF OK"
    }

} elseif (<<button_name>> == "Btn08") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
       send tcp (get var myIP6980) 6980 "send serial decoder3 ""PON""\x0D" contains "send serial success"
    }else{
        send tcp (get var my myIP6980) 6980 "send serial decoder3 ""POFF""\x0D" contains "send serial success"
    }

} elseif (<<button_name>> == "Btn09") {
    set ui_button <<ui_name>> Btn02 press
    set ui_button <<ui_name>> Btn05 press
    preset delay 2000
    set ui_button <<ui_name>> BtnS03 press
    set ui_button <<ui_name>> Btn11 press
    preset delay 500
    if (get ui_button <<ui_name>> BtnS04 position == "up") {
        set ui_button <<ui_name>> BtnS04 press
    }
    if (get ui_button <<ui_name>> Btn06 position == "up") {
        set ui_button <<ui_name>> Btn06 press
    }
    if (get ui_button <<ui_name>> Btn08 position == "up") {
        set ui_button <<ui_name>> Btn08 press
    }
    if (get ui_button <<ui_name>> Btn07 position == "up") {
        set ui_button <<ui_name>> Btn07 press
    }

} elseif (<<button_name>> == "Btn10") {
    if (get ui_button <<ui_name>> Btn06 position == "down") {
        set ui_button <<ui_name>> Btn06 press
    }
    if (get ui_button <<ui_name>> Btn07 position == "down") {
        set ui_button <<ui_name>> Btn07 press
    }
    if (get ui_button <<ui_name>> Btn08 position == "down") {
        set ui_button <<ui_name>> Btn08 press
    }
    set ui_button <<ui_name>> Btn04 press
    set ui_button <<ui_name>> Btn13 press
    preset delay 1000
    set ui_button <<ui_name>> Btn01 press

} elseif (<<button_name>> == "Btn11") {
    send tcp (get var myIP6980) 6980 "send ir encoder3
0000006700000015006000170017001700170017002F0017001700170017002F0017002F0017001700170017002F0017001700170017002F0017002F0017001700170017002F0017002F0017002F0204\x0D" contains
"send ir success"

} elseif (<<button_name>> == "Btn12") {
    send tcp (get var myIP6980) 6980 "send ir encoder3
0000006700000015006000170017001700170017002F0017002F0017001700170017001700170017002F0017001700170017002F0017002F0017001700170017001700170017002F0017002F0017002F0204\x0D" contains
"send ir success"

} elseif (<<button_name>> == "Btn13") {
    send tcp (get var myIP6980) 6980 "send ir encoder3
0000006700000015006000170017001700170017001700170017002F0017002F0017001700170017001700170017002F0017001700170017001700170017002F0017002F0017002F021A\x0D" contains
"send ir success"

} elseif (<<button_name>> == "Btn14") {
    send tcp (get var myIP6980) 6980 "reset\x0D" contains "reset ok"

} elseif (<<button_name>> == "BtnS04") {
    if (get ui_button <<ui_name>> BtnS04 position == "down") {
        if (get ui_button <<ui_name>> BtnS01 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder1 all_rx\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS02 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder2 all_rx\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS03 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder3 all_rx\x0D" contains "join fast success"
        }
    }else{
       send tcp (get var myIP6980) 6980 "leave all all\x0D" contains "leave all success"
    }
...
```

ToC

bt.tn — E – Preset logic continued…

```
} elseif (<<button_name>> == "BtnS05") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
        if (get ui_button <<ui_name>> BtnS01 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder1 decoder2\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS02 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder2 decoder2\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS03 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder3 decoder2\x0D" contains "join fast success"
         }
    }else{
        send tcp (get var myIP6980) 6980 "leave all decoder2\x0D" contains "leave all success"
    }

} elseif (<<button_name>> == "BtnS06") {
    if (get ui_button <<ui_name>> <<button_name>> position == "down") {
        if (get ui_button <<ui_name>> BtnS01 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder1 decoder1\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS02 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder2 decoder1\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS03 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder3 decoder1\x0D" contains "join fast success"
         }
    }else{
        send tcp (get var myIP6980) 6980 "leave all decoder1\x0D" contains "leave all success"
    }

} elseif (<<button_name>> == "BtnS07") {
    if (get ui_button <<ui_name>> BtnS07 position == "down") {
        if (get ui_button <<ui_name>> BtnS01 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder1 decoder3\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS02 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder2 decoder3\x0D" contains "join fast success"
        }elseif (get ui_button <<ui_name>> BtnS03 position == "down") {
            send tcp (get var myIP6980) 6980 "join fast encoder3 decoder3\x0D" contains "join fast success"
         }
    }else{
        send tcp (get var myIP6980) 6980 "leave all decoder3\x0D" contains "leave all success"
    }
```

ToC