

# ***DIRECTOR***

## **COMMAND GUIDE**

# **CONTROL U**





# DIRECTOR CONTROL UI COMMAND GUIDE V1.3.6

## DIRECTOR





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Table of Contents

|   |    |
|---|----|
| <b>1 Introduction</b>   | 7  |
| 1.1 License Requirements  | 7  |
| 1.2 Telnet Connection and API Commands                          | 7  |
| 1.3 HTTP Requests   | 7  |
| <b>2 Command Overview</b>                                       | 8  |
| <b>3 Command set</b>  | 9  |
| 3.1 Command set listener  | 9  |
| 3.1.1 Command usage   | 9  |
| 3.1.2 Description   | 9  |
| 3.1.3 Arguments   | 9  |
| 3.1.4 Notes   | 9  |
| 3.1.5 Return Value  | 9  |
| 3.1.6 Command Examples  | 9  |
| 3.1.7 Return Examples   | 10 |
| 3.1.8 GC-100 Series IR/Sensor Connector Wiring                  | 10 |
| 3.1.9 iTach and GlobalConnect Series IR/Sensor Connector Wiring | 10 |
| 3.1.10 Flex Link Relay & Sensor Cable Wiring                    | 10 |
| 3.1.11 Technical Notes  | 11 |
| 3.2 Command set events  | 12 |
| 3.2.1 Command usage   | 12 |
| 3.2.2 Description   | 12 |
| 3.2.3 Arguments   | 12 |
| 3.2.4 Notes   | 12 |
| 3.2.5 Return Value  | 12 |
| 3.2.6 Command Example   | 12 |
| 3.2.7 Return Example  | 12 |
| 3.3 Command set var   | 12 |
| 3.3.1 Command usage   | 12 |
| 3.3.2 Description   | 12 |
| 3.3.3 Arguments   | 12 |
| 3.3.4 Notes   | 12 |
| 3.3.5 Return Value  | 12 |
| 3.3.6 Command Examples  | 12 |
| 3.3.7 Return Examples   | 12 |
| 3.4 Command set ui  | 14 |
| 3.4.1 Command usage   | 14 |
| 3.4.2 Description   | 14 |
| 3.4.3 Arguments   | 14 |
| 3.4.4 Notes   | 14 |
| 3.4.5 Return Value  | 14 |
| 3.4.6 Command Example   | 14 |
| 3.4.7 Return Example  | 14 |
| 3.5 Command set ui_button                                       | 15 |
| 3.5.1 Command usage   | 15 |
| 3.5.2 Description   | 15 |
| 3.5.3 Arguments   | 15 |
| 3.5.4 Notes   | 15 |
| 3.5.5 Return Value  | 15 |
| 3.5.6 Command Examples  | 16 |
| 3.5.7 Return Examples   | 16 |



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Table of Contents continued...

|                              |    |
|------------------------------|----|
| 3.6 Command set ui_label     | 17 |
| 3.6.1 Command usage          | 17 |
| 3.6.2 Description            | 17 |
| 3.6.3 Arguments              | 17 |
| 3.6.4 Notes                  | 17 |
| 3.6.5 Return Value           | 17 |
| 3.6.6 Command Example        | 17 |
| 3.6.7 Return Example         | 17 |
| 3.7 Command set ui_image     | 18 |
| 3.7.1 Command usage          | 18 |
| 3.7.2 Description            | 18 |
| 3.7.3 Arguments              | 18 |
| 3.7.4 Notes                  | 18 |
| 3.7.5 Return Value           | 18 |
| 3.7.6 Command Example        | 18 |
| 3.7.7 Return Example         | 18 |
| 3.8 Command set ui_page      | 19 |
| 3.8.1 Command usage          | 19 |
| 3.8.2 Description            | 19 |
| 3.8.3 Arguments              | 19 |
| 3.8.4 Notes                  | 19 |
| 3.8.5 Return Value           | 19 |
| 3.8.6 Command Example        | 19 |
| 3.8.7 Return Example         | 19 |
| 3.9 Command set ui_indicator | 20 |
| 3.9.1 Command usage          | 20 |
| 3.9.2 Description            | 20 |
| 3.9.3 Arguments              | 20 |
| 3.9.4 Notes                  | 20 |
| 3.9.5 Return Value           | 20 |
| 3.9.6 Command Example        | 20 |
| 3.9.7 Return Example         | 20 |
| 3.10 Command set ui_slider   | 21 |
| 3.10.1 Command usage         | 21 |
| 3.10.2 Description           | 21 |
| 3.10.3 Arguments             | 21 |
| 3.10.4 Notes                 | 21 |
| 3.10.5 Return Value          | 21 |
| 3.10.6 Command Example       | 21 |
| 3.10.7 Return Example        | 21 |
| 3.11 Command set ui_redirect | 22 |
| 3.11.1 Command usage         | 22 |
| 3.11.2 Description           | 22 |
| 3.11.3 Arguments             | 22 |
| 3.11.4 Notes                 | 22 |
| 3.11.5 Return Value          | 22 |
| 3.11.6 Command Example       | 22 |
| 3.11.7 Return Example        | 22 |
| 3.12 Command set ui_revert   | 23 |
| 3.12.1 Command usage         | 23 |
| 3.12.2 Description           | 23 |
| 3.12.3 Arguments             | 23 |
| 3.12.4 Notes                 | 23 |
| 3.12.5 Return Value          | 23 |
| 3.12.6 Command Example       | 23 |
| 3.12.7 Return Example        | 23 |



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Table of Contents continued...

|                              |           |
|------------------------------|-----------|
| <b>4 Command get</b>         | <b>24</b> |
| 4.1 Command get events       | 24        |
| 4.1.1 Command usage          | 24        |
| 4.1.2 Description            | 24        |
| 4.1.3 Arguments              | 24        |
| 4.1.4 Notes                  | 24        |
| 4.1.5 Return Value           | 24        |
| 4.1.6 Command Examples       | 24        |
| 4.1.7 Return Examples        | 24        |
| 4.2 Command get var          | 25        |
| 4.2.1 Command usage          | 25        |
| 4.2.2 Description            | 25        |
| 4.2.3 Arguments              | 25        |
| 4.2.4 Notes                  | 25        |
| 4.2.5 Return Value           | 25        |
| 4.2.6 Command Examples       | 25        |
| 4.2.7 Return Examples        | 25        |
| 4.3 Command get ui           | 26        |
| 4.3.1 Command usage          | 26        |
| 4.3.2 Description            | 26        |
| 4.3.3 Arguments              | 26        |
| 4.3.4 Notes                  | 26        |
| 4.3.5 Return Value           | 26        |
| 4.3.6 Command Examples       | 26        |
| 4.3.7 Return Examples        | 26        |
| 4.4 Command get ui_button    | 27        |
| 4.4.1 Command usage          | 27        |
| 4.4.2 Description            | 27        |
| 4.4.3 Arguments              | 27        |
| 4.4.4 Notes                  | 27        |
| 4.4.5 Return Value           | 27        |
| 4.4.6 Command Examples       | 27        |
| 4.5 Command get ui_indicator | 28        |
| 4.5.1 Command usage          | 28        |
| 4.5.2 Description            | 28        |
| 4.5.3 Arguments              | 28        |
| 4.5.4 Notes                  | 28        |
| 4.5.5 Return Value           | 28        |
| 4.5.6 Command Examples       | 28        |
| 4.6 Command get ui_slider    | 29        |
| 4.6.1 Command usage          | 29        |
| 4.6.2 Description            | 29        |
| 4.6.3 Arguments              | 29        |
| 4.6.4 Notes                  | 29        |
| 4.6.5 Return Value           | 29        |



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Table of Contents continued...

|   |           |
|---|-----------|
| <b>5 Command send</b>                   | <b>30</b> |
| 5.1 Command send gc                     | 32        |
| 5.1.1 Command usage                     | 32        |
| 5.1.2 Description                       | 32        |
| 5.1.3 Arguments                         | 32        |
| 5.1.4 Notes                             | 32        |
| 5.1.5 Return Value                      | 33        |
| 5.1.6 Command Examples                  | 33        |
| 5.1.7 Return Examples                   | 33        |
| 5.2 Command send tcp                    | 34        |
| 5.2.1 Command usage                     | 34        |
| 5.2.2 Description                       | 34        |
| 5.2.3 Arguments                         | 34        |
| 5.2.4 Notes                             | 34        |
| 5.2.5 Return Value                      | 34        |
| 5.2.6 Command Examples                  | 34        |
| 5.2.7 Return Examples                   | 34        |
| <b>6 Command preset</b>                 | <b>35</b> |
| 6.1 Command preset add                  | 35        |
| 6.1.1 Command usage                     | 35        |
| 6.1.2 Description                       | 35        |
| 6.1.3 Arguments                         | 35        |
| 6.1.4 Notes                             | 35        |
| 6.1.5 Return Value                      | 35        |
| 6.1.6 Command Examples                  | 35        |
| 6.1.7 Return Examples                   | 35        |
| 6.2 Command preset delay                | 36        |
| 6.2.1 Command usage                     | 36        |
| 6.2.2 Description                       | 36        |
| 6.2.3 Arguments                         | 36        |
| 6.2.4 Notes                             | 36        |
| 6.2.5 Return Value                      | 36        |
| 6.2.6 Command Examples                  | 36        |
| 6.2.7 Return Examples                   | 36        |
| 6.3 Command preset delete               | 37        |
| 6.3.1 Command usage                     | 37        |
| 6.3.2 Description                       | 37        |
| 6.3.3 Arguments                         | 37        |
| 6.3.4 Notes                             | 37        |
| 6.3.5 Return Value                      | 37        |
| 6.3.6 Command Examples                  | 37        |
| 6.3.7 Return Examples                   | 37        |
| 6.4 Command preset load                 | 38        |
| 6.4.1 Command usage                     | 38        |
| 6.4.2 Description                       | 38        |
| 6.4.3 Arguments                         | 38        |
| 6.4.4 Notes                             | 38        |
| 6.4.5 Return Value                      | 38        |
| 6.4.6 Command Examples                  | 38        |
| 6.4.7 Return Examples                   | 38        |
| <b>Appendix A - How to HTTP request</b> | <b>39</b> |
| <b>Appendix B – Preset Logic</b>        | <b>41</b> |



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 1 Introduction

This document describes everything that a developer needs to be aware of to use the DIRECTOR Control UI command guide and develop client control applications for 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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 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>**
- 
- send gc 172.30.20.129 4998 setstate,1:1,1<cr>
  - send tcp 172.30.30.221 1234 "command"<cr>
  - set ui\_button Ulexample btn01 position down<cr>





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3 Command set

The **set** commands are used to change various conditions.

#### 3.1 Command set listener

##### 3.1.1 Command usage

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

##### 3.1.2 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.

##### 3.1.3 Arguments

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

##### 3.1.4 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.

##### 3.1.5 Return Value

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

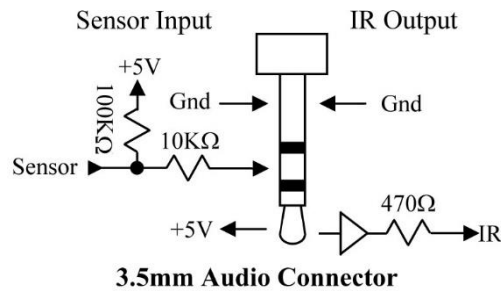
##### 3.1.6 Command Examples

```
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>
```

### 3.1.7 Return Examples

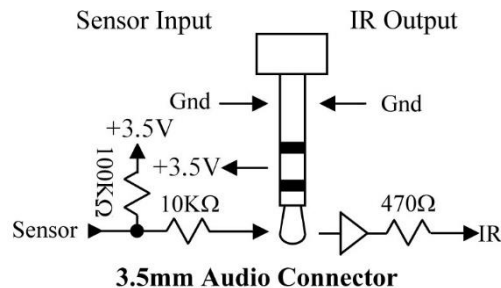
```
set listener success<cr>
set listener error [incomplete]<cr>
set listener error [reserved notify port]<cr>
set listener error [invalid parameter]<cr>
set listener error [preset 'preset' not found]<cr>
set listener error [no active listener found]<cr>
set listener error [invalid device port]<cr>
```

### 3.1.8 GC-100 Series IR/Sensor Connector Wiring



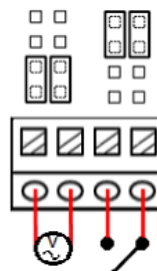
*Note: Use connector RING and SLEEVE for closed contact detection*

### 3.1.9 iTach and GlobalConnect Series IR/Sensor Connector Wiring



*Note: Use connector TIP and SLEEVE for closed contact detection*

### 3.1.10 Flex Link Relay & Sensor Cable Wiring



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

### 3.1.1.1 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

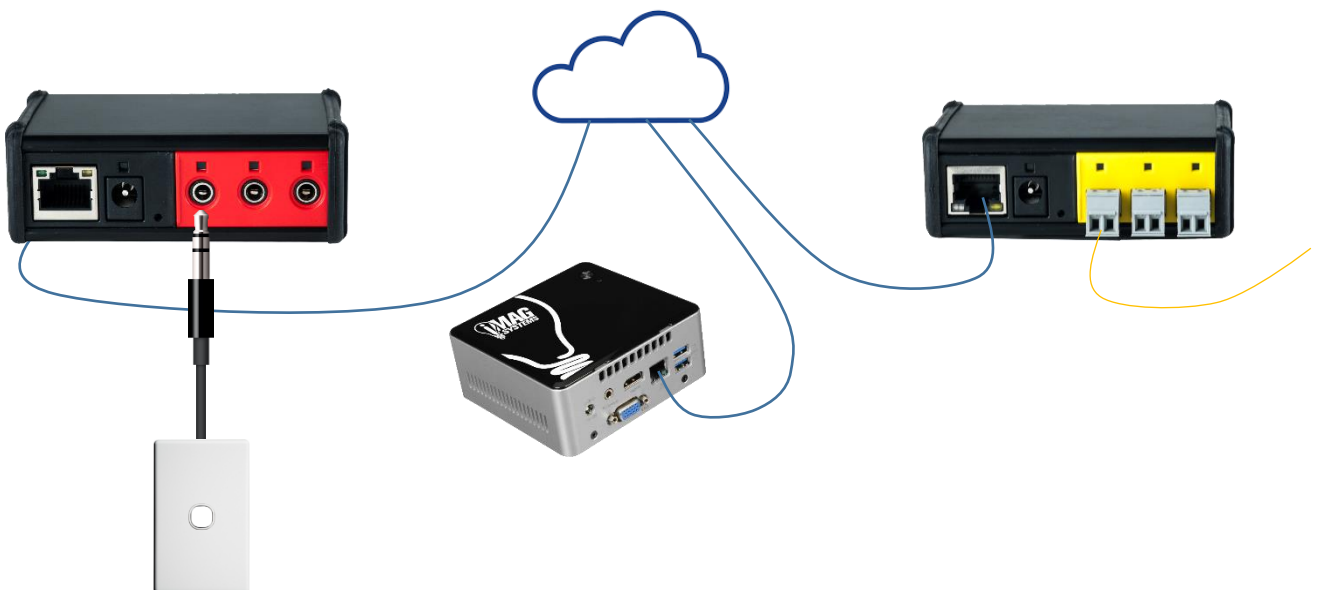
Not required

\* 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  $\pm 24V$ , with an "on" indication for voltages greater than 2.5V and "off" when less than 0.8V with an input impedance of  $\sim 100K\Omega$ .

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



ToC



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.2 Command set events

#### 3.2.1 Command usage

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

#### 3.2.2 Description

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

#### 3.2.3 Arguments

|                   |                                       |
|-------------------|---------------------------------------|
| <i>event_name</i> | Name of the event                     |
| <i>function</i>   | Currently only <b>state</b> supported |
| <i>value</i>      | enabled   disabled                    |

#### 3.2.4 Notes

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

#### 3.2.5 Return Value

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

#### 3.2.6 Command Examples

```
set events myEvent state enabled<cr>
set events myEvent state disabled<cr>
```

#### 3.2.7 Return Examples

```
set events state success<cr>

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.3 Command set var

#### 3.3.1 Command usage

set var [key:<security\_key>] <var\_name> <value> [delete]<cr>

#### 3.3.2 Description

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

#### 3.3.3 Arguments

|                 |   |
|-----------------|---|
| <i>var_name</i> | Name of the variable                                |
| <i>value</i>    | String   Integer   Boolean                          |
| [delete]        | Keyword used to delete the variable from the system |

#### 3.3.4 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.
- Used within a preset, <<slider\_value>> can be used in place of <value>.
- String: set var <var\_name> "a string"
- Integer: set var <var\_name> 100
- Boolean: set var <var\_name> true / false
- Refer **Appendix B – Preset Logic**

#### 3.3.5 Return Value

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

#### 3.3.6 Command Examples

```
set var myVar true<cr>
set var myVar "any string up to 256 characters"<cr>
set var myVar 100<cr>
set var myVar [delete]<cr>
set var key:abc123 myVar false<cr>
set var sliderValue <<slider_value>>
```

#### 3.3.7 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>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.4 Command set ui

#### 3.4.1 Command usage

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

#### 3.4.2 Description

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

#### 3.4.3 Arguments

|                |   |
|----------------|---|
| <i>ui_name</i> | name of the User Interface or preset logic <<ui_name>>                        |
| <i>service</i> | enabled   disabled   logout   |
| <i>clients</i> | Optional with enabled service to set the maximum client limit from 1 to 100   |
| <i>pin</i>     | Optional with enabled service to set a fixed or random 4 digit login pin code |

#### 3.4.4 Notes

- Service logout is the same as disabled then enabled.
- Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.

#### 3.4.5 Return Value

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

#### 3.4.6 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>
set ui <<ui_name>> logout
```

#### 3.4.7 Return Examples

```
set ui success<cr>

set ui error [incomplete]<cr>
set ui error [ui 'myUI' 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>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.5 Command set ui\_button

#### 3.5.1 Command usage

set ui\_button [key:<security\_key>] <ui\_name> <button\_name / button\_group> <function> [<button\_position>] [<value>] [<bypass>]<cr>

#### 3.5.2 Description

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

#### 3.5.3 Arguments

|                                   |   |
|-----------------------------------|---|
| <i>ui_name</i>                    | Name of the User Interface or preset logic <<ui_name>>                          |
| <i>button_name / button_group</i> | Name of the button or preset logic <<button_name>><br>Or name of a button group |
| <i>function</i>                   | position   state   text   press   |
| <i>button_position</i>            | up   down   both  |
| <i>value</i>                      | up   down, enabled   disabled or text string depending on the function          |
| <i>bypass</i>                     | Optional keyword used to perform all button actions except executing the preset |

#### 3.5.4 Notes

- The UI service must be enabled.
- Used within a preset, <<button\_name>> can be used in place of <button\_name>.
- Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.
- For button **text**, \r can be used for a line break.
- For button **press**, optional keyword '**bypass**' is useful for status updates where only changes to UI are required and not executing presets.
- For button **press**, the pressed button must be enabled, same as with a physical press.
- 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 and Radio Toggle button types.
- **function > state** uses **enabled | disabled**
- **button\_position** is only required for function text for Toggle and Radio Toggle button types otherwise 'up' can only be used.

- |                    |                 |                       |                 |
|--------------------|-----------------|-----------------------|-----------------|
| • <b>Momentary</b> | • <b>Toggle</b> | • <b>Radio Toggle</b> | • <b>Repeat</b> |
| ○ state            | ○ position      | ○ position            | ○ state         |
| ○ text             | ○ state         | ○ state               | ○ text          |
| ○ press            | ○ text          | ○ text                | ○ press         |
|                    | ○ press         | ○ press               |                 |

#### 3.5.5 Return Value

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.5.6 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\rHere"<cr>
set ui_button myUI myButtonGroup position up<cr>
set ui_button myUI myButtonGroup state disabled<cr>
set ui_button myUI myButton press<cr>
set ui_button key:abc123 myUI myButton press bypass<cr>
set ui_button <ui_name> myButton state disabled
```

### 3.5.7 Return Examples

```
set ui_button success<cr>

set ui_button error [incomplete]<cr>
set ui_button error [ui 'myUI' not found]<cr>
set ui_button error [button 'myButton' not found]<cr>
set ui_button error [invalid value 'xxx']<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>
set ui_button error [button disabled]<cr>
```





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.6 Command set ui\_label

#### 3.6.1 Command usage

set ui\_label [key:<security\_key>] <ui\_name> <label\_name> <function> <value><cr>

#### 3.6.2 Description

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

#### 3.6.3 Arguments

|                   |  |
|-------------------|--|
| <i>ui_name</i>    | Name of the User Interface or preset logic <<ui_name>> |
| <i>label_name</i> | Name of the label                                      |
| <i>function</i>   | color   visibility   text                              |
| <i>value</i>      | depending on the function                              |

#### 3.6.4 Notes

- The UI service must be enabled.
- function color uses HEX RGB color code from 000000 to FFFFFFFF
- function visibility uses true | false
- For label text, \r can be used for a line break.
- Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.

#### 3.6.5 Return Value

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

#### 3.6.6 Command Examples

```
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 "My\rText"<cr>
set ui_label <<ui_name>> myLabel text "magic"
```

#### 3.6.7 Return Examples

```
set ui_label color success<cr>
set ui_label visibility success<cr>
set ui_label text success<cr>

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.7 Command set ui\_image

#### 3.7.1 Command usage

set ui\_image [key:<security\_key>] <ui\_name> <image\_name> <function> <value><cr>

#### 3.7.2 Description

The command **set ui\_image** is used to control an image within an active User Interface.

#### 3.7.3 Arguments

|                   |  |
|-------------------|--|
| <i>ui_name</i>    | Name of the User Interface or preset logic <<ui_name>> |
| <i>image_name</i> | Name of the image                                      |
| <i>function</i>   | visibility   |
| <i>value</i>      | true   false   |

#### 3.7.4 Notes

- The UI service must be enabled.
- Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.

#### 3.7.5 Return Value

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

#### 3.7.6 Command Examples

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

#### 3.7.7 Return Examples

```
set ui_image visibility success<cr>

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.8 Command set ui\_page

#### 3.8.1 Command usage

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

#### 3.8.2 Description

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

#### 3.8.3 Arguments

|                  |  |
|------------------|--|
| <i>ui_name</i>   | Name of the User Interface or preset logic <<ui_name>> |
| <i>page_name</i> | Name of the page                                       |

#### 3.8.4 Notes

- The UI service must be enabled.
- Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.

#### 3.8.5 Return Value

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

#### 3.8.6 Command Examples

```
set ui_page myUI myPage<cr>
set ui_page key:abc123 myUI Page2<cr>
set ui_page <<ui_name>> Page1
```

#### 3.8.7 Return Examples

```
set ui_page success<cr>

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

### 3.9 Command set ui\_indicator

#### 3.9.1 Command usage

set ui\_indicator [key:<security\_key>] <ui\_name> <indicator\_name> <function> <value><cr>

#### 3.9.2 Description

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

#### 3.9.3 Arguments

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

#### 3.9.4 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>.

#### 3.9.5 Return Value

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

#### 3.9.6 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>>
```

#### 3.9.7 Return Examples

```
set ui_indicator value success<cr>

set ui_indicator error [incomplete]<cr>
set ui_indicator error [ui 'myUI' not found]<cr>
set ui_indicator error [indicator 'myIndicator' not found]<cr>
set ui_indicator error [invalid parameter]<cr>
set ui_indicator error [service disabled]<cr>
```

### 3.10 Command set ui\_slider

#### 3.10.1 Command usage

set ui\_slider [key:<security\_key>] <ui\_name> <slider\_name> <function> <value><cr>

#### 3.10.2 Description

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

#### 3.10.3 Arguments

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

#### 3.10.4 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>.

#### 3.10.5 Return Value

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

#### 3.10.6 Command Examples

```
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>>
```

#### 3.10.7 Return Examples

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

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.11 Command set ui\_redirect

#### 3.11.1 Command usage

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

#### 3.11.2 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.

#### 3.11.3 Arguments

|                         |   |
|-------------------------|---|
| <i>current_ui_name</i>  | Name of the current User Interface                        |
| <i>redirect_ui_name</i> | Name of the replacement User Interface                    |
| <i>page_name</i>        | Optional page selection of the replacement User Interface |

#### 3.11.4 Notes

- The UI service must be enabled.
- Refer command set ui\_revert.

#### 3.11.5 Return Value

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

#### 3.11.6 Command Examples

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

#### 3.11.7 Return Examples

```
set ui_redirect success<cr>

set ui_redirect error [incomplete]<cr>
set ui_redirect error [ui 'myUI' not found]<cr>
set ui_redirect error [service disabled]<cr>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 3.12 Command set ui\_revert

#### 3.12.1 Command usage

set ui\_revert [key:<security\_key>] <original\_ui\_name><cr>

#### 3.12.2 Description

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

#### 3.12.3 Arguments

|                         |  |
|-------------------------|--|
| <i>original_ui_name</i> | Name of the original redirected User Interface |
|-------------------------|--|

#### 3.12.4 Notes

- The UI service must be enabled.
- Refer command set ui\_redirect.

#### 3.12.5 Return Value

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

#### 3.12.6 Command Examples

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

#### 3.12.7 Return Examples

```
set ui_revert success<cr>

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 4 Command get

The **get** commands are used to retrieve various information.

#### 4.1 Command get events

##### 4.1.1 Command usage

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

##### 4.1.2 Description

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

##### 4.1.3 Arguments

|                   |                                       |
|-------------------|---------------------------------------|
| <i>event_name</i> | Name of the event                     |
| <i>function</i>   | Currently only <b>state</b> supported |

##### 4.1.4 Notes

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

##### 4.1.5 Return Value

| command | <space> | mode   | <space> | status                                    | <space> | value    | terminator |
|---------|---------|--------|---------|---|---------|----------|------------|
| get     | <space> | events | <space> | <i>success data /<br/>error [message]</i> | <space> | <string> | <cr>       |

##### 4.1.6 Command Example

```
get events state myEvent<cr>
```

##### 4.1.7 Return Examples

```
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>
```





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 4.2 Command get var

#### 4.2.1 Command usage

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

#### 4.2.2 Description

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

#### 4.2.3 Arguments

|                 |                      |
|-----------------|----------------------|
| <i>var_name</i> | Name of the variable |
|-----------------|----------------------|

#### 4.2.4 Notes

- Refer **Appendix B – Preset Logic**

#### 4.2.5 Return Value

| command | <space> | mode | <space> | status                                    | <space> | value    | terminator |
|---------|---------|------|---------|---|---------|----------|------------|
| get     | <space> | var  | <space> | <i>success data /<br/>error [message]</i> | <space> | <string> | <cr>       |

#### 4.2.6 Command Example

```
get var myVar<cr>
```

#### 4.2.7 Return Examples

```
get var success a string<cr>
```

```
get var success 100<cr>
```

```
get var success true<cr>
```

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

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 4.3 Command get ui

#### 4.3.1 Command usage

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

#### 4.3.2 Description

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

#### 4.3.3 Arguments

|                |  |
|----------------|--|
| <i>ui_name</i> | Name of the User Interface or preset logic <<ui_name>> |
|----------------|--|

#### 4.3.4 Notes

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

#### 4.3.5 Return Value

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

#### 4.3.6 Command Example

```
get ui myUI<cr>
get ui <<ui_name>>
```

#### 4.3.7 Return Examples

```
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 'myUI' not found]<cr>
get ui error [service disabled]<cr>
```

### 4.4 Command get ui\_button

#### 4.4.1 Command usage

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

#### 4.4.2 Description

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

#### 4.4.3 Arguments

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

#### 4.4.4 Notes

- The UI service must be enabled.
  - Used within a preset, <<ui\_name>> can be used in place of <ui\_name>.
  - Use the command **set ui\_button** to change this setting.
  - function > position** uses **up | down**
  - function > state** uses **enabled | disabled**
- 
- |  |   |   |   |
|--|---|---|---|
| <ul style="list-style-type: none"> <li><b>Momentary</b> <ul style="list-style-type: none"> <li>state</li> <li>press</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li><b>Toggle</b> <ul style="list-style-type: none"> <li>position</li> <li>state</li> <li>press</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li><b>Radio Toggle</b> <ul style="list-style-type: none"> <li>position</li> <li>state</li> <li>press</li> </ul> </li> </ul> | <ul style="list-style-type: none"> <li><b>Repeat</b> <ul style="list-style-type: none"> <li>state</li> <li>press</li> </ul> </li> </ul> |
|--|---|---|---|

#### 4.4.5 Return Value

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

#### 4.4.6 Command Examples

```
get ui_button myUI myButton position<cr>
get ui_button key:abc123 myUI myButton state<cr>
get ui_button <<ui_name>> <<button_name>> position
```

#### 4.4.7 Return Examples

```
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>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 4.5 Command get ui\_indicator

#### 4.5.1 Command usage

get ui\_indicator [key:<security\_key>] <ui\_name> <indicator\_name> <function><cr>

#### 4.5.2 Description

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

#### 4.5.3 Arguments

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

#### 4.5.4 Notes

- 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>.

#### 4.5.5 Return Value

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

#### 4.5.6 Command Examples

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

#### 4.5.7 Return Examples

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

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 4.6 Command get ui\_slider

#### 4.6.1 Command usage

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

#### 4.6.2 Description

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

#### 4.6.3 Arguments

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

#### 4.6.4 Notes

- 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>.

#### 4.6.5 Return Value

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

#### 4.6.6 Command Examples

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

#### 4.6.7 Return Examples

```
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 [ui 'myUI' not found]<cr>
get ui_slider error [slider 'mySlider' not found]<cr>
get ui_slider error [invalid parameter]<cr>
get ui_slider error [service disabled]<cr>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 5 Command send

The **send** commands are used to send either general TCP or Global Caché commands. Any TCP controllable device or Global Caché device can be controlled.

References in this manual to \x0D and <cr> both refer to a carriage return or \r, hex 0x0D, decimal 13.

The use of \x0D is the standard method of using and displaying all hexadecimal characters.

The use of \x within any command string will be converted to hexadecimal before sending.

Any received hexadecimal characters will be escaped the same with \x, so often the peripheral device's feedback string is terminated with \x0D or \x0D\x0A. This is something to be aware of when using feedback comparison option "equals" over "contains". The specified "equals" string must also include all terminators or otherwise hidden hexadecimal non ASCII characters.

Sending any hexadecimal value \x00 to \xFF is supported.

Only non printable ASCII hexadecimal characters received will be terminated with \x.

<cr> is used to specifically show the termination of the Director Control UI API command.

#### **send tcp**

The command send tcp will open a TCP connection to the specified IP Address and Port then send the specified command string. The TCP connection will be maintained for 2min for any consecutive commands before being disconnected.

The feedback operation is optional as follows:

##### **a) No feedback required**

When no feedback is required, then nothing is specified.

The send tcp command will result in a Boolean true with a successful TCP connection otherwise false.

eg: send tcp 192.168.1.1 1234 "my command"<cr>

##### **b) External device feedback**

When the external device's original feedback is required, the keyword "reply" is used.

The send tcp command result is the returned string only.

eg: send tcp 169.254.100.1 1234 "my command" reply<cr>

##### **c) Comparison contains**

When an instance of a string is required in the feedback, the keyword "contains" along with the string to find is specified.

The send tcp command result is Boolean true/false as a result of the comparison.

eg: send tcp 169.254.100.1 1234 "my command\x0D" contains "command"<cr>

##### **d) Comparison equals**

When an exact match is required of the feedback, the keyword "equals" along with the string to find specified.

The send tcp command result is Boolean true/false as a result of the comparison.

eg: send tcp 169.254.100.1 1234 "my command\x0D" equals "my command\x0D"<cr>



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### **send gc**

The command send gc is similar to send tcp except operates with specific Global Cache conditions. It will also open a TCP connection to the specified IP Address and Port then send the specified Global Cache command string automatically appended with port 4998. The TCP connection will be maintained for 2min for any consecutive commands before being disconnected.

Variations of send tcp include the following:

#### **a) Reserved port numbers**

Only ports 4998, 4999 & 5000 are valid.

#### **b) Command string**

All command strings are strictly Global Cache API without the need for termination with \x0D within the command string itself. This will automatically be appended before sending to the device. Note: The send gc command itself still needs to be terminated with <cr>

eg: send gc 169.254.1.70 4998 "setstate,1:1,1"<cr>

#### **c) Command results**

Command result depends on the network port used.

Commands using port 4998 result in Boolean true/false relative to the specific function specified. So for example a relay closure command "setstate,1:1,1" will confirm the correct device feedback.

Commands using port 4999 or 5000 for serial devices result in the same feedback results as send tcp with return string comparison. Refer send tcp for details.

eg: send tcp 169.254.1.70 4998 "GC command"<cr>

eg: send tcp 169.254.1.70 4999 "GC command"<cr>

eg: send tcp 169.254.1.70 4999 "GC command" reply<cr>

eg: send tcp 169.254.1.70 4999 "GC command" contains "GC feedback"<cr>

eg: send tcp 169.254.1.70 4999 "GC command" equals "GC feedback\x0D"<cr>

Support for UDP and Telnet devices will be available in future updates.



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 5.1 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](http://www.globalcache.com).

#### 5.1.1 Command usage

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

#### 5.1.2 Description

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

#### 5.1.3 Arguments

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

#### 5.1.4 Notes

- 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/](http://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.





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 5.1.5 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> = <feedback_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 [<feedback_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 [<feedback_string>]
```

### 5.1.6 Examples

```
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>
```

### 5.1.7 Return Examples

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 5.2 Command send tcp

#### 5.2.1 Command usage

send tcp [key:<security\_key>] <address> <port> "<command>" [<feedback> ["<feedback\_string>"]]<cr>

#### 5.2.2 Description

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

#### 5.2.3 Arguments

|                        |   |
|------------------------|---|
| <i>address</i>         | TCP IP address  |
| <i>port</i>            | TCP port  |
| <i>command</i>         | Command string to send to TCP device                                |
| <i>feedback</i>        | Keyword <b>reply</b> , <b>equals</b> or <b>contains</b> (optional)  |
| <i>feedback_string</i> | Expected feedback string used with <b>equals</b> or <b>contains</b> |

#### 5.2.4 Notes

- 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.
- ***feedback\_string*** contains the expected device's feedback string result.
- Use keyword "[**disconnect**]" in place of ***command*** string to terminate the connection.

#### 5.2.5 Return Value

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

#### 5.2.6 Examples

```
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>
```

#### 5.2.7 Return Examples

```
send tcp success<cr>
send tcp success [<feedback>]<cr>
send tcp error [incomplete]<cr>
send tcp error [device '172.30.1.111' not found]<cr>
send tcp error [device '172.30.1.111:1000' not connected]<cr>

send tcp success [disconnected]<cr>
send tcp error [invalid parameter]<cr>
send tcp error [<feedback>]<cr>
```

ToC



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

## 6 Command preset

The preset commands are used to store and apply a series of commands available from this manual. Refer Appendix B - Preset logic for logic that can be applied within a preset.

### 6.1 Command preset add

#### 6.1.1 Command usage

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

#### 6.1.2 Description

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

#### 6.1.3 Arguments

|                    |                                |
|--------------------|--------------------------------|
| <i>preset_name</i> | The name defined as the preset |
| <i>preset_data</i> | A valid Control Command string |

#### 6.1.4 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.

#### 6.1.5 Return Value

| command | <space> | mode | <space> | name           | <space> | status                                     | terminator |
|---------|---------|------|---------|----------------|---------|--|------------|
| preset  | <space> | add  | <space> | <i>preset1</i> | <space> | <i>success / error</i><br><i>[message]</i> | <cr>       |

#### 6.1.6 Command Examples

```
preset add preset1 send gc 172.30.20.129 4998 setstate,1:1,1<cr>
preset add key:abc123 preset1 send gc 172.30.20.112 4999 "My String"<cr>
```

#### 6.1.7 Return Examples

```
preset add preset1 success<cr>

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 6.2 Command preset delay

#### 6.2.1 Command usage

preset delay [key:<security\_key>] <milliseconds>

#### 6.2.2 Description

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

#### 6.2.3 Arguments

|                     |                                       |
|---------------------|---------------------------------------|
| <i>milliseconds</i> | Delay time in milliseconds up to 9999 |
|---------------------|---------------------------------------|

#### 6.2.4 Notes

- This command can only be used within a preset.

#### 6.2.5 Return Value

None

#### 6.2.6 Command Examples

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

#### 6.2.7 Return Example

```
No return is given for a valid command

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 6.3 Command preset delete

#### 6.3.1 Command usage

preset delete [key:<security\_key>] <preset\_name><cr>

#### 6.3.2 Description

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

#### 6.3.3 Arguments

|                    |                                |
|--------------------|--------------------------------|
| <i>preset_name</i> | The name defined as the preset |
|--------------------|--------------------------------|

#### 6.3.4 Notes

#### 6.3.5 Return Value

| command | <space> | mode   | <space> | name           | <space> | status                               | terminator |
|---------|---------|--------|---------|----------------|---------|--------------------------------------|------------|
| preset  | <space> | delete | <space> | <i>preset1</i> | <space> | <i>success / error<br/>[message]</i> | <cr>       |

#### 6.3.6 Command Examples

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

#### 6.3.7 Return Examples

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 6.4 Command preset load

#### 6.4.1 Command usage

preset load [key:<security\_key>] <preset\_name> [delay]<cr>

#### 6.4.2 Description

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

#### 6.4.3 Arguments

|                    |  |
|--------------------|--|
| <i>preset_name</i> | The name defined as the preset                       |
| <i>delay</i>       | Delay in minutes before preset is applied (optional) |

#### 6.4.4 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.

#### 6.4.5 Return Value

| command | <space> | mode | <space> | name           | <space> | status                                     | terminator |
|---------|---------|------|---------|----------------|---------|--|------------|
| preset  | <space> | load | <space> | <i>preset1</i> | <space> | <i>success / error</i><br><i>[message]</i> | <cr>       |

#### 6.4.6 Command Examples

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

#### 6.4.7 Return Examples

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

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



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix A - How to HTTP request

GET = `http://<controllerURL>/api/command/<DIRECTOR_API_COMMAND>/<KEY>`

POST = `http://<controllerURL>/api/command{"cmd": "<DIRECTOR_API_COMMAND>", "key": "<KEY>"}`

#### Example 1: POST - ajax

```
<script language='JavaScript' type='text/javascript'>
  var controllerIP = '169.254.1.1'; *change this to the same IP address as the SDVoE Director controller
  var BaseURL = 'http://' + controllerIP + '/api/command';
  var MAXIMUM_WAITING_TIME = 5000; *timeout in milliseconds
  var CheckStatusTimer;
  var key = '123xyz'; *replace this with the generated security key
  var command = '123xyz'; *replace with any 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 SDVoE Director controller
  var BaseURL = 'http://' + controllerIP + '/api/command';
  var key = '123xyz'; *replace this with the generated security key
  var command = '123xyz'; *replace with any 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>
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### 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) {
      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>
```





# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix 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 subtract 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.



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix B – 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:

```
substr(<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))  
    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]
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix B – 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 lb101 text "good"
}else{
    set ui_label AT01 lb101 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 lb101 text "good"
}elseif (trim(get var myVar) == "state,1:1,0" {
    set ui_label AT01 lb101 text "bad"
} else {
    set ui_label AT01 lb101 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 lb101 text "good"
}elseif (get var myVar == "state,1:1,0"){
    set ui_label AT01 lb101 text "bad"
} else {
    set ui_label AT01 lb101 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"
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix B – 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.

### Appendix B – 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  
}
```

### Appendix B – 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>  
}
```



# DIRECTOR CONTROL UI

## COMMAND GUIDE V1.3.6

### Appendix B – 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  
}
```