SDVoE用コントローラー

SZ-SDCNT100

コマンドガイド

2024 年 4 月

Rev. 1.1

Table of Contents

1 Introduction 26
- 1.1 License Requirements 26
- 1.2 Telnet Connection and API Commands 26
- 1.3 HTTP Requests 26

2 Command Overview 26

3 System Commands 27

- 3.1 Command default 27
  - 3.1.1 Command usage 27
  - 3.1.2 Description 27
  - 3.1.3 Arguments 27
  - 3.1.4 Notes 27
  - 3.1.5 Return Value 27
  - 3.1.6 Command Examples 27
  - 3.1.7 Return Examples 27
- 3.2 Command reboot 28
  - 3.2.1 Command usage 28
  - 3.2.2 Description 28
  - 3.2.3 Arguments 28
  - 3.2.4 Notes 28
  - 3.2.5 Return Value 28
  - 3.2.6 Command Examples 28
  - 3.2.7 Return Examples 28

4 Command join 29

- 4.1 Command join fast 30
  - 4.1.1 Command usage 30
  - 4.1.2 Description 30
  - 4.1.3 Arguments 30
  - 4.1.4 Notes 30
  - 4.1.5 Return Value 30
  - 4.1.6 Command Examples 31
  - 4.1.7 Return Examples 31
- 4.2 Command join sync 32
  - 4.2.1 Command usage 32
  - 4.2.2 Description 32
  - 4.2.3 Arguments 32
  - 4.2.4 Notes 32
  - 4.2.5 Return Value 32
  - 4.2.6 Command Examples 32
  - 4.2.7 Return Examples 32
- 4.3 Command join sync_scale 33
  - 4.3.1 Command usage 33
  - 4.3.2 Description 33
  - 4.3.3 Arguments 33
  - 4.3.4 Notes 33
  - 4.3.5 Return Value 34
  - 4.3.6 Command Examples 34
  - 4.3.7 Return Examples 34
- 4.4 Command join adv 35
  - 4.4.1 Command usage 35
  - 4.4.2 Description 35
  - 4.4.3 Arguments 35
  - 4.4.4 Notes 35
  - 4.4.5 Return Value 35
  - 4.4.6 Command Examples 36
  - 4.4.7 Return Examples 36

Table of Contents continued…

4.5 Command join audio_a
- 4.5.1 Command usage 37
- 4.5.2 Description 37
- 4.5.3 Arguments 37
- 4.5.4 Notes 37
- 4.5.5 Return Value 37
- 4.5.6 Command Examples 37
- 4.5.7 Return Examples 37

4.6 Command join audio_d 38
- 4.6.1 Command usage 38
- 4.6.2 Description 38
- 4.6.3 Arguments 38
- 4.6.4 Notes 38
- 4.6.5 Return Value 38
- 4.6.6 Command Examples 38
- 4.6.7 Return Examples 38

4.7 Command join ir 39
- 4.7.1 Command usage 39
- 4.7.2 Description 39
- 4.7.3 Arguments 39
- 4.7.4 Notes 39
- 4.7.5 Return Value 39
- 4.7.6 Command Examples 39
- 4.7.7 Return Examples 39

4.8 Command join serial 40
- 4.8.1 Command usage 40
- 4.8.2 Description 40
- 4.8.3 Arguments 40
- 4.8.4 Notes 40
- 4.8.5 Return Value 40
- 4.8.6 Command Examples 40
- 4.8.7 Return Examples 40

4.9 Command join usb 41
- 4.9.1 Command usage 41
- 4.9.2 Description 41
- 4.9.3 Arguments 41
- 4.9.4 Notes 41
- 4.9.5 Return Value 41
- 4.9.6 Command Examples 41
- 4.9.7 Return Examples 41

4.10 Command join usb_hid 42
- 4.10.1 Command usage 42
- 4.10.2 Description 42
- 4.10.3 Arguments 42
- 4.10.4 Notes 42
- 4.10.5 Return Value 42
- 4.10.6 Command Examples 42
- 4.10.7 Return Examples 42

4.11 Command join multi 43
- 4.11.1 Command usage 43
- 4.11.2 Description 43
- 4.11.3 Arguments 43
- 4.11.4 Notes 43
- 4.11.5 Return Value 43
- 4.11.6 Command Examples 43
- 4.11.7 Return Examples 44

Table of Contents continued…

4.12 Command join wall 45
4.12.1 Command usage 45
4.12.2 Description 45
4.12.3 Arguments 45
4.12.4 Notes 45
4.12.5 Return Value 46
4.12.6 Command Example 46
4.12.7 Return Examples 46

4.13 Command join walladv 48
4.13.1 Command usage 48
4.13.2 Description 48
4.13.3 Arguments 48
4.13.4 Notes 48
4.13.5 Return Value 49
4.13.6 Command Example 49
4.13.7 Return Examples 49

Table of Contents continued…

5 Command leave 50

5.1 Command leave video 50
5.1.1 Command usage 50
5.1.2 Description 50
5.1.3 Arguments 50
5.1.4 Notes 50
5.1.5 Return Value 50
5.1.6 Command Examples 50
5.1.7 Return Examples 50

5.2 Command leave sub 51
5.2.1 Command usage 51
5.2.2 Description 51
5.2.3 Arguments 51
5.2.4 Notes 51
5.2.5 Return Value 51
5.2.6 Command Examples 51
5.2.7 Return Examples 51

5.3 Command leave av 52
5.3.1 Command usage 52
5.3.2 Description 52
5.3.3 Arguments 52
5.3.4 Notes 52
5.3.5 Return Value 52
5.3.6 Command Examples 52
5.3.7 Return Examples 52

5.4 Command leave audio_a 53
5.4.1 Command usage 53
5.4.2 Description 53
5.4.3 Arguments 53
5.4.4 Notes 53
5.4.5 Return Value 53
5.4.6 Command Examples 53
5.4.7 Return Examples 53

5.5 Command leave audio_d 54
5.5.1 Command usage 54
5.5.2 Description 54
5.5.3 Arguments 54
5.5.4 Notes 54
5.5.5 Return Value 54
5.5.6 Command Examples 54
5.5.7 Return Examples 54

5.6 Command leave all 55
5.6.1 Command usage 55
5.6.2 Description 55
5.6.3 Arguments 55
5.6.4 Notes 55
5.6.5 Return Value 55
5.6.6 Command Example 55
5.6.7 Return Examples 55

5.7 Command leave usb 56
5.7.1 Command usage 56
5.7.2 Description 56
5.7.3 Arguments 56
5.7.4 Notes 56
5.7.5 Return Value 56
5.7.6 Command Example 56
5.7.7 Return Examples 56

Table of Contents continued…

5.8 Command leave usb_hid 57
5.8.1 Command usage 57
5.8.2 Description 57
5.8.3 Arguments 57
5.8.4 Notes 57
5.8.5 Return Value 57
5.8.6 Command Example 57
5.8.7 Return Examples 57

Table of Contents continued…

6 Command stop 58

6.1 Command stop video 59
6.1.1 Command usage 59
6.1.2 Description 59
6.1.3 Arguments 59
6.1.4 Notes 59
6.1.5 Return Value 59
6.1.6 Command Example 59
6.1.7 Return Examples 59

6.2 Command stop sub 60
6.2.1 Command usage 60
6.2.2 Description 60
6.2.3 Arguments 60
6.2.4 Notes 60
6.2.5 Return Value 60
6.2.6 Command Example 60
6.2.7 Return Examples 60

6.3 Command stop av 61
6.3.1 Command usage 61
6.3.2 Description 61
6.3.3 Arguments 61
6.3.4 Notes 61
6.3.5 Return Value 61
6.3.6 Command Example 61
6.3.7 Return Examples 61

6.4 Command stop audio_a 62
6.4.1 Command usage 62
6.4.2 Description 62
6.4.3 Arguments 62
6.4.4 Notes 62
6.4.5 Return Value 62
6.4.6 Command Example 62
6.4.7 Return Examples 62

6.5 Command stop audio_d 63
6.5.1 Command usage 63
6.5.2 Description 63
6.5.3 Arguments 63
6.5.4 Notes 63
6.5.5 Return Value 63
6.5.6 Command Example 63
6.5.7 Return Examples 63

6.6 Command stop ir 64
6.6.1 Command usage 64
6.6.2 Description 64
6.6.3 Arguments 64
6.6.4 Notes 64
6.6.5 Return Value 64
6.6.6 Command Example 64
6.6.7 Return Examples 64

6.7 Command stop serial 65
6.7.1 Command usage 65
6.7.2 Description 65
6.7.3 Arguments 65
6.7.4 Notes 65
6.7.5 Return Value 65
6.7.6 Command Examples 65
6.7.7 Return Examples 65

Table of Contents continued…

6.8 Command stop all 66
6.8.1 Command usage 66
6.8.2 Description 66
6.8.3 Arguments 66
6.8.4 Notes 66
6.8.5 Return Value 66
6.8.6 Command Example 66
6.8.7 Return Examples 66

Table of Contents continued…

7 Command start 67

7.1 Command start video 67
7.1.1 Command usage 67
7.1.2 Description 67
7.1.3 Arguments 67
7.1.4 Notes 67
7.1.5 Return Value 67
7.1.6 Command Examples 67
7.1.7 Return Examples 67

7.2 Command start sub 69
7.2.1 Command usage 69
7.2.2 Description 69
7.2.3 Arguments 69
7.2.4 Notes 69
7.2.5 Return Value 69
7.2.6 Command Examples 69
7.2.7 Return Examples 69

7.3 Command start av 69
7.3.1 Command usage 69
7.3.2 Description 69
7.3.3 Arguments 69
7.3.4 Notes 69
7.3.5 Return Value 69
7.3.6 Command Examples 69
7.3.7 Return Examples 69

7.4 Command start audio_a 71
7.4.1 Command usage 71
7.4.2 Description 71
7.4.3 Arguments 71
7.4.4 Notes 71
7.4.5 Return Value 71
7.4.6 Command Examples 71
7.4.7 Return Examples 71

7.5 Command start audio_d 72
7.5.1 Command usage 72
7.5.2 Description 72
7.5.3 Arguments 72
7.5.4 Notes 72
7.5.5 Return Value 72
7.5.6 Command Examples 72
7.5.7 Return Examples 72

Table of Contents continued…

8 Command set 73
8.1 Command set for devices 73

8.1.1 Command set audio_io 74
8.1.1.1 Command usage 74
8.1.1.2 Description 74
8.1.1.3 Arguments 74
8.1.1.4 Notes 74
8.1.1.5 Return Value 74
8.1.1.6 Command Example 74
8.1.1.7 Return Example 74

8.1.2 Command set audio_out 75
8.1.2.1 Command usage 75
8.1.2.2 Description 75
8.1.2.3 Arguments 75
8.1.2.4 Notes 75
8.1.2.5 Return Value 75
8.1.2.6 Command Example 75
8.1.2.7 Return Example 75

8.1.3 Command set audio_source 76
8.1.3.1 Command usage 76
8.1.3.2 Description 76
8.1.3.3 Arguments 76
8.1.3.4 Notes 76
8.1.3.5 Return Value 76
8.1.3.6 Command Example 76
8.1.3.7 Return Example 76

8.1.4 Command set edid 77
8.1.4.1 Command usage 77
8.1.4.2 Description 77
8.1.4.3 Arguments 77
8.1.4.4 Notes 77
8.1.4.5 Return Value 77
8.1.4.6 Command Example 77
8.1.4.7 Return Example 78

8.1.5 Command set frame_converter 79
8.1.5.1 Command usage 79
8.1.5.2 Description 79
8.1.5.3 Arguments 79
8.1.5.4 Notes 79
8.1.5.5 Return Value 79
8.1.5.6 Command Examples 79
8.1.5.7 Return Examples 79

8.1.6 Command set scaler 80
8.1.6.1.1 Command usage 80
8.1.6.1.2 Description 80
8.1.6.1.3 Arguments 80
8.1.6.1.4 Notes 80
8.1.6.1.5 Return Value 80
8.1.6.1.6 Command Examples 80
8.1.6.1.7 Return Examples 80

8.1.7 Command set security 81
8.1.7.1 Command usage 81
8.1.7.2 Description 81
8.1.7.3 Arguments 81
8.1.7.4 Notes 81
8.1.7.5 Return Value 81
8.1.7.6 Command Example 81
8.1.7.7 Return Example 81

Table of Contents continued…

8.1.8 Command set video_compress 82
8.1.8.1 Command usage 82
8.1.8.2 Description 82
8.1.8.3 Arguments 82
8.1.8.4 Notes 82
8.1.8.5 Return Value 82
8.1.8.6 Command Example 82
8.1.8.7 Return Example 82

8.1.9 Command set video_mode 83
8.1.9.1 Command usage 83
8.1.9.2 Description 83
8.1.9.3 Arguments 83
8.1.9.4 Notes 83
8.1.9.5 Return Value 83
8.1.9.6 Command Example 83
8.1.9.7 Return Example 83

8.1.10 Command set video_mute 84
8.1.10.1 Command usage 84
8.1.10.2 Description 84
8.1.10.3 Arguments 84
8.1.10.4 Notes 84
8.1.10.5 Return Value 84
8.1.10.6 Command Example 84
8.1.10.7 Return Example 84

8.1.11 Command set video_source 85
8.1.11.1 Command usage 85
8.1.11.2 Description 85
8.1.11.3 Arguments 85
8.1.11.4 Notes 85
8.1.11.5 Return Value 85
8.1.11.6 Command Example 85
8.1.11.7 Return Example 85

Table of Contents continued…

8.2 Command set for system 86

8.2.1 Command set events 86
8.2.1.1 Command usage 86
8.2.1.2 Description 86
8.2.1.3 Arguments 86
8.2.1.4 Notes 86
8.2.1.5 Return Value 86
8.2.1.6 Command Example 86
8.2.1.7 Return Example 86

8.2.2 Command set listener *licensed feature 87

8.2.3 Command set presenter 88
8.2.3.1 Command usage 88
8.2.3.2 Description 88
8.2.3.3 Arguments 88
8.2.3.4 Notes 88
8.2.3.5 Return Value 88
8.2.3.6 Command Example 88
8.2.3.7 Return Example 88

8.2.4 Command set var 89
8.2.4.1 Command usage 89
8.2.4.2 Description 89
8.2.4.3 Arguments 89
8.2.4.4 Notes 89
8.2.4.5 Return Value 89
8.2.4.6 Command Example 89
8.2.4.7 Return Example 89

Table of Contents continued…

8.3 Command set for User Interfaces 90

8.3.1 Command set ui 90
8.3.1.1 Command usage 90
8.3.1.2 Description 90
8.3.1.3 Arguments 90
8.3.1.4 Notes 90
8.3.1.5 Return Value 90
8.3.1.6 Command Example 90
8.3.1.7 Return Example 90

8.3.2 Command set ui_button 91
8.3.2.1 Command usage 91
8.3.2.2 Description 91
8.3.2.3 Arguments 91
8.3.2.4 Notes 91
8.3.2.5 Return Value 91
8.3.2.6 Command Example 91
8.3.2.7 Return Example 92

8.3.3 Command set ui_image 93
8.3.3.1 Command usage 93
8.3.3.2 Description 93
8.3.3.3 Arguments 93
8.3.3.4 Notes 93
8.3.3.5 Return Value 93
8.3.3.6 Command Example 93
8.3.3.7 Return Example 93

8.3.4 Command set ui_indicator 94
8.3.4.1 Command usage 94
8.3.4.2 Description 94
8.3.4.3 Arguments 94
8.3.4.4 Notes 94
8.3.4.5 Return Value 94
8.3.4.6 Command Example 94
8.3.4.7 Return Example 94

8.3.5 Command set ui_label 95
8.3.5.1 Command usage 95
8.3.5.2 Description 95
8.3.5.3 Arguments 95
8.3.5.4 Notes 95
8.3.5.5 Return Value 95
8.3.5.6 Command Example 95
8.3.5.7 Return Example 95

8.3.6 Command set ui_page 96
8.3.6.1.1 Command usage 96
8.3.6.1.2 Description 96
8.3.6.1.3 Arguments 96
8.3.6.1.4 Notes 96
8.3.6.1.5 Return Value 96
8.3.6.1.6 Command Example 96
8.3.6.1.7 Return Example 96

8.3.7 Command set ui_redirect 97
8.3.7.1 Command usage 97
8.3.7.2 Description 97
8.3.7.3 Arguments 97
8.3.7.4 Notes 97
8.3.7.5 Return Value 97
8.3.7.6 Command Example 97
8.3.7.7 Return Example 97

Table of Contents continued…

8.3.8 Command set ui_revert 98
8.3.8.1 Command usage 98
8.3.8.2 Description 98
8.3.8.3 Arguments 98
8.3.8.4 Notes 98
8.3.8.5 Return Value 98
8.3.8.6 Command Example 98
8.3.8.7 Return Example 98

8.3.9 Command set ui_slider 99
8.3.9.1 Command usage 99
8.3.9.2 Description 99
8.3.9.3 Arguments 99
8.3.9.4 Notes 99
8.3.9.5 Return Value 99
8.3.9.6 Command Example 99
8.3.9.7 Return Example 99

Table of Contents continued…

9 Command get 100
9.1 Command get for devices 100

9.1.1 Command get api 100
9.1.1.1 Command usage 100
9.1.1.2 Description 100
9.1.1.3 Arguments 100
9.1.1.4 Notes 100
9.1.1.5 Return Value 100
9.1.1.6 Command Example 100
9.1.1.7 Return Examples 100

9.1.2 Command get audio_io 101
9.1.2.1 Command usage 101
9.1.2.2 Description 101
9.1.2.3 Arguments 101
9.1.2.4 Notes 101
9.1.2.5 Return Value 101
9.1.2.6 Command Example 101
9.1.2.7 Return Examples 101

9.1.3 Command get audio_out 102
9.1.3.1 Command usage 102
9.1.3.2 Description 102
9.1.3.3 Arguments 102
9.1.3.4 Notes 102
9.1.3.5 Return Value 102
9.1.3.6 Command Example 102
9.1.3.7 Return Examples 102

9.1.4 Command get audio_source 103
9.1.4.1 Command usage 103
9.1.4.2 Description 103
9.1.4.3 Arguments 103
9.1.4.4 Notes 103
9.1.4.5 Return Value 103
9.1.4.6 Command Example 103
9.1.4.7 Return Examples 103

9.1.5 Command get bandwidth / get rt_bandwidth 104
9.1.5.1 Command usage 104
9.1.5.2 Description 104
9.15.3 Arguments 104
9.1.5.4 Notes 104
9.1.5.5 Return Value 104
9.1.5.6 Command Example 104
9.1.5.7 Return Examples 104

9.1.6 Command get devices 105
9.1.6.1 Command usage 105
9.1.6.2 Description 105
9.1.6.3 Arguments 105
9.1.6.4 Notes 105
9.1.6.5 Return Value 105
9.1.6.6 Command Example 105
9.1.6.7 Return Examples 105

9.1.7 Command get display_status 106
9.1.7.1 Command usage 106
9.1.7.2 Description 106
9.1.7.3 Arguments 106
9.1.7.4 Notes 106
9.1.7.5 Return Value 106
9.1.7.6 Command Example 106
9.1.7.7 Return Examples 106

Table of Contents continued…

9.1.8 Command get edid 107
9.1.8.1 Command usage 107
9.1.8.2 Description 107
9.1.8.3 Arguments 107
9.1.8.4 Notes 107
9.1.8.5 Return Value 107
9.1.8.2 Command Example 107
9.1.8.7 Return Examples 107

9.1.9 Command get frame_converter 111
9.1.9.1 Command usage 111
9.1.9.2 Description 111
9.1.9.3 Arguments 111
9.1.9.4 Notes 111
9.1.9.5 Return Value 111
9.1.9.6 Command Examples 111
9.1.9.7 Return Examples 111

9.1.10 Command get joins 109
9.1.10.1 Command usage 109
9.1.10.2 Description 109
9.1.10.3 Arguments 109
9.1.10.4 Notes 109
9.1.10.5 Return Value 109
9.1.10.6 Command Example 109
9.1.10.7 Return Examples 109

9.1.11 Command get json 110
9.1.11.1 Command usage 110
9.1.11.2 Description 110
9.1.11.3 Arguments 110
9.1.11.4 Notes 110
9.1.11.5 Return Value 110
9.1.11.6 Command Examples 110
9.1.11.7 Return Examples 110

9.1.12 Command get preferred 111
9.1.12.1 Command usage 111
9.1.12.2 Description 111
9.1.12.3 Arguments 111
9.1.12.4 Notes 111
9.1.12.5 Return Value 111
9.1.12.6 Command Example 111
9.1.12.7 Return Examples 111

9.1.13 Command get scaler 112
9.1.13.1 Command usage 112
9.1.13.2 Description 112
9.1.13.3 Arguments 112
9.1.13.4 Notes 112
9.1.13.5 Return Value 112
9.1.13.6 Command Examples 112
9.1.13.7 Return Examples 112

9.1.14 Command get security 113
9.1.14.1 Command usage 113
9.1.14.2 Description 113
9.1.14.3 Arguments 113
9.1.14.4 Notes 113
9.1.14.5 Return Value 113
9.1.14.6 Command Example 113
9.1.14.7 Return Examples 113

Table of Contents continued…

9.1.15 Command get status 114
9.1.15.1 Command usage 114
9.1.15.2 Description 114
9.1.15.3 Arguments 114
9.1.14.4 Notes 114
9.1.14.5 Return Value 114
9.1.14.6 Command Example 114
9.1.14.7 Return Examples 114

9.1.16 Command get temp 115
9.1.16.1 Command usage 115
9.1.16.2 Description 115
9.1.16.3 Arguments 115
9.1.16.4 Notes 115
9.1.16.5 Return Value 115
9.1.16.6 Command Example 115
9.1.16.7 Return Examples 115

9.1.17 Command get ver 116
9.1.17.1 Command usage 116
9.1.17.2 Description 116
9.1.17.3 Arguments 116
9.1.17.4 Notes 116
9.1.17.5 Return Value 116
9.1.17.6 Command Example 116
9.1.17.7 Return Examples 116

9.1.18 Command get video / get rt_video 117
9.1.18.1 Command usage 117
9.1.18.2 Description 117
9.1.18.3 Arguments 117
9.1.18.4 Notes 117
9.1.18.5 Return Value 117
9.1.18.2 Command Examples 117
9.1.18.7 Return Examples 117

9.1.19 Command get video_compress 118
9.1.19.1 Command usage 118
9.1.19.2 Description 118
9.1.19.3 Arguments 118
9.1.19.4 Notes 118
9.1.19.5 Return Value 118
9.1.19.6 Command Example 118
9.1.19.7 Return Examples 118

9.1.20 Command get video_mode 119
9.1.20.1 Command usage 119
9.1.20.2 Description 119
9.1.20.3 Arguments 119
9.1.20.4 Notes 119
9.1.20.5 Return Value 119
9.1.20.6 Command Example 119
9.1.20.7 Return Examples 119

9.1.21 Command get video_mute 120
9.1.21.1 Command usage 120
9.1.21.2 Description 120
9.1.21.3 Arguments 120
9.1.21.4 Notes 120
9.1.21.5 Return Value 120
9.1.21.6 Command Example 120
9.1.21.7 Return Examples 120

Table of Contents continued…

9.1.22 Command get video_source 121
9.1.22.1 Command usage 121
9.1.22.2 Description 121
9.1.22.3 Arguments 121
9.1.22.4 Notes 121
9.1.22.5 Return Value 121
9.1.22.6 Command Example 121
9.1.22.7 Return Examples 121

9.1.23 Command get video_status / get rt_video_status 122
9.1.23.1 Command usage 122
9.1.23.2 Description 122
9.1.23.3 Arguments 122
9.1.23.4 Notes 122
9.1.23.5 Return Value 122
9.1.23.6 Command Example 122
9.1.23.7 Return Examples 122

9.1.23 Command get window / get rt_window 123
9.1.24.1 Command usage 123
9.1.24.2 Description 123
9.1.24.3 Arguments 123
9.1.24.4 Notes 123
9.1.24.5 Return Value 123
9.1.24.6 Command Examples 123
9.1.24.7 Return Examples 123

Table of Contents continued…

9.2 Command get for system 124

9.2.1 Command get events 124
9.2.1.1 Command usage 124
9.2.1.2 Description 124
9.2.1.3 Arguments 124
9.2.1.4 Notes 124
9.2.1.5 Return Value 124
9.2.6 Command Examples 124
9.2.7 Return Examples 124

9.2.2 Command get matrix 125
9.2.2.1 Command usage 125
9.2.2.2 Description 125
9.2.2.3 Arguments 125
9.2.2.4 Notes 125
9.2.2.5 Return Value 125
9.2.2.6 Command Examples 125
9.2.2.7 Return Examples 125

9.2.3 Command get presenter 126
9.2.3.1 Command usage 126
9.2.3.2 Description 126
9.2.3.3 Arguments 126
9.2.3.4 Notes 126
9.2.3.5 Return Value 126
9.2.3.6 Command Example 126
9.2.3.7 Return Examples 126

9.2.4 Command get var 127
9.2.4.1 Command usage 127
9.2.4.2 Description 127
9.2.4.3 Arguments 127
9.2.4.4 Notes 127
9.2.4.5 Return Value 127
9.2.4.6 Command Examples 127
9.2.4.7 Return Examples 127

Table of Contents continued…

9.3 Command get for User Interfaces 128

9.3.1 Command get ui 128
9.3.1.1 Command usage 128
9.3.1.2 Description 128
9.3.1.3 Arguments 128
9.3.1.4 Notes 128
9.3.1.5 Return Value 128
9.3.1.6 Command Examples 128
9.3.1.7 Return Examples 128

9.3.2 Command get ui_button 129
9.3.2.1 Command usage 129
9.3.2.2 Description 129
9.3.2.3 Arguments 129
9.3.2.4 Notes 129
9.3.2.5 Return Value 129
9.3.2.6 Command Examples 129
9.3.2.7 Return Examples 129

9.3.3 Command get ui_indicator 130
9.3.3.1 Command usage 130
9.3.3.2 Description 130
9.3.3.3 Arguments 130
9.3.3.4 Notes 130
9.3.3.5 Return Value 130
9.3.3.6 Command Examples 130

9.3.4 Command get ui_slider 131
9.3.4.1 Command usage 131
9.3.4.2 Description 131
9.3.4.3 Arguments 131
9.3.4.4 Notes 131
9.3.4.5 Return Value 131

Table of Contents continued…

10 Command send 132

10.1 Command send infrared 133
10.1.1 Command usage 133
10.1.2 Description 133
10.1.3 Arguments 133
10.1.4 Notes 133
10.1.5 Return Value 133
10.1.6 Command Examples 133
10.1.7 Return Examples 134

10.2 Command send serial 135
10.2.1 Command usage 135
10.2.2 Description 135
10.2.3 Arguments 135
10.2.4 Notes 135
10.2.5 Return Value 136
10.2.6 Command Examples 136
10.2.7 Return Examples 136

10.3 Command send cec 137
10.3.1 Command usage 137
10.3.2 Description 137
10.3.3 Arguments 137
10.3.4 Notes 137
10.3.5 Return Value 137
10.3.6 Command Examples 137
10.3.7 Return Examples 137

10.4 Command send gc *licensed feature 138

10.5 Command send tcp 139
10.5.1 Command usage 139
10.5.2 Description 139
10.5.3 Arguments 139
10.5.4 Notes 139
10.5.5 Return Value 139
10.5.6 Command Examples 139
10.5.7 Return Examples 140

Table of Contents continued…

11 Commands for Multiview 141

11.1 Command multiview 141
11.1.1 Command usage 141
11.1.2 Description 141
11.1.3 Arguments 141
11.1.4 Notes 141
11.1.5 Return Value 141
11.1.6 Command Examples 141
11.1.7 Return Examples 141

11.2 Command layout 142
11.2.1 Command layout new 143
11.2.1.1 Command usage 143
11.2.1.2 Description 143
11.2.1.3 Arguments 143
11.2.1.4 Notes 143
11.2.1.5 Return Value 143
11.2.1.6 Command Examples 143
11.2.1.7 Return Examples 143

11.2.2 Command layout window 144
11.2.2.1 Command usage 144
11.2.2.2 Description 144
11.2.2.3 Arguments 144
11.2.2.4 Notes 144
11.2.2.5 Return Value 144
11.2.2.6 Command Examples 144
11.2.2.7 Return Examples 145

11.2.3 Command layout black 146
11.2.3.1 Command usage 146
11.2.3.2 Description 146
11.2.3.3 Arguments 146
11.2.3.4 Notes 146
11.2.3.5 Return Value 146
11.2.3.6 Command Examples 146
11.2.3.7 Return Examples 146

11.3.4 Command layout delete 147
11.3.4.1 Command usage 147
11.3.4.2 Description 147
11.3.4.3 Arguments 147
11.3.4.4 Notes 147
11.3.4.5 Return Value 147
11.3.4.6 Command Examples 147
11.3.4.7 Return Examples 147

11.3.5 Command layout surface 148
11.3.5.1 Command usage 148
11.3.5.2 Description 148
11.3.5.3 Arguments 148
11.3.5.4 Notes 148
11.3.5.5 Return Value 148
11.3.5.6 Command Examples 148
11.3.5.7 Return Examples 148

Table of Contents continued…

12 Message notify 149

12.1 Message notify serial 149
12.1.1 Message received 149
12.1.2 Description 149
12.1.3 Arguments 149
12.1.4 Notes 149
12.1.5 Received Value 149
12.1.6 Examples 149

12.2 Message notify network 150
12.2.1 Message received 150
12.2.2 Description 150
12.2.3 Arguments 150
12.2.4 Notes 150
12.2.5 Received Value 150
12.2.6 Examples 150

12.3 Message notify display 151
12.3.1 Message received 151
12.3.2 Description 151
12.3.3 Arguments 151
12.3.4 Notes 151
12.3.5 Received Value 151
12.3.6 Examples 151

12.4 Message notify source 152
12.4.1 Message received 152
12.4.2 Description 152
12.4.3 Arguments 152
12.4.4 Notes 152
12.4.5 Received Value 152
12.4.6 Examples 152

12.5 Message notify stream 153
12.5.1 Message received 153
12.5.2 Description 153
12.5.3 Arguments 153
12.5.4 Notes 153
12.5.5 Received Value 153
12.5.6 Examples 153

Table of Contents continued…

13 Command preset 154

13.1 Command preset add 154
13.1.1 Command usage 154
13.1.2 Description 154
13.1.3 Arguments 154
13.1.4 Notes 154
13.1.5 Return Value 154
13.1.6 Command Examples 154
13.1.7 Return Examples 154

13.2 Command preset load 155
13.2.1 Command usage 155
13.2.2 Description 155
13.2.3 Arguments 155
13.2.4 Notes 155
13.2.5 Return Value 155
13.2.6 Command Examples 155
13.2.7 Return Examples 155

13.3 Command preset delete 156
13.3.1 Command usage 156
13.3.2 Description 156
13.3.3 Arguments 156
13.3.4 Notes 156
13.3.5 Return Value 156
13.3.6 Command Examples 156
13.3.7 Return Examples 156

13.4 Command preset delay 157
13.4.1 Command usage 157
13.4.2 Description 157
13.4.3 Arguments 157
13.4.4 Notes 157
13.4.5 Return Value 157
13.4.6 Command Examples 157
13.4.7 Return Example 157

13.5 Command preset wait 158
13.5.1 Command usage 158
13.5.2 Description 158
13.5.3 Arguments 158
13.5.4 Notes 158
13.5.5 Return Value 158
13.5.6 Command Example 158
13.5.7 Return Example 158

13.6 Command preset dynamic 159
13.6.1 Command usage 159
13.6.2 Description 159
13.6.3 Arguments 159
13.6.4 Notes 159
13.6.5 Return Value 159
13.6.6 Command Example 159
13.6.7 Return Example 159

13.7 Command preset inactiveBypass 160
13.7.1 Command usage 160
13.7.2 Description 160
13.7.3 Arguments 160
13.7.4 Notes 160
13.7.5 Return Value 160
13.7.6 Command Example 160
13.7.7 Return Example 160

Table of Contents continued…

Appendix A - How to Multiview 161

Appendix B - How to Video wall 167

Appendix C - How to Video Wall with Multiview 185

Appendix D - How to HTTP request 188

Appendix E - Preset logic 191

1 Introduction

This document describes everything that a developer needs to be aware of to use the COMMANDER command guide and develop client control applications for Canare's SDVoE appliance.

1.1 License Requirements

The SDVoE Controller must have a valid license key entered before use or trying to connect to the TCP control port 6980.
If no valid license is active the SDVoE Controller will return 'Invalid License' and terminate the TCP connection. Please contact our sales for licensing information..

1.2 Telnet Connection

The control PC connect to the SDVoE Controller and issue commands using ASCII strings terminated with a carriage return. This allows any Telnet client to be used with the system.

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

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

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

1.3 HTTP requests

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

GET = http://<*controllerURL*>/api/command/<*COMMANDER_API_COMMAND*>/<*KEY*>
POST = http://<controllerURL>/api/command{"cmd":"<*COMMANDER_API_COMMAND*>","key":"<*KEY*>"}
Refer Appendix D - How to HTTP request

2 Command Overview

Commands are in a simple ASCII text format. For each command, the SDVoE 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 **error [Unknown]**<cr>
- Missing security key will return **error [security key missing]**<cr>
- Invalid security key will return **error [security key invalid]**<cr>
- Invalid security key will return **error [security key invalid]**<cr>

ToC

3 System Commands

3.1 Command default

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

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

3.1.1 Command usage

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

3.1.2 Description

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

3.1.3 Arguments

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

3.1.4 Notes

3.1.5 Return Value

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

3.1.6 Command Examples

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

3.1.7 Return Examples

default success<cr>

default error [incomplete]<cr>
default error [value]<cr>

ToC

3.2 Command reboot

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

3.2.1 Command usage

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

3.2.2 Description

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

3.2.3 Arguments

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

3.2.4 Notes

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

3.2.5 Return Value

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

3.2.6 Command Examples

reboot *Encoder1*<cr>
reboot all<cr>
reboot all_rx<cr>
reboot all_tx<cr>
reboot *MyGroup*<cr>
reboot key:*abc123 Decoder1*<cr>

3.2.7 Return Examples

reboot success<cr>

reboot error [incomplete]<cr>
reboot error [device '*Encoder1'* not found]<cr>
reboot error [device '*Decoder1'* disconnected]<cr>
reboot error [group devices not found]<cr>

ToC

4 Command join

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

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

In **fast** mode if the aspect ratio of the video source and the output resolution are not the same the default behaviour is to add black bars on the sides (pillarbox) or at the top and bottom (letterbox) of the image to preserve the aspect ratio. An alternate behaviour can be specified by adding the keyword **crop** or **stretch** to the command line.

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

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

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

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

join audio_a command provides independent routing of the analog audio.

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

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

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

join usb command provides independent routing of USB.

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

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

Commands missing **mode** return:

join error [incomplete]<cr>

Commands with invalid **mode** return:

join error [invalid mode]<cr>

ToC

4.1 Command join fast

4.1.1 Command usage

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

4.1.2 Description

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

4.1.3 Arguments

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

4.1.4 Notes

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

4.1.5 Return Value

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

ToC

4.1.6 Command Examples

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

4.1.7 Return Examples

join fast success<cr>

join fast error [incomplete]<cr>
join fast error [unsupported monitor]<cr>
join fast error [monitor not detected]<cr>
join fast error [invalid parameter]<cr>
join fast error [invalid size]<cr>
join fast error [video format is not supported on this device]<cr>
join fast error [join not permitted]<cr>
join fast error [encoder '*Encoder1*' not found]<cr>
join fast error [decoder '*Decoder1*' not found]<cr>
join fast error [encoder '*Encoder1*' disconnected]<cr>
join fast error [decoder '*Decoder1*' disconnected]<cr>
join fast error [group devices not found]<cr>

ToC

4.2 Command join sync

4.2.1 Command usage

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

4.2.2 Description

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

4.2.3 Arguments

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

4.2.4 Notes

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

4.2.5 Return Value

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

4.2.6 Command Examples

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

4.2.7 Return Examples

join sync success<cr>

join sync error [incomplete]<cr>
join sync error [invalid parameter]<cr>
join sync error [join not permitted]<cr>
join sync error [encoder '*Encoder1*' not found]<cr>
join sync error [decoder '*Decoder1*' not found]<cr>
join sync error [encoder '*Encoder1*' disconnected]<cr>
join sync error [decoder '*Decoder1*' disconnected]<cr>
join sync error [group devices not found]<cr>

ToC

4.3 Command join sync_scale

4.3.1 Command usage

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

4.3.2 Description

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

4.3.3 Arguments

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

4.3.4 Notes

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

ToC

4.3.5 Return Value

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

4.3.6 Command Examples

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

4.3.7 Return Examples

join sync_scale success<cr>

join sync_scale error [incomplete]<cr>
join sync_scale error [unsupported monitor]<cr>
join sync_scale error [monitor not detected]<cr>
join sync_scale error [invalid parameter]<cr>
join sync_scale error [invalid size]<cr>
join sync_scale error [video format is not supported on this device]<cr>
join sync_scale error [join not permitted]<cr>
join sync_scale error [encoder '*Encoder1*' not found]<cr>
join sync_scale error [decoder '*Decoder1*' not found]<cr>
join sync_scale error [encoder '*Encoder1*' disconnected]<cr>
join sync_scale error [decoder '*Decoder1*' disconnected]<cr>
join sync_scale error [group devices not found]<cr>

4.4 Command join adv

4.4.1 Command usage

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

4.4.2 Description

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

4.4.3 Arguments

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

4.4.4 Notes

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

4.4.5 Return Value

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

ToC

4.4.6 Command Examples

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

4.4.7 Return Examples

join adv success<cr>

join adv error [incomplete]<cr>
join adv error [invalid parameter]<cr>
join adv error [invalid size]<cr>
join adv error [video format is not supported on this device]<cr>
join adv error [join not permitted]<cr>
join adv error [encoder '*Encoder1*' not found]<cr>
join adv error [decoder '*Decoder1*' not found]<cr>
join adv error [encoder '*Encoder1*' disconnected]<cr>
join adv error [decoder '*Decoder1*' disconnected]<cr>
join adv error [monitor not detected]<cr>
join adv error [group devices not found]<cr>

4.5 Command join audio_a

4.5.1 Command usage

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

4.5.2 Description

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

4.5.3 Arguments

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

4.5.4 Notes

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

4.5.5 Return Value

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

4.5.6 Command Examples

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

4.5.7 Return Examples

join audio_a success<cr>

join audio_a error [not supported]<cr>
join audio_a error [incomplete]<cr>
join audio_a error [invalid parameter]<cr>
join audio_a error [join not permitted]<cr>
join audio_a error [encoder '*Encoder1*' not found]<cr>
join audio_a error [decoder '*Decoder1*' not found]<cr>
join audio_a error [encoder '*Encoder1*' disconnected]<cr>
join audio_a error [decoder '*Decoder1*' disconnected]<cr>
join audio_a error [group devices not found]<cr>

ToC

4.6 Command join audio_d

4.6.1 Command usage

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

4.6.2 Description

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

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

4.6.3 Arguments

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

4.6.4 Notes

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

4.6.5 Return Value

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

4.6.6 Examples

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

4.6.7 Return Examples

join audio_d success<cr>

join audio_d error [incomplete]<cr>
join audio_d error [invalid parameter]<cr>
join audio_d error [join not permitted]<cr>
join audio_d error [encoder '*Encoder1*' not found]<cr>
join audio_d error [decoder '*Decoder1*' not found]<cr>
join audio_d error [encoder '*Encoder1*' disconnected]<cr>
join audio_d error [decoder '*Decoder1*' disconnected]<cr>
join audio_d error [group devices not found]<cr>

ToC

4.7 Command join ir

4.7.1 Command usage

join ir **[**key:<*security_key*>**]** <*source*> <*destination*> / all / all_rx / all_tx<cr>

4.7.2 Description

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

4.7.3 Arguments

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

4.7.4 Notes

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

4.7.5 Return Value

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

4.7.6 Command Examples

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

4.7.7 Return Examples

join ir success<cr>

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

ToC

4.8 Command join serial

4.8.1 Command usage

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

4.8.2 Description

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

4.8.3 Arguments

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

4.8.4 Notes

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

4.8.5 Return Value

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

4.8.6 Command Examples

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

4.8.7 Return Examples

join serial success<cr>

join serial error [not supported]<cr>
join serial error [function not allowed with mixed devices]<cr>
join serial error [incomplete]<cr>
join serial error [invalid parameter]<cr>
join serial error [join not permitted]<cr>
join serial error [source '*Encoder1*' not found]<cr>
join serial error [destination '*Decoder1*' not found]<cr>
join serial error [source '*Encoder1*' disconnected]<cr>
join serial error [destination '*Decoder1*' disconnected]<cr>

ToC

4.9 Command join usb

4.9.1 Command usage

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

4.9.2 Description

The command **join usb** is used for routing Icron USB and extenders.

4.9.3 Arguments

lex_device_name	Device name of Encoder / Host
rex_device_name	Device name of Decoder / Client

4.9.4 Notes

- Embedded Encoder / Decoder Icron USB LEX can only pair with a single USB REX device.

4.9.5 Return Value

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

4.9.6 Command Examples

join usb *Encoder1 Decoder1*<cr>
join usb *USBEXT1 USBEXT2*<cr>
join usb key:*abc123 Encoder1 Decoder1*<cr>

4.9.7 Return Examples

join usb success<cr>

join usb error [not supported]<cr>
join usb error [incomplete]<cr>
join usb error [join failed]<cr>
join usb error [join not permitted]<cr>
join usb error [lex '*Encoder1*' not found]<cr>
join usb error [rex '*Decoder1*' not found]<cr>
join usb error [lex '*Encoder1*' disconnected]<cr>
join usb error [rex '*Decoder1*' disconnected]<cr>

4.10 Command join usb_hid

4.10.1 Command usage

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

4.10.2 Description

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

4.10.3 Arguments

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

4.10.4 Notes

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

4.10.5 Return Value

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

4.10.6 Command Examples

join usb_hid *Encoder1 Decoder1*<cr>
join usb_hid key:*abc123 Encoder1 Decoder1*<cr>

4.10.7 Return Examples

join usb_hid success<cr>

join usb_hid error [not supported]<cr>
join usb_hid error [incomplete]<cr>
join usb_hid error [join not permitted]<cr>
join usb_hid error [lex '*Encoder1*' not found]<cr>
join usb_hid error [rex '*Decoder1*' not found]<cr>
join usb_hid error [lex '*Encoder1*' disconnected]<cr>
join usb_hid error [rex '*Decoder1*' disconnected]<cr>

ToC

4.11 Command join multi

4.11.1 Command usage

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

4.11.2 Description

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

4.11.3 Arguments

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

4.11.4 Notes

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

4.11.5 Return Value

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

4.11.6 Command Examples

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

ToC

4.11.7 Return Examples

join multi success<cr>

join multi error [not supported]<cr>
join multi error [incomplete]<cr>
join multi error [layout '*MyLayout*' not found]<cr>
join multi error [invalid subscription '*32*']<cr>
join multi error [subscription '*3*' not found]<cr>
join multi error [only hdmi subscription index 0 can be joined in display modes other than multiview]<cr>
join multi error [invalid parameter]<cr>
join multi error [join not permitted]<cr>
join multi error [encoder '*Encoder1*' scaler not set]<cr>
join multi error [encoder '*Encoder1*' not found]<cr>
join multi error [decoder '*Decoder1*' not found]<cr>
join multi error [encoder '*Encoder1*' disconnected]<cr>
join multi error [decoder '*Decoder1*' disconnected]<cr>

4.12 Command join wall

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

4.12.1 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*>**]**
[<keep>**]** **[**<*wall_mode*>**]**<cr>

4.12.2 Description

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

4.12.3 Arguments

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

ToC

4.12.4 Notes

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

4.12.5 Return Value

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

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

join wall *Encoder1 Decoder1 2x2 1* size *1920 1080 60* bezel *615 595 365 335* keep<cr>
join wall *Encoder1 Decoder1 2x2 2* size *1920 1080 60* bezel *615 595 365 335* keep<cr>
join wall *Encoder1 Decoder1 2x2 3* size *1920 1080 60* bezel *615 595 365 335* keep<cr>
join wall *Encoder1 Decoder1 2x2 4* size *1920 1080 60* bezel *615 595 365 335* keep<cr>

join wall key:*abc123 Encoder1 Decoder1 2x2 1* size *1920 1080 60* bezel *615 595 365 335* keep<cr>

4.12.7 Return Examples

join wall success<cr>

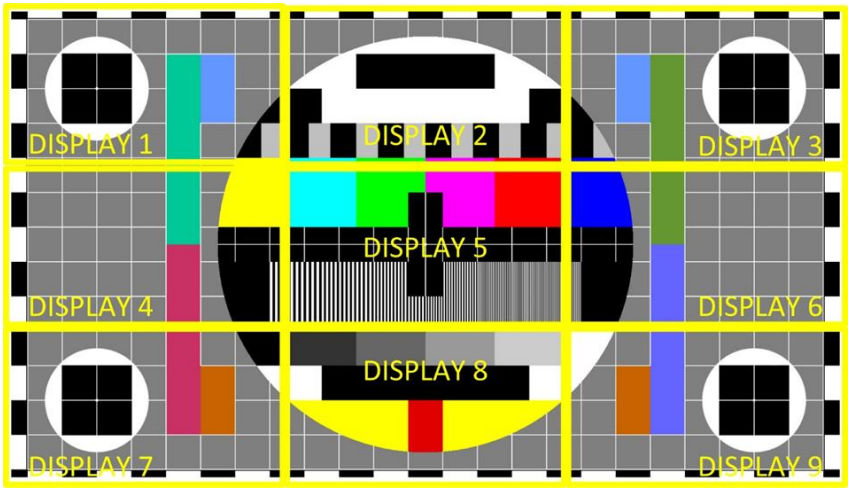
join wall error [incomplete]<cr>
join wall error [invalid parameter]<cr>
join wall error [join not permitted]<cr>
join wall error [encoder '*Encoder1*' not found]<cr>
join wall error [decoder '*Decoder1*' not found]<cr>
join wall error [encoder '*Encoder1*' disconnected]<cr>
join wall error [decoder '*Decoder1*' disconnected]<cr>

wall_type (columns x rows)

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

display_position

The position of the display is found by counting from left to right and top to bottom. Below is an example of a 3x3 video wall and the display number.



ToC

4.13 Command join walladv

4.13.1 Command usage

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

4.13.2 Description

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

4.13.3 Arguments

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

4.13.4 Notes

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

  join walladv Encoder1 Decoder1 1920 1080 0 0 944 516 0 0 1920 1080 60 // **2x2 1 V 615 365 10 20 10 10**
  join walladv Encoder1 Decoder2 1920 1080 976 0 944 516 0 0 1920 1080 60 // **2x2 2 V 615 365 10 20 10 10**
  join walladv Encoder1 Decoder3 1920 1080 0 564 944 516 0 0 1920 1080 60 // **2x2 3 V 615 365 10 20 10 10**
  join walladv Encoder1 Decoder4 1920 1080 976 564 944 516 0 0 1920 1080 60 // **2x2 4 V 615 365 10 20 10 10**

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

4.13.4 Notes continued…

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

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

- The **viewport_horiz** and **viewport_vert** arguments specify respectively the horizontal and vertical position of the viewport's top-left corner on the screen, in pixels.
- The **viewport_width** and **viewport_height** arguments specify respectively the width and height of the viewport, in pixels. This is the area of the displayed video.
- For full screen, **viewport_horiz** and **viewport_vert** would be set to 0, and **viewport_width** along with **viewport_height** would equal **width** and **height** respectively.
- The **width, keep_width, viewport_ horiz** and **viewport_width** arguments must be even (i.e. a multiple of two).
- The viewport must be completely contained within the screen size.

4.13.5 Return Value

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

4.13.6 Command Examples

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

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

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

4.13.7 Return Examples

join walladv success<cr>

join walladv error [incomplete]<cr>
join walladv error [invalid parameter]<cr>
join walladv error [join not permitted]<cr>
join walladv error [encoder '*Encoder1*' not found]<cr>
join walladv error [decoder '*Decoder1*' not found]<cr>
join walladv error [encoder '*Encoder1*' disconnected]<cr>
join walladv error [decoder '*Decoder1*' disconnected]<cr>

ToC

5 Command leave

The **leave** command is 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>

5.1 Command leave video

5.1.1 Command usage

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

5.1.2 Description

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

5.1.3 Arguments

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

5.1.4 Notes

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

5.1.5 Return Value

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

5.1.6 Command Examples

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

5.1.7 Return Examples

leave video success<cr>

leave video error [incomplete]<cr>
leave video error [decoder '*Decoder1*' not found]<cr>
leave video error [decoder '*Decoder1*' disconnected]<cr>
leave video error [group devices not found]<cr>

ToC

5.2 Command leave sub

5.2.1 Command usage

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

5.2.2 Description

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

5.2.3 Arguments

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

5.2.4 Notes

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

5.2.5 Return Value

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

5.2.6 Command Examples

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

5.2.7 Return Examples

leave sub success<cr>

leave sub error [not supported]<cr>
leave sub error [incomplete]<cr>
leave sub error [invalid subscription]<cr>
leave sub error [decoder '*Decoder1*' not found]<cr>
leave sub error [decoder '*Decoder1*' disconnected]<cr>

ToC

5.3 Command leave av

5.3.1 Command usage

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

5.3.2 Description

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

5.3.3 Arguments

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

5.3.4 Notes

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

5.3.5 Return Value

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

5.3.6 Command Examples

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

5.3.7 Return Examples

leave av success<cr>

leave av error [incomplete]<cr>
leave av error [decoder '*Decoder1*' not found]<cr>
leave av error [decoder '*Decoder1*' disconnected]<cr>
leave av error [group devices not found]<cr>

ToC

5.4 Command leave audio_a

5.4.1 Command usage

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

5.4.2 Description

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

5.4.3 Arguments

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

5.4.4 Notes

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

5.4.5 Return Value

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

5.4.6 Command Examples

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

5.4.7 Return Examples

leave audio_a success<cr>

leave audio_a error [not supported]<cr>
leave audio_a error [incomplete]<cr>
leave audio_a error [decoder '*Decoder1*' not found]<cr>
leave audio_a error [decoder '*Decoder1*' disconnected]<cr>
leave audio_a error [group devices not found]<cr>

ToC

5.5 Command leave audio_d

5.5.1 Command usage

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

5.5.2 Description

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

5.5.3 Arguments

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

5.5.4 Notes

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

5.5.5 Return Value

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

5.5.6 Command Examples

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

5.5.7 Return Examples

leave audio_d success<cr>

leave audio_d error [incomplete]<cr>
leave audio_d error [decoder '*Decoder1*' not found]<cr>
leave audio_d error [decoder '*Decoder1*' disconnected]<cr>
leave audio_d error [group devices not found]<cr>

ToC

5.6 Command leave all

5.6.1 Command usage

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

5.6.2 Description

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

5.6.3 Arguments

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

5.6.4 Notes

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

5.6.5 Return Value

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

5.6.6 Command Example

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

5.6.7 Return Examples

leave all success<cr>

leave all error [incomplete]<cr>
leave all error [decoder '*Decoder1*' not found]<cr>
leave all error [decoder '*Decoder1*' disconnected]<cr>
leave all error [group devices not found]<cr>

ToC

5.7 Command leave usb

5.7.1 Command usage

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

5.7.2 Description

The command **leave usb** is used to remove device USB pairing.

5.7.3 Arguments

rex_device_name	Device name of Decoder / Client

5.7.4 Notes

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

5.7.5 Return Value

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

5.7.6 Example

leave usb *Decoder1*<cr>
leave usb *EXTUSB1*<cr>
leave usb key:*abc123 Decoder1*<cr>

5.7.7 Return Examples

leave usb success<cr>

leave usb error [not supported]<cr>
leave usb error [incomplete]<cr>
leave usb error [leave failed]<cr>
leave usb error [rex '*Decoder1*' not found]<cr>
leave usb error [rex '*Decoder1*' disconnected]<cr>

ToC

5.8 Command leave usb_hid

5.8.1 Command usage

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

5.8.2 Description

The command **leave usb_hid** is used to remove device USB HID pairing

5.8.3 Arguments

rex_device_name	Device name of Decoder / Client

5.8.4 Notes

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

5.8.5 Return Value

command	<space>	**mode**	<space>	**status**	**terminator**
leave	<space>	usb_hid	<space>	*success / error*	<cr>

5.8.6 Example

leave usb_hid *Decoder1*<cr>
leave usb_hid key:*abc123 Decoder1*<cr>

5.8.7 Return Examples

leave usb_hid success<cr>

leave usb_hid error [not supported]<cr>
leave usb_hid error [incomplete]<cr>
leave usb_hid error [rex '*Decoder1*' not found]<cr>
leave usb_hid error [rex '*Decoder1*' disconnected]<cr>

ToC

6 Command stop

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

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

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

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

Commands missing **mode** return:

stop error [incomplete]<cr>

Commands with invalid **mode** return:

stop error [invalid mode]<cr>

ToC

6.1 Command stop video

6.1.1 Command usage

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

6.1.2 Description

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

6.1.3 Arguments

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

6.1.4 Notes

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

6.1.5 Return Value

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

6.1.6 Command Examples

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

6.1.7 Return Examples

stop video success<cr>

stop video error [incomplete]<cr>
stop video error [invalid parameter]<cr>
stop video error [encoder '*Encoder1*' not found]<cr>
stop video error [encoder '*Encoder1*' disconnected]<cr>
stop video error [group devices not found]<cr>

ToC

6.2 Command stop sub

6.2.1 Command usage

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

6.2.2 Description

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

6.2.3 Arguments

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

6.2.4 Notes

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

6.2.5 Return Value

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

6.2.6 Command Examples

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

6.2.7 Return Examples

stop sub success<cr>

stop sub error [not supported]<cr>
stop sub error [incomplete]<cr>
stop sub error [invalid parameter]<cr>
stop sub error [encoder '*Encoder1*' not found]<cr>
stop sub error [encoder '*Encoder1*' disconnected]<cr>

ToC

6.3 Command stop av

6.3.1 Command usage

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

6.3.2 Description

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

6.3.3 Arguments

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

6.3.4 Notes

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

6.3.5 Return Value

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

6.3.6 Command Examples

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

6.3.7 Return Examples

stop av success<cr>

stop av error [incomplete]<cr>
stop av error [invalid parameter]<cr>
stop av error [encoder '*Encoder1*' not found]<cr>
stop av error [encoder '*Encoder1*' disconnected]<cr>
stop av error [group devices not found]<cr>

ToC

6.4 Command stop audio_a

6.4.1 Command usage

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

6.4.2 Description

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

6.4.3 Arguments

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

6.4.4 Notes

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

6.4.5 Return Value

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

6.4.6 Command Examples

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

6.4.7 Return Examples

stop audio_a success<cr>

stop audio_a error [not supported]<cr>
stop audio_a error [incomplete]<cr>
stop audio_a error [invalid parameter]<cr>
stop audio_a error [encoder '*Encoder1*' not found]<cr>
stop audio_a error [encoder '*Encoder1*' disconnected]<cr>
stop audio_a error [group devices not found]<cr>

ToC

6.5 Command stop audio_d

6.5.1 Command usage

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

6.5.2 Description

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

6.5.3 Arguments

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

6.5.4 Notes

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

6.5.5 Return Value

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

6.5.6 Command Examples

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

6.5.7 Return Examples

stop audio_d success<cr>

stop audio_d error [incomplete]<cr>
stop audio_d error [invalid parameter]<cr>
stop audio_d error [encoder '*Encoder1*' not found]<cr>
stop audio_d error [encoder '*Encoder1*' disconnected]<cr>
stop audio_d error [group devices not found]<cr>

ToC

6.6 Command stop ir

6.6.1 Command usage

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

6.6.2 Description

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

6.6.3 Arguments

device_name	Device name of either Encoder or Decoder

6.6.4 Notes

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

6.6.5 Return Value

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

6.6.6 Example

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

6.6.7 Return Examples

stop ir success<cr>

stop ir error [not supported]<cr>
stop ir error [incomplete]<cr>
stop ir error [device '*Decoder1*' not found]<cr>
stop ir error [device 'Encoder1' disconnected]

ToC

6.7 Command stop serial

6.7.1 Command usage

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

6.7.2 Description

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

6.7.3 Arguments

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

6.7.4 Notes

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

6.7.5 Return Value

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

6.7.6 Examples

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

6.7.7 Return Examples

stop serial success<cr>

stop serial error [not supported]<cr>
stop serial error [incomplete]<cr>
stop serial error [invalid parameter]<cr>
stop serial error [device '*Encoder1*' not found]<cr>
stop serial error [device '*Decoder1*' disconnected]<cr>

6.8 Command stop all

6.8.1 Command usage

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

6.8.2 Description

The command **stop all** is used to stop all device streams from being sent to any device.

6.8.3 Arguments

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

6.8.4 Notes

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

6.8.5 Return Value

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

6.8.6 Example

stop all *Encoder1*<cr>
stop all *Encoder1* free<cr>
stop all *Encoder1* free_all<cr>
stop all *MyGroup*<cr>
stop all *MyGroup* free<cr>
stop all all<cr>
stop all key:*abc123 Encoder1*<cr>

6.8.7 Return Examples

stop all success<cr>

stop all error [incomplete]<cr>
stop all error [invalid parameter]<cr>
stop all error [encoder '*Encoder1*' not found]<cr>
stop all error [encoder '*Encoder1*' disconnected]<cr>

ToC

7 Command start

The **start** command is used to start an Encoder's stream. A join will automatically start a stream.

Commands missing **mode** return:

start error [incomplete]<cr>

Commands with invalid **mode** return:

start error [invalid mode]<cr>

ToC

7.1 Command start video

7.1.1 Command usage

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

7.1.2 Description

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

7.1.3 Arguments

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

7.1.4 Notes

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

7.1.5 Return Value

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

7.1.6 Command Examples

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

7.1.7 Return Examples

start video success<cr>

start video error [incomplete]<cr>
start video error [out of range]<cr>
start video error [invalid multicast_addr]<cr>
start video error [multicast IP address is in use]<cr>
start video error [encoder '*Encoder1*' not found]<cr>
start video error [encoder '*Encoder1*' disconnected]<cr>
start video error [group devices not found]<cr>

ToC

7.2 Command start sub

7.2.1 Command usage

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

7.2.2 Description

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

7.2.3 Arguments

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

7.2.4 Notes

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

7.2.5 Return Value

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

7.2.6 Command Examples

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

7.2.7 Return Examples

start sub success<cr>

start sub error [not supported]<cr>
start sub error [incomplete]<cr>
start sub error [out of range]<cr>
start sub error [invalid multicast_addr]<cr>
start sub error [multicast IP address is in use]<cr>
start sub error [encoder '*Encoder1*' scaler not set]<cr>
start sub error [encoder '*Encoder1*' not found]<cr>
start sub error [encoder '*Encoder1*' disconnected]<cr>

ToC

7.3 Command start av

7.3.1 Command usage

start av **[**key:<*security_key*>**]** <*encoder_device_name*> / <*group_name*>
[<*audio_multicast_addr*> <*video*_multicast_addr>**]**<cr>

7.3.2 Description

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

7.3.3 Arguments

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

7.3.4 Notes

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

7.3.5 Return Value

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

7.3.6 Command Examples

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

7.3.7 Return Examples

start av success<cr>

start av error [incomplete]<cr>
start av error [out of range]<cr>
start av error [invalid audio_multicast_addr]<cr>
start av error [invalid video_multicast_addr]<cr>
start av error [multicast IP address is in use]<cr>
start av error [encoder '*Encoder1*' not found]<cr>
start av error [encoder '*Encoder1*' disconnected]<cr>
start av error [group devices not found]<cr>

ToC

7.4 Command start audio_a

7.4.1 Command usage

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

7.4.2 Description

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

7.4.3 Arguments

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

7.4.4 Notes

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

7.4.5 Return Value

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

7.4.6 Command Examples

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

7.4.7 Return Examples

start audio_a success<cr>

start audio_a error [not supported]<cr>
start audio_a error [incomplete]<cr>
start audio_a error [out of range]<cr>
start audio_a error [invalid multicast_addr]<cr>
start audio_a error [multicast IP address is in use]<cr>
start audio_a error [encoder '*Encoder1*' not found]<cr>
start audio_a error [encoder '*Encoder1*' disconnected]<cr>
start audio_a error [group devices not found]<cr>

ToC

7.5 Command start audio_d

7.5.1 Command usage

Start audio_d **[**key:<*security_key*>**]** <*encoder_device_name*> / <*group_name*> **[**<*multicast_addr*>**]**<cr>

7.5.2 Description

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

7.5.3 Arguments

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

7.5.4 Notes

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

7.5.5 Return Value

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

7.5.6 Command Examples

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

7.5.7 Return Examples

start audio_d success<cr>

start audio_d error [incomplete]<cr>
start audio_d error [invalid multicast_addr]<cr>
start audio_d error [out of range]<cr>
start audio_d error [multicast IP address is in use]<cr>
start audio_d error [encoder '*Encoder1*' not found]<cr>
start audio_d error [encoder '*Encoder1*' disconnected]<cr>
start audio_d error [group devices not found]<cr>

ToC

8 Command set

The **set** commands are used to change the working parameters of various settings.

Commands missing **mode** return:

set error [incomplete]<cr>

Commands with invalid **mode** return:

set error [invalid mode]<cr>

8.1 Command set for devices

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

8.1.1 Command set audio_io

8.1.1.1 Command usage

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

8.1.1.2 Description

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

8.1.1.3 Arguments

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

8.1.1.4 Notes

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

8.1.1.5 Return Value

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

8.1.1.6 Command Examples

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

8.1.1.7 Return Examples

set audio_io success<cr>

set audio_io error [not supported]<cr>
set audio_io error [incomplete]<cr>
set audio_io error [invalid value '*value*']<cr>
set audio_io error [encoder '*Encoder1*' not found]<cr>
set audio_io error [encoder '*Encoder1*' disconnected]<cr>

ToC

8.1.2 Command set audio_out

8.1.2.1 Command usage

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

8.1.2.2 Description

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

8.1.2.3 Arguments

decoder_device_name	Device name of the Decoder
value	'hdmi_2' | 'analog'

8.1.2.4 Notes

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

8.1.2.5 Return Value

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

8.1.2.6 Command Examples

set audio_out *Decoder1* hdmi_2<cr>
set audio_out *Decoder1* analog<cr>
set audio_out key:*abc123 Decoder1* in<cr>

8.1.2.7 Return Examples

set audio_out success<cr>

set audio_out error [not supported]<cr>
set audio_out error [incomplete]<cr>
set audio_out error [invalid value '*<value>*']<cr>
set audio_out error [decoder '*Decoder1*' not found]<cr>
set audio_out error [decoder '*Decoder1*' disconnected]<cr>

ToC

8.1.3 Command set audio_source

8.1.3.1 Command usage

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

8.1.3.2 Description

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

8.1.3.3 Arguments

decoder_device_name	Device name of the Decoder
value	'analog' = ANALOG (Stereo Audio)
	'hdmi_sync' = HDMI (Sync Mode)
	'hdmi_2' = HDMI (Downmix Audio)
	'hdmi' = HDMI (Multichannel Audio)

8.1.3.4 Notes

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

8.1.3.5 Return Value

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

8.1.3.6 Command Examples

set audio_source *Decoder1* hdmi<cr>
set audio_source *Decoder1* hdmi_2<cr>
set audio_source *Decoder1* hdmi_sync<cr>
set audio_source *Decoder1* analog<cr>
set audio_source key:*abc123 Decoder1* hdmi<cr>

8.1.3.7 Return Examples

set audio_source success<cr>

set audio_source error [not supported]<cr>
set audio_source error [incomplete]<cr>
set audio_source error [invalid value '*<value>*']<cr>
set audio_source error [decoder '*Decoder1*' not found]<cr>
set audio_source error [decoder '*Decoder1*' disconnected]<cr>

8.1.4 Command set edid

8.1.4.1 Command usage

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

8.1.4.2 Description

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

8.1.4.3 Arguments

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

8.1.4.4 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.

8.1.4.5 Return Value

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

8.1.4.6 Command Example

Set edid *Encoder1 00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818 0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000000fd00384c1 e5315000a2020202020200000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131411f230907078301000065 030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e206e285500dc0c 1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000000000000000000000000000000000ac*<cr>

set edid *MyGroup 00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818 0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000000fd00384c1 e5315000a2020202020200000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131411f230907078301000065 030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e206e285500dc0c 1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000000000000000000000000000000000ac* <cr>

set edid key:*abc123* all_tx *00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f818 0818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731370000000fd00384c1 e5315000a2020202020200000000fc005068696c69707320323231450a01b602031ef04b1005010203040612131411f230907078301000065 030c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e206e285500dc0c 1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000000000000000000000000000000000ac* <cr>

ToC

8.1.4.7 Return Examples

set edid success<cr>

set edid error [incomplete]<cr>
set edid error [invalid edid_data]<cr>
set edid error [encoder '*Encoder1*' not found]<cr>
set edid error [encoder '*Encoder1*' disconnected]<cr>
set edid error [group devices not found]<cr>

ToC

8.1.5 Command set frame_converter

8.1.5.1 Command usage

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

8.1.5.2 Description

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

8.1.5.3 Arguments

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

8.1.5.4 Notes

- Only PROGRESSIVE video signals are supported. INTERLACED video signals cannot have their frame rates reduced. The command will return an invalid format error with INTERLACED signals when setting the frame_converter to true.
  Use the "get <*encoder_device_name*> video sm" command to confirm a PROGRESSIVE signal.

8.1.5.5 Return Value

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

8.1.5.6 Command Examples

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

8.1.5.7 Return Examples

set frame_converter success<cr>

set frame_converter error [not supported]<cr>
set frame_converter error [incomplete]<cr>
set frame_converter error [invalid state '<*state*>']<cr>
set frame_converter error [invalid stream '<*stream*>']<cr>
set frame_converter error [encoder '*Encoder1*' not found]<cr>
set frame_converter error [encoder '*Encoder1*' disconnected]<cr>

ToC

8.1.6 Command set scaler

8.1.6.1 Command usage

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

8.1.6.2 Description

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

8.1.6.3 Arguments

encoder_device_name	Device name of the Encoder
Width	Horizontal size of the scaled video in pixels
Height	Vertical size of the scaled video in pixels

8.1.6.4 Notes

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

8.1.6.5 Return Value

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

8.1.6.6 Command Examples

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

8.1.6.7 Return Examples

set scaler success<cr>

set scaler error [not supported]<cr>
set scaler error [incomplete]<cr>
set scaler error [invalid width '<*width*>']<cr>
set scaler error [invalid height '<*height*>']<cr>
set scaler error [encoder '*Encoder1*' not found]<cr>
set scaler error [encoder '*Encoder1*' disconnected]<cr>

8.1.7 Command set security

8.1.7.1 Command usage

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

8.1.7.2 Description

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

8.1.7.3 Arguments

device_name	Name of the Encoder, Decoder or *Group*
key	User defined 8 character key or 'default'

8.1.7.4 Notes

- To enable encryption the key must be set with an eight (8) character ASCII string.
- To disable encryption the key must be set with the word 'default'.

8.1.7.5 Return Value

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

8.1.7.6 Command Examples

set security *Encoder1 12345678*<cr>
set security *Decoder1 12345678*<cr>
set security *MyGroup 12345678*<cr>
set security *MyGroup* default<cr>
set security key:*abc123 Encoder1 12345678*<cr>

8.1.7.7 Return Examples

set security success<cr>

set security error [incomplete]<cr>
set security error [invalid key '*<key>*']<cr>
set security error [group devices not found]<cr>
set security error [device '*Encoder1*' not found]<cr>
set security error [device '*Encoder1*' disconnected]<cr>

ToC

8.1.8 Command set video_compress

8.1.8.1 Command usage

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

8.1.8.2 Description

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

8.1.8.3 Arguments

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

8.1.8.4 Notes

- The scaled sub stream remains uncompressed

8.1.8.5 Return Value

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

8.1.8.6 Command Examples

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

8.1.8.7 Return Examples

set video_compress success<cr>

set video_compress error [incomplete]<cr>
set video_compress error [invalid state '<*state*>']<cr>
set video_compress error [encoder '*Encoder1*' not found]<cr>
set video_compress error [encoder '*Encoder1*' disconnected]<cr>

ToC

8.1.9 Command set video_mode

8.1.9.1 Command usage

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

8.1.9.2 Description

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

8.1.9.3 Arguments

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

8.1.9.4 Notes

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

8.1.9.5 Return Value

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

8.1.9.6 Command Examples

set video_mode *Decoder1* sync<cr>
set video_mode *Decoder1* sync_scale *1920 1080*<cr>
set video_mode *Decoder1* fast *1920 1080 60*<cr>
set video_mode *Decoder1* fast keep *1920 1080 60*<cr>
set video_mode *Decoder1* fast crop *1920 1080 60*<cr>
set video_mode key:*abc123 Decoder1* fast stretch *1920 1080 60*<cr>

8.1.9.7 Return Examples

set video_mode success<cr>

set video_mode error [not supported]<cr>
set video_mode error [incomplete]<cr>
set video_mode error [invalid display_mode '<*display_mode*>']<cr>
set video_mode error [invalid width '<*width*>']<cr>
set video_mode error [invalid height '<*height*>']<cr>
set video_mode error [invalid fps'<*fps*>']<cr>
set video_mode error [decoder '*Decoder1*' not found]<cr>
set video_mode error [decoder '*Decoder1*' disconnected]<cr>
set video_mode error [group devices not found]<cr>

ToC

8.1.10 Command set video_mute

8.1.10.1 Command usage

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

8.1.10.2 Description

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

8.1.10.3 Arguments

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

8.1.10.4 Notes

- **color** string has a format of RRGGBB where each pair of hexadecimal numbers represent the 8-bit value

8.1.10.5 Return Value

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

8.1.10.6 Command Examples

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

8.1.10.7 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 color '<*color*>']<cr>
set video_mute error [decoder '*Decoder1*' not found]<cr>
set video_mute error [decoder '*Decoder1*' disconnected]<cr>

ToC

8.1.11 Command set video_source

8.1.11.1 Command usage

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

8.1.11.2 Description

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

8.1.11.3 Arguments

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

8.1.11.4 Notes

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

8.1.11.5 Return Value

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

8.1.11.6 Command Examples

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

8.1.11.7 Return Examples

set video_source success<cr>

set video_source error [not supported]<cr>
set video_source error [incomplete]<cr>
set video_source error [invalid value '<*value*>']<cr>
set video_source error [encoder '*Encoder1*' not found]<cr>
set video_source error [encoder '*Encoder1*' disconnected]<cr>

ToC

8.2 Command set for system

This sections contains set commands for the system.

8.2.1 Command set 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 tabs.

8.2.1.3 Arguments

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 Return Value

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

8.2.1.6 Command Examples

set events MyE*vent* enabled<cr>
set events MyE*vent* disabled<cr>

8.2.1.7 Return Examples

set events success<cr>

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

8.2.2 Command set listener

Not supported.

ToC

ToC

8.2.3 Command set presenter

8.2.3.1 Command usage

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

8.2.3.2 Description

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

8.2.3.3 Arguments

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

8.2.3.4 Notes

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

8.2.3.5 Return Value

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

8.2.3.6 Command Examples

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

8.2.3.7 Return Examples

set presenter success<cr>

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

ToC

8.2.4 Command set var

8.2.4.1 Command usage

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

8.2.4.2 Description

The command **set var** is used to create or change a user defined variable to store a string or value.

8.2.4.3 Arguments

var_name	Name of the variable
value	Maximum of 256 characters or [delete]

8.2.4.4 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".

8.2.4.5 Return Value

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

8.2.4.6 Command Examples

set var *myVar true*<cr>
set var *myVar "any string unto 256"*<cr>
set var *myVar* [delete]<cr>
set var key:*abc123 MyVar* false<cr>

8.2.4.7 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>

ToC

8.3 Command set for User Interfaces

This sections contains set commands for interaction with User Interfaces.

8.3.1 Command set ui

8.3.1.1 Command usage

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

8.3.1.2 Description

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

8.3.1.3 Arguments

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

8.3.1.4 Notes

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

8.3.1.5 Return Value

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

8.3.1.6 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>

8.3.1.7 Return Examples

set ui success<cr>

set ui error [incomplete]<cr>
set ui error [invalid parameter]<cr>
set ui error [invalid clients '<*value*>']<cr>
set ui error [invalid pin '<*value*>']<cr>
set ui error [ui '*myUI*' not found]<cr>

ToC

8.3.2 Command set ui_button

8.3.2.1 Command usage

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

8.3.2.2 Description

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

8.3.2.3 Arguments

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
value	up | down, enabled | disabled or text string depending on the function

8.3.2.4 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**
- **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, 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

8.3.2.5 Return Value

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

8.3.2.6 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

8.3.2.7 Return Examples

set ui_button success<cr>

set ui_button error [incomplete]<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 [ui '*myUI*' not found]<cr>
set ui_button error [button '*myButton*' not found]<cr>

ToC

8.3.3 Command set ui_image

8.3.3.1 Command usage

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

8.3.3.2 Description

The command **set ui_image** is used to control an image within a Control UI.

8.3.3.3 Arguments

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

8.3.3.4 Notes

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

8.3.3.5 Return Value

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

8.3.3.6 Command Examples

set ui_image *myUI myImage* visibility *false*<cr>
set ui_image *myUI myImage* visibility *true*<cr>

8.3.3.7 Return Examples

set ui_image success<cr>

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

8.3.4.7 Return Examples

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 '*myUI*' not found]<cr>
set ui_indicator error [indicator '*myIndicator*' not found]<cr>

ToC

8.3.5 Command set ui_label

8.3.5.1 Command usage

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

8.3.5.2 Description

The command **set ui_label** is used to control a label within a Control UI.

8.3.5.3 Arguments

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

8.3.5.4 Notes

- Control UI must be enabled as a feature for command to be used.
- function color uses HEX RGB color code from 000000 to FFFFFF
- function visibility uses true | false

8.3.5.5 Return Value

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

8.3.5.6 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 <<encoder>><cr>

8.3.5.7 Return Examples

set ui_label success<cr>

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

ToC

8.3.6 Command set ui_page

8.3.6.1 Command usage

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

8.3.6.2 Description

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

8.3.6.3 Arguments

ui_name	Name of the User Interface
page_name	Name of the page

8.3.6.4 Notes

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

8.3.6.5 Return Value

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

8.3.6.6 Command Examples

set ui_page *myUI myPage*<cr>

8.3.6.7 Return Examples

set ui_page success<cr>

set ui_page error [incomplete]<cr>
set ui_page error [invalid parameter]<cr>
set ui_page error [service disabled]<cr>
set ui_page error [ui '*myUI*' not found]<cr>
set ui_page error [page '*myPage*' not found]<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.

8.3.7.3 Arguments

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.

8.3.7.5 Return Value

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

8.3.7.6 Command Examples

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

8.3.7.7 Return Examples

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 '*myUI*' not found]<cr>
set ui_redirect error [page '*myPage*' 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.

8.3.8.3 Arguments

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.

8.3.8.5 Return Value

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

8.3.8.6 Command Examples

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

8.3.8.7 Return Examples

set ui_revert success<cr>

set ui_revert error [incomplete]<cr>
set ui_revert error [service disabled]<cr>
set ui_revert error [ui '*myUI*' 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.

8.3.9.3 Arguments

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>.

8.3.9.5 Return Value

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

8.3.9.6 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>>

8.3.9.7 Return Examples

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

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

ToC

9 Command get

The **get** commands are used to retrieve information.
Commands missing **mode** return:
get error [incomplete]<cr>

Commands with invalid **mode** return:
get error [invalid mode]<cr>

9.1 Command get for devices

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

9.1.1 Command get api

9.1.1.1 Command usage

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

9.1.1.2 Description

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

9.1.1.3 Arguments

None

9.1.1.4 Notes

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

9.1.1.5 Return Value

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

9.1.1.6 Command Example

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

9.1.1.7 Return Example

get api success *2.21.0.0*<cr>

ToC

9.1.2 Command get audio_io

9.1.2.1 Command usage

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

9.1.2.2 Description

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

9.1.2.3 Arguments

encoder_device_name	Device name of the Encoder

9.1.2.4 Notes

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

9.1.2.5 Return Value

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

9.1.2.6 Command Example

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

9.1.2.7 Return Examples

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

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

9.1.3 Command get audio_out

9.1.3.1 Command usage

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

9.1.3.2 Description

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

9.1.3.3 Arguments

decoder_device_name	Device name of the Decoder

9.1.3.4 Notes

- Returned mode is either **hdmi_2**, **analog** or **other**.
- Use the command **set audio_out** to change this setting.

9.1.3.5 Return Value

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

9.1.3.6 Command Example

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

9.1.3.7 Return Examples

get audio_out success hdmi_2<cr>
get audio_out success analog<cr>
get audio_out success other<cr>

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

9.1.4 Command get audio_source

9.1.4.1 Command usage

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

9.1.4.2 Description

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

9.1.4.3 Arguments

decoder_device_name	Device name of the Decoder

9.1.4.4 Notes

- Returned mode is either **analog, hdmi**, **hdmi_2**, **hdmi_sync** or **other**.
  - analog = ANALOG (Stereo Audio)
  - hdmi = HDMI (Multichannel Audio)
  - hdmi_2 = HDMI (Downmix Audio)
  - hdmi_sync = HDMI (Sync Mode)
- Use the command **set audio_source** to change this setting.

9.1.4.5 Return Value

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

9.1.4.6 Command Example

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

9.1.4.7 Return Examples

get audio_source success analog<cr>
get audio_source success hdmi<cr>
get audio_source success hdmi_2<cr>
get audio_source success hdmi_sync<cr>
get audio_source success other<cr>

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

ToC

9.1.5 Command get bandwidth / get rt_bandwidth

9.1.5.1 Command usage

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

9.1.5.2 Description

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

9.1.5.3 Arguments

device_name	Device name of either Encoder or Decoder
index	Stream or Subscription index (optional)
potential	Include stopped streams (optional)

9.1.5.4 Notes

- When **index** is omitted the total bandwidth will be returned. For an Encoder this will be the total transmitted bandwidth of stream 0 and stream 1. For a Decoder this will be the total received bandwidth of all unique subscriptions.
- **rt_bandwidth** is used to retrieve real time data rather than current status data.
- Keyword "potential" is used to retrieve the Encoder bandwidth assuming both stream are started.

9.1.5.5 Return Value

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

9.1.5.6 Command Example

get bandwidth *Encoder1*<cr>
get bandwidth *Encoder1 0*<cr>
get bandwidth *Encoder1 1*<cr>
get bandwidth *Encoder1* potential<cr>
get bandwidth *Decoder1*<cr>
get bandwidth *Decoder1 27*<cr>

get rt_bandwidth key:*abc123 Encoder1*<cr>
get bandwidth key:*abc123 Encoder1*<cr>

9.1.5.7 Return Example

get bandwidth success *3.33*<cr>
get rt_bandwidth success *3.33*<cr>

get bandwidth error [incomplete]<cr>
get bandwidth error [invalid parameter]<cr>
get bandwidth error [device '*Decoder1*' not found]<cr>
get bandwidth error [device '*Encoder1*' disconnected]<cr>

ToC

9.1.6 Command get devices

9.1.6.1 Command usage

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

9.1.6.2 Description

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

9.1.6.3 Arguments

target	all | all_rx | all_tx

9.1.6.4 Notes

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

9.1.6.5 Return Value

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

9.1.6.6 Command Example

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

9.1.6.7 Return Examples

get devices success ["*<device_name>-<device_id>*", "*<device_name>-<device_id>*"]<cr>
get devices success ["*<device_name>-<device_id>*"]<cr>

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

ToC

9.1.7 Command get display_status

9.1.7.1 Command usage

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

9.1.7.2 Description

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

9.1.7.3 Arguments

decoder_device_name	Device name of the Decoder

9.1.7.4 Notes

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

9.1.7.5 Return Value

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

9.1.7.6 Command Example

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

9.1.7.7 Return Examples

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

get display_status error [incomplete]<cr>
get display_status error [decoder '*Decoder1*' not found]<cr>
get display_status error [decoder '*Decoder1*' disconnected]<cr>

ToC

9.1.8 Command get edid

9.1.8.1 Command usage

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

9.1.8.2 Description

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

9.1.8.3 Arguments

decoder_device_name	Device name of the Decoder

9.1.8.4 Notes

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

9.1.8.5 Return Value

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

9.1.8.6 Command Example

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

9.1.8.7 Return Examples

get edid success
00ffffffffffff00410c2fc0c52d00000c140103802f1a782e3585a656489a241250542f6f00714f8180818a9500950fb3000101d1c0023a801871382d40582c4500dc0c1100001e000000ff00415535313031323031313731373137000000fd00384c1e5315000a20202020202000000fc0050686996c6970732032323145 0a01b602031ef04b1005010203040612131 41f2309070783010000650 30c001100023a801871382d40582c4500dc0c1100001e8c0ad08a20e02d10103e9600dc0c11000018011d007251d01e206e285500dc0c1100001e8c0ad090204031200c405500dc0c11000018000000000000000000000000000000000000000000000000000000000000ac<cr>

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

ToC

9.1.9 Command get frame_converter

9.1.9.1 Command usage

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

9.1.9.2 Description

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

9.1.9.3 Arguments

encoder_device_name	Device name of the Encoder
stream	sub | main

9.1.9.4 Notes

None

9.1.9.5 Return Value

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

9.1.9.6 Examples

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

9.1.9.7 Return Examples

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

get frame_converter error [not supported]<cr>
get frame_converter error [incomplete]<cr>
get frame_converter error [invalid stream '<*stream*>']<cr>
get frame_converter error [encoder '*Encoder1*' not found]<cr>
get frame_converter error [encoder '*Encoder1*' disconnected]<cr>

9.1.10 Command get joins

9.1.10.1 Command usage

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

9.1.10.2 Description

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

9.1.10.3 Arguments

device_name	Device name of a Decoder or Destination
subscription	audio_a | audio_d | video | serial | ir | usb | usbhid
index	Optional video index from 0 to 31

9.1.10.4 Notes

- **index** is only required for subscription **video**. Index 0 will be used by default if omitted.

9.1.10.5 Return Value

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

9.1.10.6 Command Examples

get joins *Decoder1* video *0*<cr>
get joins *Decoder1* audio_d<cr>
get joins *Decoder1* audio_a<cr>
get joins key:*abc123 Decoder1* video<cr>
get joins *Decoder1* ir<cr>
get joins *Decoder1* serial<cr>
get joins *Decoder1* usb<cr>
get joins *Decoder1* usbhid<cr>

9.1.10.7 Return Examples

get joins success *Encoder1*<cr>
get joins success null<cr>

get joins error [incomplete]<cr>
get joins error [invalid subscription '<*subscription*>']<cr>
get joins error [invalid index '<*index*>']<cr>
get joins error [device '*Decoder1*' not found]<cr>
get joins error [device '*Decoder1*' disconnected]<cr>

ToC

9.1.11 Command get json

9.1.11.1 Command usage

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

9.1.11.2 Description

The command **get json** is used to retrieve complete device status as a json payload.

9.1.11.3 Arguments

device_name	Device name of the Decoder or Encoder

9.1.11.4 Notes

9.1.11.5 Return Value

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

9.1.11.6 Command Example

get json Encoder1<cr>
get json Decoder1<cr>

9.1.11.7 Return Examples

get json success {…}<cr>

get json error [incomplete]<cr>
get json error [device '*Encoder1*' not found]<cr>
get json error [device '*Decoder1*' disconnected]<cr>

ToC

9.1.12 Command get preferred

9.1.12.1 Command usage

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

9.1.12.2 Description

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

9.1.12.3 Arguments

decoder_device_name	Device name of Decoder
option	width | height | fps

9.1.12.4 Notes

- 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.

9.1.12.5 Return Value

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

9.1.12.6 Command Example

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

9.1.12.7 Return Example

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

get preferred error [incomplete]<cr>
get preferred error [no EDID available]<cr>
get preferred error [invalid option'*<option>*']<cr>
get preferred error [decoder '*Decoder1'* not found]<cr>
get preferred error [decoder '*Decoder1'* disconnected]<cr>

ToC

9.1.13 Command get scaler

9.1.13.1 Command usage

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

9.1.13.2 Description

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

9.1.13.3 Arguments

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

9.1.13.4 Notes

None

9.1.13.5 Return Value

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

9.1.13.6 Command Examples

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

9.1.13.7 Return Examples

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

get scaler error [not supported]<cr>
get scaler error [incomplete]<cr>
get scaler error [invalid option'<*option*>']<cr>
get scaler error [encoder '*Encoder1*' not found]<cr>
get scaler error [encoder '*Encoder1*' disconnected]<cr>

9.1.14 Command get security

9.1.14.1 Command usage

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

9.1.14.2 Description

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

9.1.14.3 Arguments

device_name	Device name of Encoder or Decoder

9.1.14.4 Notes

None

9.1.14.5 Return Value

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

9.1.14.6 Command Example

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

9.1.14.7 Return Example

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

get security error [incomplete]<cr>
get security error [device '*Encoder1*' not found]<cr>
get security error [device '*Decoder1*' disconnected]<cr>

ToC

9.1.15 Command get status

9.1.15.1 Command usage

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

9.1.15.2 Description

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

9.1.15.3 Arguments

device_name	Name of the Encoder or Decoder
transceiver_function	Used only with a transceiver device to specify 'encoder' or 'decoder'
stream	audio_a | audio_d | video
index	To be used with "video" only. For Encoder 0 – 1 and for Decoder 0 – 31.

9.1.15.4 Notes

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

9.1.15.5 Return Value

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

9.1.15.6 Command Example

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

get status *Decoder1*<cr>
get status *Decoder1* audio_a<cr>
get status *Decoder1* audio_d<cr>
get status *Decoder1* video 0<cr>
get status *Decoder1* video *31*<cr>
get status *Transceiver1 decoder* video *31*<cr>

9.1.15.7 Return Examples

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

get status error [incomplete]<cr>
get status error [invalid transceiver_function '< transceiver_function >']<cr>
get status error [invalid stream '<*stream*>']<cr>
get status error [invalid index '<*index*>']<cr>
get status error [device '*Encoder1*' not found]<cr>

ToC

9.1.16 Command get temp

9.1.16.1 Command usage

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

9.1.16.2 Description

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

9.1.16.3 Arguments

device_name	Device name of either a Decoder or Encoder

9.1.16.4 Notes

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

9.1.16.5 Return Value

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

9.1.16.6 Command Examples

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

9.1.16.7 Return Examples

get temp success *69*<cr>

get temp error [incomplete]<cr>
get temp error [device '*Encoder1*' not found]<cr>
get temp error [device '*Decoder1*' disconnected]<cr>

9.1.17 Command get ver

9.1.17.1 Command usage

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

9.1.17.2 Description

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

9.1.17.3 Arguments

device_name	Device name of either a Decoder or Encoder

9.1.17.4 Notes

9.1.17.5 Return Value

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

9.1.17.6 Command Examples

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

9.1.17.7 Return Examples

get ver success *3.5.2.0*<cr>

get ver error [incomplete]<cr>
get ver error [device '*Encoder1*' not found]<cr>
get ver error [device '*Decoder1*' disconnected]<cr>

ToC

9.1.18 Command get video / get rt_video

9.1.18.1 Command usage

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

9.1.18.2 Description

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

9.1.18.3 Arguments

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

9.1.18.4 Notes

- **all** returns <*width*> <*height*> <*frames_per_second*> <*color_space*> <*bits_per_pixel*> <*scan_mode*> <hdcp>
- **rt_video** is used to retrieve real time data rather than current status data.
- **hdcp** returns as false or HDCP version 1.4 or 2.2.

9.1.18.5 Return Value

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

9.1.18.6 Command Example

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

9.1.18.7 Return Examples

get video success *1920 1080 60 YCBCR_444 8 PROGRESSIVE 1.4*<cr>
get video success *1920*<cr>
get video success *1080*<cr>
get video success *60*<cr>
get video success RGB<cr>
get video success YCBCR_444<cr> *can also be YCBCR_422 or YCBCR_420
get video success *8*<cr> *can also be 10 or 12
get video success PROGRESSIVE<cr> * can also be INTERLACED
get video success false<cr> * can also be 1.4 or 2.2
get rt_video success *1920 1080 60 YCBCR_444 8 PROGRESSIVE false*<cr>

get video error [incomplete]<cr>
get video error [invalid option'<*option*>']<cr>
get video error [encoder '*Encoder1*' not found]<cr>
get video error [encoder '*Encoder1*' disconnected]<cr>

ToC

9.1.19 Command get video_compress

9.1.19.1 Command usage

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

9.1.19.2 Description

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

9.1.19.3 Arguments

encoder_device_name	Device name of the Encoder

9.1.19.4 Notes

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

9.1.19.5 Return Value

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

9.1.19.6 Command Example

get video_compress *Encoder1*<cr>
get video_compress key:*abc123 Encoder1*<cr>

9.1.19.7 Return Examples

get video_compress success true<cr>
get video_compress success false<cr>

get video_compress error [not supported]<cr>
get video_compress error [incomplete]<cr>
get video_compress error [encoder '*Encoder1*' not found]<cr>
get video_compress error [encoder '*Encoder1*' disconnected]<cr>

9.1.20 Command get video_mode

9.1.20.1 Command usage

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

9.1.20.2 Description

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

9.1.20.3 Arguments

decoder_device_name	Device name of Decoder

9.1.20.4 Notes

None

9.1.20.5 Return Value

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

9.1.20.6 Command Example

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

9.1.20.7 Return Example

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

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

9.1.21 Command get video_mute

9.1.21.1 Command usage

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

9.1.21.2 Description

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

9.1.21.3 Arguments

decoder_device_name	Device name of the Decoder

9.1.21.4 Notes

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

9.1.21.5 Return Value

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

9.1.21.6 Command Example

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

9.1.21.7 Return Examples

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

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

ToC

9.1.22 Command get video_source

9.1.22.1 Command usage

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

9.1.22.2 Description

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

9.1.22.3 Arguments

encoder_device_name	Device name of the Encoder

9.1.22.4 Notes

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

9.1.22.5 Return Value

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

9.1.22.6 Command Example

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

9.1.22.7 Return Examples

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

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

ToC

9.1.23 Command get video_status / get rt_video_status

9.1.23.1 Command usage

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

9.1.23.2 Description

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

9.1.23.3 Arguments

encoder_device_name	Device name of the Encoder

9.1.23.4 Notes

- Returned mode is either **true** or **false.**
- **rt_video_status** is used to retrieve real time data rather than current status data.

9.1.23.5 Return Value

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

9.1.23.6 Command Example

get video_status *Encoder1*<cr>
get video_status key:*abc123 Encoder1*<cr>
get rt_video_status key:*abc123 Encoder1*<cr>

9.1.23.7 Return Examples

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

get video_status error [incomplete]<cr>
get video_status error [encoder '*Encoder1*' not found]<cr>
get video_status error [encoder '*Encoder1*' disconnected]<cr>

ToC

9.1.24 Command get window / get rt_window

9.1.24.1 Command usage

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

9.1.24.2 Description

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

9.1.24.3 Arguments

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

9.1.24.4 Notes

- The layout must already be executed by the SDVoE Controller which is then held in memory.
- **rt_window** is used to retrieve real time data rather than current status data.

9.1.24.5 Return Value

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

9.1.24.6 Command Example

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

9.1.24.7 Return Examples

get window success *1920 1080*<cr>
get window success *1920*<cr>
get window success *1080*<cr>
get rt_window success *1920 1080*<cr>

get window error [incomplete]<cr>
get window error [layout '*MyLayout*' not found]<cr>
get window error [index '*0*' not found]<cr>
get window error [invalid option'<*option*>']<cr>

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 and Events tabs.

9.2.1.3 Arguments

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

9.2.1.4 Notes

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

9.2.1.5 Return Value

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 Return Examples

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

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

ToC

9.2.2 Command get matrix

9.2.2.1 Command usage

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

9.2.2.2 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.

9.2.2.3 Arguments

stream	audio_a | audio_d | video | serial | ir | usb | usbhid
index (optional)	Optional video index from 0 to 31

9.2.2.4 Notes

- null will be the result when device is not joined with specified stream type.
- Video index 0 will be used by default if omitted.
- *Command renamed from 'get joins'*

9.2.2.5 Return Value

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

9.2.2.6 Command Example

get matrix audio_a<cr>
get matrix audio_d<cr>
get matrix video<cr>
get matrix video 1<cr>
get matrix serial<cr>
get matrix ir<cr>
get matrix usb<cr>
get matrix usbhid<cr>

9.2.2.7 Return Examples

get matrix success [<decoder_device_name:<encoder_device_name> / null, …]<cr>
get matrix error [incomplete]<cr>
get matrix error [invalid stream]<cr>
get matrix error [invalid index]<cr>

ToC

9.2.3 Command get presenter

9.2.3.1 Command usage

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

9.2.3.2 Description

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

9.2.3.3 Arguments

group	Name of the Group

9.2.3.4 Notes

- Group names have no spaces eg 'group1'

9.2.3.5 Return Value

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

9.2.3.6 Command Example

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

9.2.3.7 Return Example

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

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

ToC

9.2.4 Command get var

9.2.4.1 Command usage

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

9.2.4.2 Description

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

9.2.4.3 Arguments

var_name	Name of the variable

9.2.4.4 Notes

9.2.4.5 Return Value

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

9.2.4.6 Command Example

get var *MyVar*<cr>

9.2.4.7 Return Examples

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.

9.3.1 Command get ui

9.3.1.1 Command usage

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

9.3.1.2 Description

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

9.3.1.3 Arguments

ui_name	Name of the User Interface

9.3.1.4 Notes

- User Interface must be enabled as a feature for command to be used.

9.3.1.5 Return Value

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

9.3.1.6 Command Example

get ui *MyUI*<cr>

9.3.1.7 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 [incomplete]<cr>
get ui error [controlui '*MyUI*' not found]<cr>

ToC

9.3.2 Command get ui_button

9.3.2.1 Command usage

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

9.3.2.2 Description

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

9.3.2.3 Arguments

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

9.3.2.4 Notes

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

9.3.2.5 Return Value

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

9.3.2.6 Command Examples

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

9.3.2.7 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 [invalid parameter]<cr>
get ui_button error [service disabled]<cr>
get ui_button error [ui '<*ui_name*>' not found]<cr>
get ui_button error [button '<*button_name*>' not found]<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.

9.3.3.3 Arguments

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

9.3.3.4 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>.

9.3.3.5 Return Value

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

9.3.3.6 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

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

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.

9.3.4.3 Arguments

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

9.3.4.4 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>.

9.3.4.5 Return Value

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

9.3.4.6 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

9.3.4.7 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 [invalid parameter]<cr>
get ui_slider error [service disabled]<cr>
get ui_slider error [ui '*myUI*' not found]<cr>
get ui_slider error [slider '*mySider*' not found]<cr>

ToC

10 Command send

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

Commands missing **mode** return:

send error [incomplete]<cr>

Commands with invalid **mode** return:

send error [invalid mode]<cr>

ToC

10.1 Command send infrared

10.1.1 Command usage

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

10.1.2 Description

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

10.1.3 Arguments

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

10.1.4 Notes

- The **data_hex** argument is a hexadecimal string which represent the Pronto infrared code to be sent.
- Its length must be a multiple of eight (i.e. data length must be a multiple of four bytes) and cannot exceed 256 burst pairs and a maximum length of 1032 bytes.
- ***group_name*** is used as a destination when all Encoders and Decoders in a group are required to send IR.

10.1.5 Return Value

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

10.1.6 Command Examples

send ir *Decoder1 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015*
00150015004000150040001500400015001500150015001500150015001500150015001500150015001500150015001500150040001500400
01500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150015001500150040001500400
01500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150015001500150040001500400
01500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150015001500150040001500689<cr>

send ir *MyGroup 0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015*
00150015004000150040001500400015001500150015001500150015001500150015001500150015001500150015001500150040001500400
01500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150015001500150040001500400
01500150015001500150040001500150015004000150040001500150015001500150015004000150040001500150015001500150040001500689<cr>

send ir key:*abc123* all_rx *0000006D0000002200AC00AC00150040001500400015004000150015001500150015001500150015001500150015*
00150015004000150040001500400015001500150015001500150015001500150015001500150015001500150015001500150040001500400
01500400015001500150015001500150040001500150015004000150040001500150015001500150040001500150040001500150015001500400
01500400015001500150015001500150040001500150015004000150040001500150015001500150040001500150040001500150015001500400
01500400015001500150015001500150040001500150015004000150040001500150015001500150040001500150040001500150015001500689<cr>

ToC

10.1.7 Return Examples

send ir success<cr>

send ir error [not supported]<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 8 bytes]<cr>
send ir error [invalid format]<cr>
send ir error [device '*Encoder1*' not found]<cr>
send ir error [device '*Decoder1*' disconnected]<cr>
send ir error [group devices not found]<cr>

10.2 Command send serial

10.2.1 Command usage

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

10.2.2 Description

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

10.2.3 Arguments

device_name	Device name of Encoder, Decoder, Group or '**all**', '**all_rx**', '**all_tx**'
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

10.2.4 Notes

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

- The **data_string** argument is a printable ASCII text string of any length. This string must be escaped if it contains ASCII control characters or non-ASCII byte values. Space characters must also be escaped. The following escape sequences are recognized:
  - \0 null character
  - \n line feed
  - \r carriage return
  - \t horizontal tab
  - \" double quote
  - \xnn (where nn are two hexadecimal characters) hexadecimal representation of byte
  - \\ a single backslash
  - \ (space) space character

- 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

10.2.5 Return Value

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

10.2.6 Command Examples

send serial *Decoder1* "*my data string*"\r<cr>
send serial *Decoder1* "*my data string*" *reply*\r\n<cr>
send serial *Encoder1* "*\x00\x01\x02\x03\x04*"<cr>
send serial *Decoder1* "*my data string*\r\n" contains "*OK*"<cr>
send serial *Decoder1* "*my data string*\r\n" equals "*OK*"<cr>
send serial *MyGroup* "*my data string*"\r<cr>
send serial key:*abc123 Decoder1* "*my data string*\r"<cr>

10.2.7 Return Examples

send serial success [*my return string*]<cr>
send serial success [00FF0D]<cr>
send serial success [\x00\xFF\x0D]<cr>
send serial success []<cr>
send serial success<cr>

send serial error [not supported]<cr>
send serial error [incomplete]<cr>
send serial error [invalid format]<cr>
send serial error [invalid parameter]<cr>
send serial error [invalid HEX data]<cr>
send serial error [invalid HEX feedback]<cr>
send serial error [*feedback string*]<cr>
send serial error [device '*Encoder1*' not found]<cr>
send serial error [device 'Decoder1' disconnected]<cr>
send serial error [group devices not found]<cr>

10.3 Command send cec

10.3.1 Command usage

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

10.3.2 Description

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

10.3.3 Arguments

device_name	Device name of the Decoder, Encoder, Group or '**all**', '**all_tx**', '**all_rx**'
index	Transceivers HDMI port index (optional) 0 used when not specified.
data_hex	String of ascii characters representing the hexadecimal cec code

10.3.4 Notes

- The **data_hex** argument is a hexadecimal string which represent the cec code to be sent.
- 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.
- ***index*** is only a requirement while a Transceiver's Encoder and Decoder have the same name. Otherwise the index will be automatically selected from the device name.

10.3.5 Return Value

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

10.3.6 Command Examples

send cec *Decoder1 F004*<cr>
send cec *Decoder1:1 F004*<cr>
send cec *MyGroup F004*<cr>
send cec key:*abc123 Decoder1 F004*<cr>

10.3.7 Return Examples

send cec success<cr>

send cec error [incomplete]<cr>
send cec error [the destination did not acknowledge the command]<cr>
send cec error [<device cec error message>]<cr>
send cec error [device '*Decoder1*' not found]<cr>
send cec error [device '*Encoder1*' disconnected]<cr>
send cec error [group devices not found]<cr>

ToC

10.4 Command send gc

Not supported.

ToC

10.5 Command send tcp

10.5.1 Command usage

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

10.5.2 Description

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

10.5.3 Arguments

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

10.5.4 Notes

- The TCP device must be in the same range as the SDVoE 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.

10.5.5 Return Value

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

10.5.6 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>

ToC

10.5.7 Return Examples

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

send tcp error [incomplete]<cr>
send tcp error [invalid parameter]<cr>
send tcp error [invalid format]<cr>
send tcp error [device '*172.30.1.111*' not found]<cr>
send tcp error [<feedback>]<cr>

ToC

11 Commands for Multiview

This section covers all the supported multiview commands.

11.1 Command multiview

11.1.1 Command usage

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

11.1.2 Description

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

11.1.3 Arguments

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

11.1.4 Notes

- A layout must be created before using this command by the command **layout new**.

11.1.5 Return Value

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

11.1.6 Examples

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

11.1.7 Return Examples

multiview success<cr>

multiview error [not supported]<cr>
multiview error [incomplete]<cr>
multiview error [invalid value '<value>']<cr>
multiview error [layout '*MyLayout*' not found]<cr>
multiview error [video format is not supported on this device]<cr>
multiview error [decoder '*Decoder1*' not found]<cr>
multiview error [decoder '*Decoder1*' disconnected]<cr>

ToC

11.2 Command layout

The **layout** commands are used to define a multiview layout and are stored on the controller. Layouts can then be edited and applied to a Decoder using the command **multiview**.

Commands missing **mode** return:

layout error [incomplete]<cr>

Commands with invalid **mode** return:

layout error [invalid mode]<cr>

11.2.1 Command layout new

11.2.1.1 Command usage

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

11.2.1.2 Description

The **layout new** command is used to initially create a multiview layout or reset an existing layout with the specified name. Once the multiview layout has been created with this **layout new** command, use the **layout window** command to define the windows within the multiview layout.

11.2.1.3 Arguments

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

11.2.1.4 Notes

- The **width** argument must be a multiple of 2 pixels
- **layout_name** must start with alpha characters only.

11.2.1.5 Return Value

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

11.2.1.6 Command Examples

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

11.2.1.7 Return Examples

layout new success<cr>

layout new error [incomplete]<cr>
layout new error [invalid width '*<width>*']<cr>
layout new error [invalid height '*<height>*']<cr>
layout new error [invalid layout name argument in command 'layout']<cr>

ToC

11.2.2 Command layout window

11.2.2.1 Command usage

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

11.2.2.2 Description

The **layout window** command is used to define the window size and location in the multiview display. Once the multiview layout has been created with the **layout new** command, use this **layout window** command to define the windows within the multiview layout.
A window is a high-level abstraction provided by the API. It is a rectangular area on the screen that contains either the rectangular area of a surface or black. Windows can overlap, when this occurs the content of the window with the lowest index is shown in the overlapping area.

11.2.2.3 Arguments

layout_name	The name given to the multiview layout
index	Index of the window, between 0 and 31
horiz_position	Horizontal position in pixels
vert_position	Vertical position in pixels
width	Horizontal size in pixels
height	Vertical size in pixels
offset	Keyword to specify an offset to the displayed image (optional)
horiz_offset	Horizontal offset in pixels (required with offset)
vert_offset	Vertical offset in pixels (required with offset)
subscription	Decoder index to be used with a join command, between 0 and 31

11.2.2.4 Notes

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

11.2.2.5 Return Value

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

11.2.2.6 Examples

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

ToC

11.2.2.7 Return Examples

layout window success<cr>

layout window error [incomplete]<cr>
layout window error [invalid value '<value>']<cr>
layout window error [invalid subscription '<subscription>']<cr>
layout window error [layout '<layout_name>' not found]<cr>
layout window error [window width must be at least 4]<cr>
layout window error [window width and horizontal position must be even]<cr>
layout window error [window horizontal offset must be even]<cr>
layout window error [window dimension is zero]<cr>

ToC

11.2.3 Command layout black

11.2.3.1 Command usage

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

11.2.3.2 Description

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

11.2.3.3 Arguments

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

11.2.3.4 Notes

11.2.3.5 Return Value

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

11.2.3.6 Command Examples

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

11.2.3.7 Command Examples

layout black success<cr>

layout black error [incomplete]<cr>
layout black error [invalid index '<*index*>']<cr>
layout black error [layout '*MyLayout*' not found]<cr>

11.2.4 Command layout delete

11.2.4.1 Command usage

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

11.2.4.2 Description

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

11.2.4.3 Arguments

layout_name	The name given to the multiview layout

11.2.4.4 Notes

11.2.4.5 Return Value

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

11.2.4.6 Command Examples

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

11.2.4.7 Command Examples

layout delete success<cr>

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

ToC

11.2.5 Command layout surface

11.2.5.1 Command usage

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

11.2.5.2 Description

The **layout surface** command is used to create, modify or delete a surface in specified layout.
A surface is a rectangular area within a Decoder's frame buffer reserved for use by a specific video source. Each Decoder running in multiview mode, supports up to 32 surfaces and each surface is associated with the HDMI subscription with the same index.

11.2.5.3 Arguments

layout_name	The name given to the multiview layout
index	Index of the window, between 0 and 31
horiz_position	Horizontal start position of the surface in the frame buffer, in pixels (optional)
vert_position	Vertical start positions of the surface in the frame buffer, in pixels (optional)
width	Horizontal surface size in pixels (optional)
height	Vertical surface size in pixels (optional)
delete	Keyword to delete the layout surface (optional)

11.2.5.4 Notes

- The **width** arguments must be a multiple of 8 pixels.
- Default surfaces are provided when a layout is created which are adequate for most multiview layouts.

11.2.5.5 Return Value

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

11.2.5.6 Examples

layout surface *MyLayout 1 0 0 1920 1080*<cr>
layout surface key:*abc123 MyLayout* delete<cr>
layout error [invalid parameter]<cr>

11.2.5.7 Command Examples

layout surface success<cr>

layout surface error [incomplete]<cr>
layout surface error [layout '<layout_name>' not found]<cr>
layout surface error [invalid value '<value>']<cr>
layout surface error [surface dimension is zero]<cr>

ToC

12 Message notify

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

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

12.1 Message notify serial

12.1.1 Message received

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

12.1.2 Description

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

12.1.3 Arguments

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

12.1.4 Notes

- Before **notify serial** messaged can be received, the join serial command must be used to join the device to the SDVoE Controller. *Refer to 4.8 'Command join serial'.*
- **device_name** as wrapped with single quotation 'Decoder1'.

12.1.5 Received Value

message	<space>	**mode**	<space>	**device_name**	<space>	**data_string**	**terminator**
notify	<space>	serial	<space>	*Decoder1*	<space>	*ascii_data*	<cr>

12.1.6 Examples

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

ToC

12.2 Message notify network

12.2.1 Message received

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

12.2.2 Description

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

12.2.3 Arguments

device_name	Device name of Encoder, Decoder or USB Extender
state	true | false

12.2.4 Notes

- A **notify network** message will be sent when the SDVoE 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.
- **device_name** as wrapped with single quotation 'Decoder1'.

12.2.5 Received Value

message	<space>	**mode**	<space>	**device_name**	<space>	**status**	**terminator**
notify	<space>	network	<space>	*Decoder1*	<space>	true | false	<cr>

12.2.6 Examples

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

12.3 Message notify display

12.3.1 Message received

notify display <*decoder_device_name*> <*state*> <*edid*><cr>

12.3.2 Description

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

12.3.3 Arguments

decoder_device_name	Device name of Decoder
state	true | false
edid	EDID of connected display

12.3.4 Notes

- Some non-compliant displays will require power for this event to be raised.
- **decoder_device_name** as wrapped with single quotation 'Decoder1'.

12.3.5 Received Value

message	<space>	**mode**	<space>	**decoder_device_name**	<space>	**status**	**terminator**
notify	<space>	display	<space>	*Decoder1*	<space>	true | false	<cr>

12.3.6 Examples

notify display '*Decoder1*' false<cr>
notify display '*Decoder1*' true [*00ffffffffffff00…*]<cr>

12.4 Message notify source

12.4.1 Message received

notify source <*encoder_device_name*> <*state*> <*video_details*><cr>

12.4.2 Description

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

12.4.3 Arguments

encoder_device_name	Device name of Encoder
state	true | false
video_details	Source video details, same details as 'get video' returns

12.4.4 Notes

- The SDVoE Controller is looking for a stable video signal.
- **encoder_device_name** as wrapped with single quotation 'Encoder1'.

12.4.5 Received Value

message	<space>	**mode**	<space>	**encoder_device_name**	<space>	**status**	**terminator**
notify	<space>	source	<space>	*Encoder1*	<space>	true | false	<cr>

12.4.6 Examples

notify source '*Encoder1*' false<cr>
notify source '*Encoder1*' true [*1920 1080 60 YCBCR_444 8 PROGRESSIVE false*]<cr>
notify source '*Encoder1*' true [*1920 1080 60 YCBCR_444 8 PROGRESSIVE 2.2*]<cr>

12.5 Message notify stream

12.5.1 Message received

notify stream <*decoder_device_name*> <*encoder_device_name*> <*stream*><cr>
notify stream <*client_device_name*> <*host_device_name*> <*stream*><cr>

12.5.2 Description

A **notify stream** message is sent whenever a Decoder stream subscription or USB Client pairing changes.

12.5.3 Arguments

decoder_device_name	Device name of Decoder
encoder_device_name	Device name of Encoder
client_device_name	Device name of USB Client
host_device_name	Device name of USB Host
stream	video 0..31 | audio_a | audio_d I serial | ir | usb | usbhid

12.5.4 Notes

- **decoder_device_name / *host_device_name*** is wrapped with single quotation 'Decoder1'.
- **encoder_device_name / *client_device_name*** is wrapped with single quotation 'Encoder1' or null.
- **encoder_device_name / *client_device_name*** will return as **null** after a **leave** command.

12.5.5 Received Value

message	<space>	**mode**	<space>	**decoder_device_name**	<space>	**encoder_device_name**	<space>	**stream**	**terminator**
notify	<space>	stream	<space>	*Decoder1*	<space>	*Encoder1*	<space>	video 0..31 | audio_a | audio_d I serial | ir | usb | usbhid	<cr>

12.5.6 Examples

notify stream '*Decoder1*' '*Encoder1*' video 0<cr>
notify stream '*Decoder1*' '*Encoder1*' video 31<cr>
notify stream '*Decoder1*' '*Encoder1*' audio_a<cr>
notify stream '*Decoder1*' '*Encoder1*' audio_d<cr>
notify stream '*Decoder1*' '*Encoder1*' serial<cr>
notify stream '*Decoder1*' '*Encoder1*' ir<cr>
notify stream '*Client1*' '*Host1*' usb<cr>
notify stream '*Client1*' '*Host1*' usbhid<cr>

notify stream '*Decoder1*' null video 0<cr>
notify stream '*Decoder1*' null video 31<cr>
notify stream '*Decoder1*' null audio_a<cr>
notify stream '*Decoder1*' null audio_d<cr>
notify stream '*Decoder1*' null serial<cr>
notify stream '*Decoder1*' null ir<cr>
notify stream '*Clinet1*' null usb<cr>
notify stream '*Client1*' null usbhid<cr>

ToC

13 Command preset

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

Commands missing **mode** return:

preset error [incomplete]<cr>

Commands with invalid **mode** return:

preset error [invalid mode]<cr>

13.1 Command preset add

13.1.1 Command usage

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

13.1.2 Description

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

13.1.3 Arguments

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

13.1.4 Notes

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

13.1.5 Return Value

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

13.1.6 Command Examples

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

13.1.7 Return Examples

preset add *preset1* success<cr>

preset add error [incomplete]<cr>

13.2 Command preset load

13.2.1 Command usage

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

13.2.2 Description

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

13.2.3 Arguments

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

13.2.4 Notes

- The preset is created with the **preset add** command or directly via the UI.
- Presets are stored in the SDVoE Controller.
- Optional "***delay***" is used to delay to 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.

13.2.5 Return Value

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

13.2.6 Command Examples

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

13.2.7 Return Examples

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

13.3 Command preset delete

13.3.1 Command usage

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

13.3.2 Description

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

13.3.3 Arguments

preset_name	The name defined as the preset

13.3.4 Notes

13.3.5 Return Value

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

13.3.6 Command Examples

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

13.3.7 Return Examples

preset delete *preset1* success<cr>

preset delete error [incomplete]<cr>
preset delete error [preset '*preset1*' not found]<cr>

ToC

13.4 Command preset delay

13.4.1 Command usage

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

13.4.2 Description

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

13.4.3 Arguments

milliseconds	Delay time in milliseconds up to 9999

13.4.4 Notes

- This command can only be used within a preset.

13.4.5 Return Value

None

13.4.6 Command Examples

preset delay *1000*

13.4.7 Return Example

No return is given for a valid command

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

ToC

13.5 Command preset wait

13.5.1 Command usage

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

13.5.2 Description

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

13.5.3 Arguments

13.5.4 Notes

- This command can only be used within a preset.

13.5.5 Return Value

None

13.5.6 Command Example

preset wait

13.5.7 Return Example

None

ToC

13.6 Command preset dynamic

13.6.1 Command usage

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

13.6.2 Description

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

13.6.3 Arguments

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

13.6.4 Notes

- When the video wall preset is no longer active, you must issue a preset dynamic <encoder> false command to prevent the wall preset from being applied each time the Encoder is connected to a source.

- All active dynamic presets for a given Encoder will automatically be disabled when any of the following commands are used on an active Encoder:
  join fast, join sync, join sync_scale, join adv, join wall, join walladv, stop video, stop av, reboot & reset

13.6.5 Return Value

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

13.6.6 Command Examples

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

13.6.7 Return Examples

preset dynamic *preset1* success<cr>

preset dynamic error [incomplete]<cr>
preset dynamic error [invalid state]<cr>
preset dynamic error [preset '*preset1*' not found]<cr>

ToC

13.7 Command preset inactiveBypass

13.7.1 Command usage

preset inactiveBypass

13.7.2 Description

The command **preset inactiveBypass** is used within a preset to bypass non active device errors. If placed on the first or last line of the preset then all preset commands will bypass non active device errors otherwise only from the location of the command down.

13.7.3 Arguments

13.7.4 Notes

- This command can only be used within a preset.
- Without the use of this command a preset will stop executing on first error.

13.7.5 Return Value

None

13.7.6 Command Example

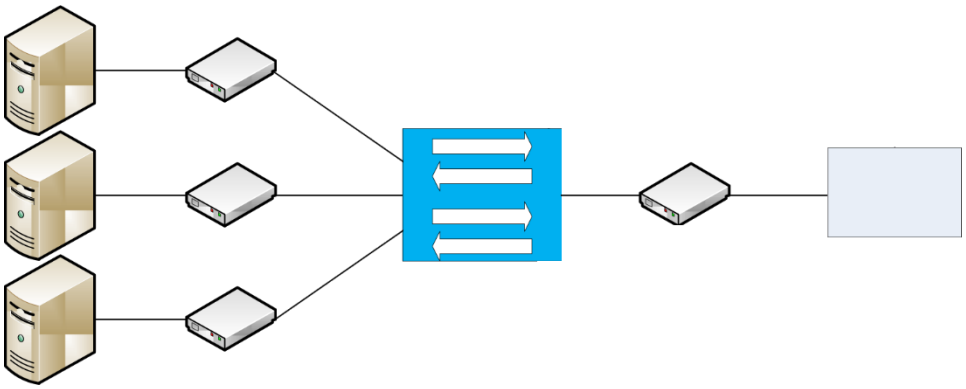
```
preset inactiveBypass
```

13.7.7 Return Example

None

ToC

Appendix A - How to Multiview



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

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

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

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

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

Appendix A - How to Multiview continued…

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

- A **surface** is a rectangular area within the Decoder's frame buffer that is reserved for a specific video source. A Decoder supports up to 32 surfaces and each surface is associated with the HDMI subscription that has the same index (e.g. the video from HDMI subscription 14 goes into surface 14).

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

- A **window** is a rectangular area on the screen that either contains a rectangular area of a surface or is black. Windows can overlap, in which case the content of the window with the smallest index is shown. Multiple tiles may be required to represent each window (when windows overlap) and the black areas that are not part of any window.

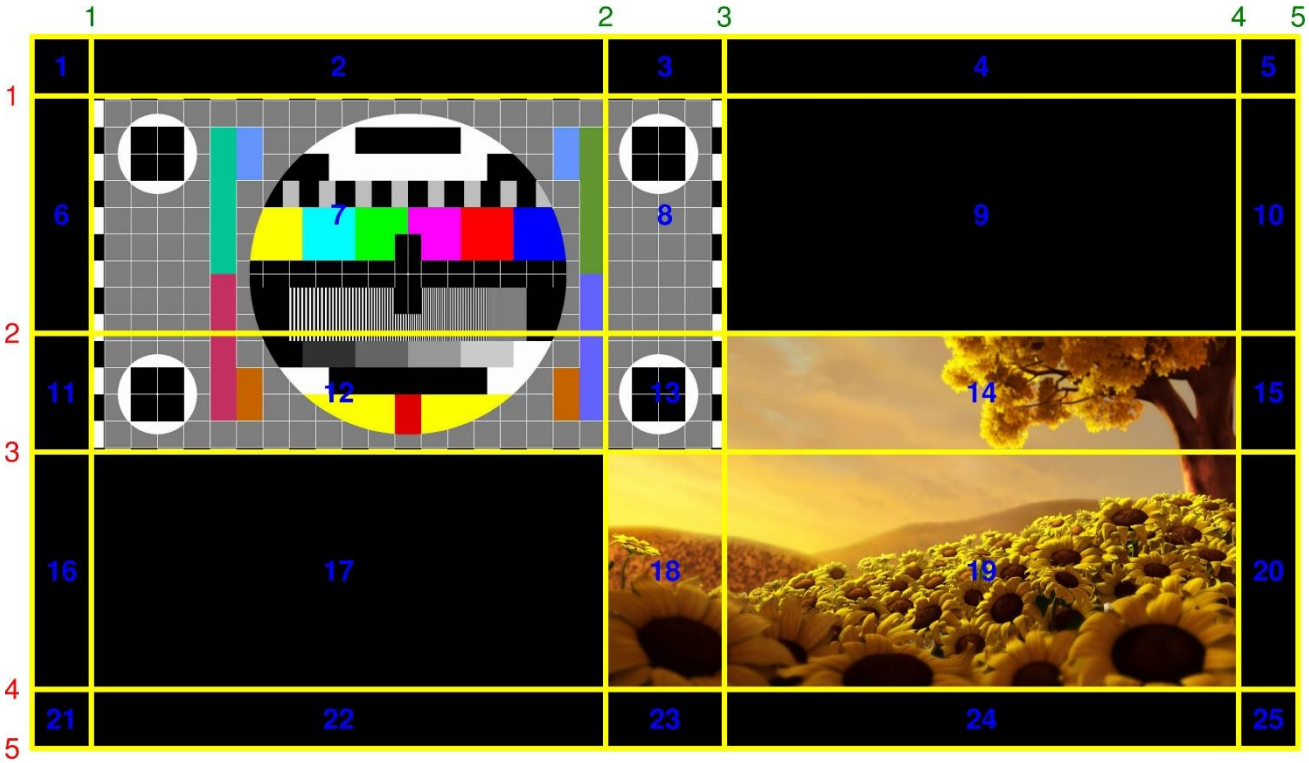
- A **layout** represents the full set of multiview configuration for a Decoder. A layout has a name that is used to refer to it by the various commands, and an output resolution. In addition, layouts defined using API commands have a set of surfaces and a set of windows.

- The following limitations apply:
    - The total number of tiles cannot exceed 240
    - Number of horizontal or vertical tile borders cannot exceed 16
    - The largest possible tile grid is either 16x15 or 15x16
    - Tile widths must be a multiply of 32 pixels
    - Surfaces have to be horizontally separated in frame buffer memory by 32 pixels margin
    - The number of distinct video sources cannot exceed 32
    - Number of windows cannot exceed 32
    - The total bandwidth sent by an individual transmitter and received by an individual receiver must never exceed 10Gbit/s and for all practical purposes should never exceed 8.1.5 Gb/s to allow room for Gigabit Ethernet, USB, Audio and control (IR and RS-232)
    - Video streams used by multiviewer must be progressive RGB 8 bits or YUV 4:4:4 bits

ToC

Appendix A - How to Multiview continued…

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



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

ToC

Appendix A - How to Multiview continued…

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

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

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

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

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

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

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

RESOLUTION	Hvisible	Htotal	Vvisable	Vtotal	color format	format	color depth	Ebpp	Line (KHz)	Frame (Hz)	BW line	Network Bandwidth
720P	1280	1648	720	750	RGB	888	24	24	45	60	1.382	**1.451**
1080i	1920	2200	1080	1125	RGB	888	24	24	33.75	30	1.555	**1.632**
1080P	1920	2200	1080	1125	RGB	888	24	24	67.5	60	3.110	**3.265**
	1920	2200	1080	1125	YUV	422	24	16	67.5	60	2.073	**2.177**
	1920	2200	1080	1125	YUV	420	24	12	67.5	60	1.555	**1.632**
UHD 24	3840	5500	2160	2250	RGB	888	24	24	67.5	24	4.976	**5.225**
UHD 30	3840	5500	2160	2250	RGB	888	24	24	67.5	30	6.220	**6.531**
	3840	5500	2160	2250	YUV	422	24	16	67.5	30	4.147	**4.354**
	3840	5500	2160	2250	YUV	420	24	12	67.5	30	3.110	**3.265**
UHD 60	3840	5500	2160	2250	RGB	888	24	24	135	60	12.441	**13.063**
	3840	5500	2160	2250	YUV	422	24	16	135	60	8.1.294	**8.709**
	3840	5500	2160	2250	YUV	420	24	12	135	60	6.220	**6.531**
	3840	5500	2160	2250	YUV	422	30	20	135	60	10.368	**10.886**
	3840	5500	2160	2250	YUV	420	30	15	135	60	7.776	**8.165**

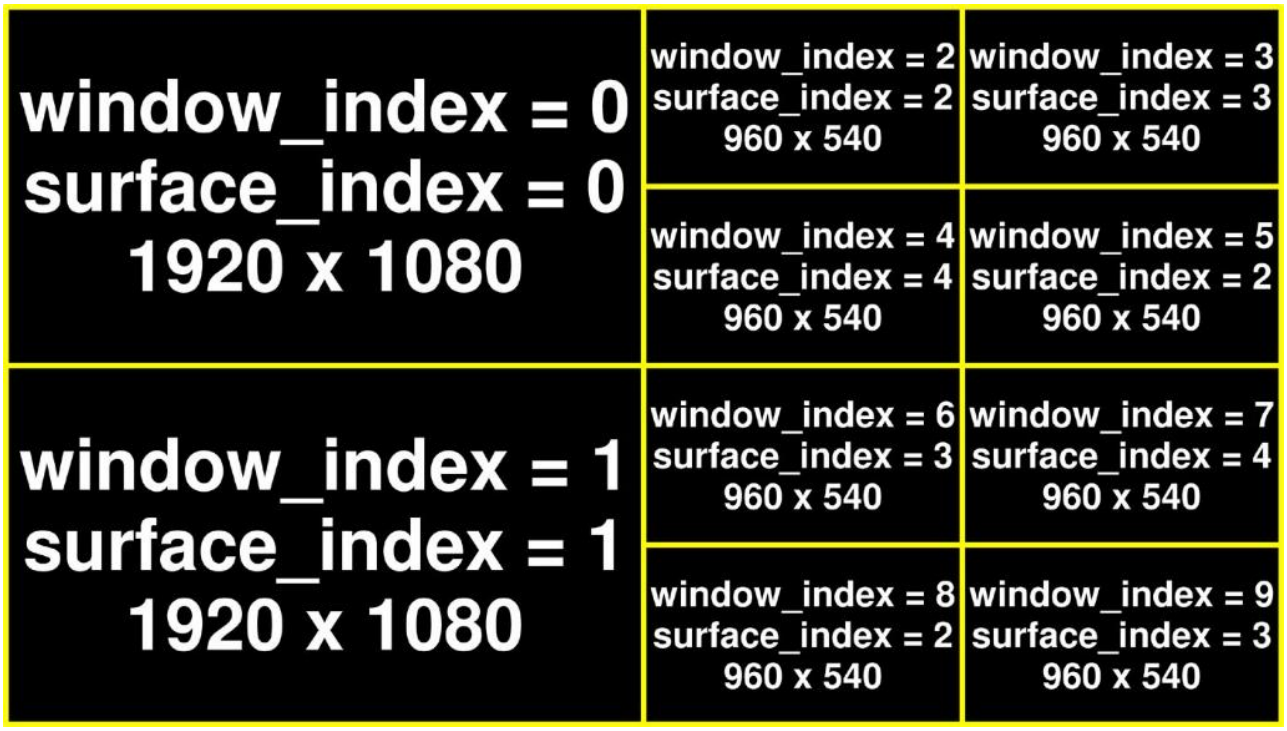
Approximate maximum network bandwidth for several common video resolutions

ToC

Appendix A - How to Multiview continued…

The table below shows available multiviewer commands:

COMMAND	DESCRIPTION	REFERENCE
set scaler	Used to set the resolution of the sub stream	8.1.6
set frame_converter	Used to half the frame rate of the Encoder video stream	8.1.5
multiview	Used once to activate the layout	11.1
layout new	Used once to define the layout name and size	11.2.1
layout window	Used multiple times to define the window size and location	11.2.2
layout black	Used to apply a black area to the layout	11.2.3
layout delete	Used to delete an existing layout	11.2.4
join multi	Used multiple times to join the Encoders to layout surfaces	4.11
leave sub	Used to leave a video stream subscription	5.2
stop video	Used to stop the main stream and reduce Encoder bandwidth	6.1
stop sub	Used to stop the multiview stream	6.2



Examples used to create the above scene:

ToC

Example 1: (recommended)

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

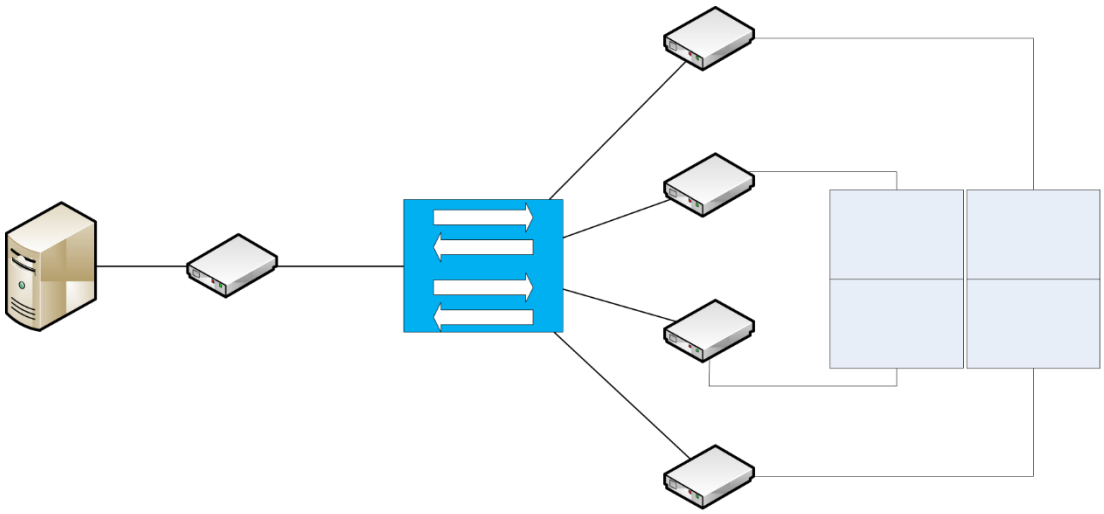
layout new *MyLayout 3840 2160*<cr>
layout window *MyLayout* 0 *0 0 1920 1080 0*<cr>
layout window *MyLayout* 1 *0 1080 1920 1080 1*<cr>
layout window *MyLayout* 2 *1920 0 960 540 2*<cr>
layout window *MyLayout* 3 *2880 0 960 540 3*<cr>
layout window *MyLayout* 4 *1920 540 960 540 4*<cr>
layout window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

Example 2:

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

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

Appendix B - How to Video wall



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

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

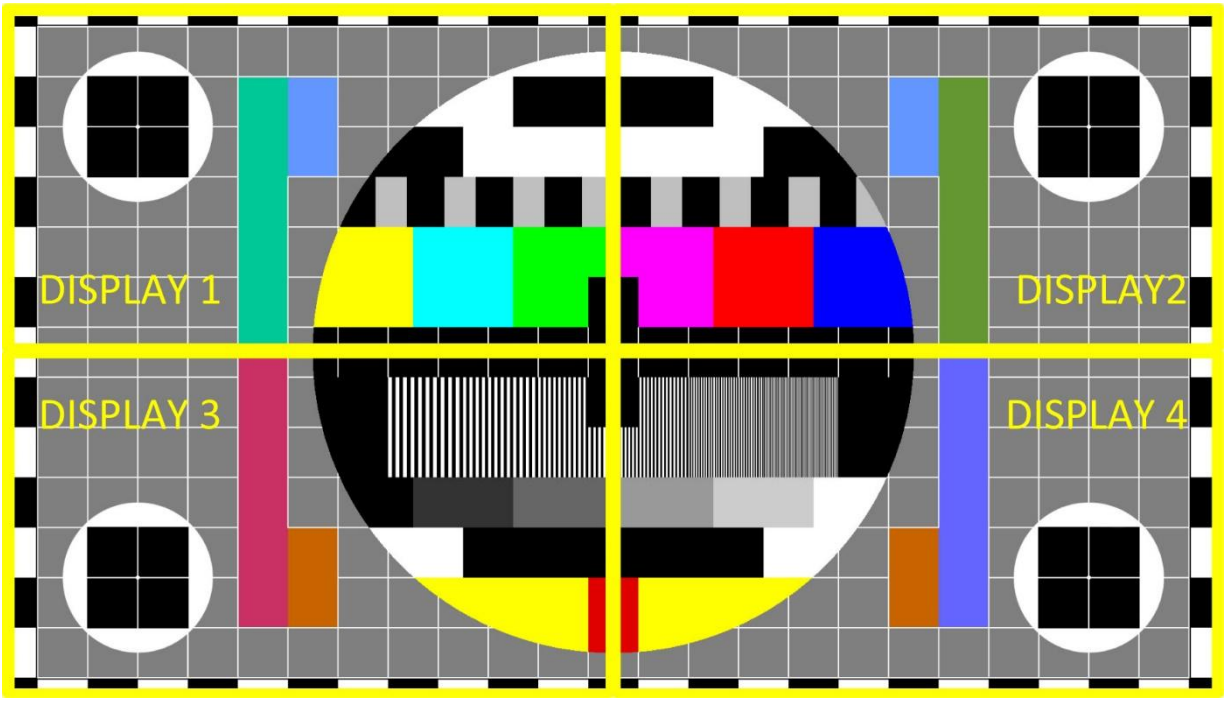
The table below shows available Video wall commands:

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

ToC

2x2 Video Wall example

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



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

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

ToC

2x2 Video Wall example continued…

DISPLAY 1

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

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

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

DISPLAY 2

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

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

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

DISPLAY 3

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

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

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

DISPLAY 4

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

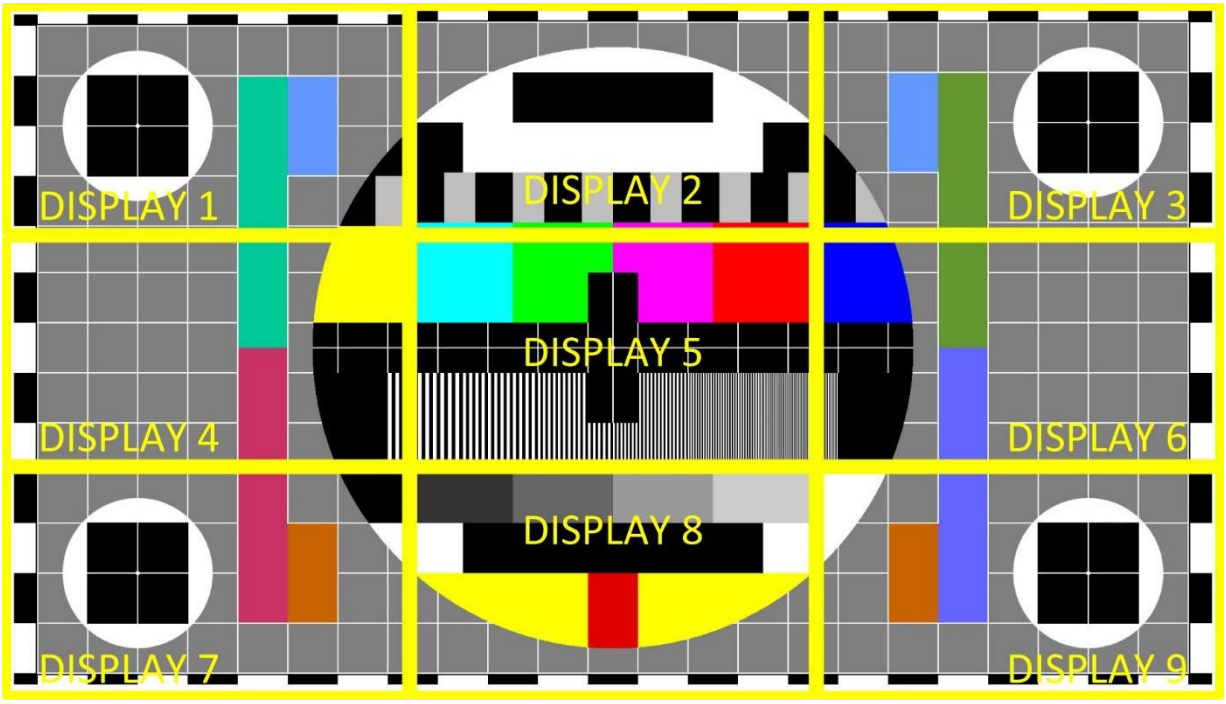
Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder4 1920 1080 992 572 928 508 0 0 1920 1080 60<cr>

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

ToC

3x3 Video Wall example

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



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

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

DISPLAY 1

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

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

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

ToC

3x3 Video Wall example continued…

DISPLAY 2

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

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

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

DISPLAY 3

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

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

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

DISPLAY 4

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

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

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

DISPLAY 5

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

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

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

ToC

3x3 Video Wall example continued…

DISPLAY 6

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

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

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

DISPLAY 7

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

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

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

DISPLAY 8

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

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

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

DISPLAY 9

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

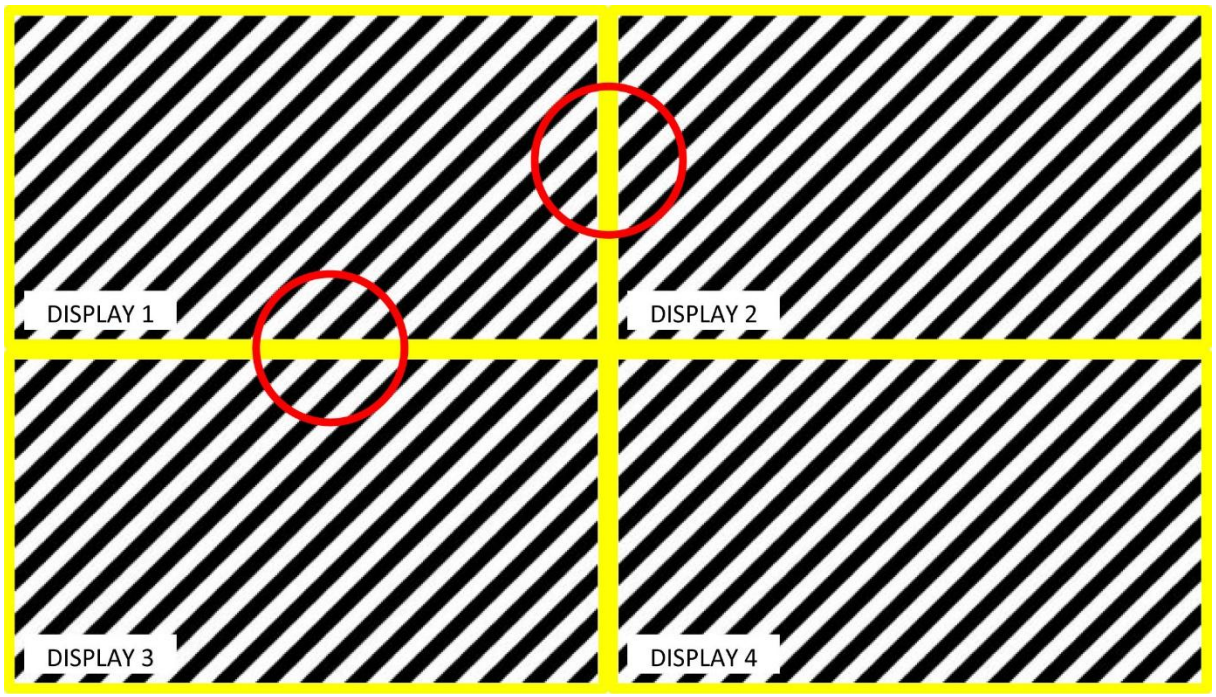
Advanced wall - calculated bezel compensation
join walladv Encoder1 Decoder9 1920 1080 2624 1504 1216 656 0 0 1920 1080 60<cr>

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

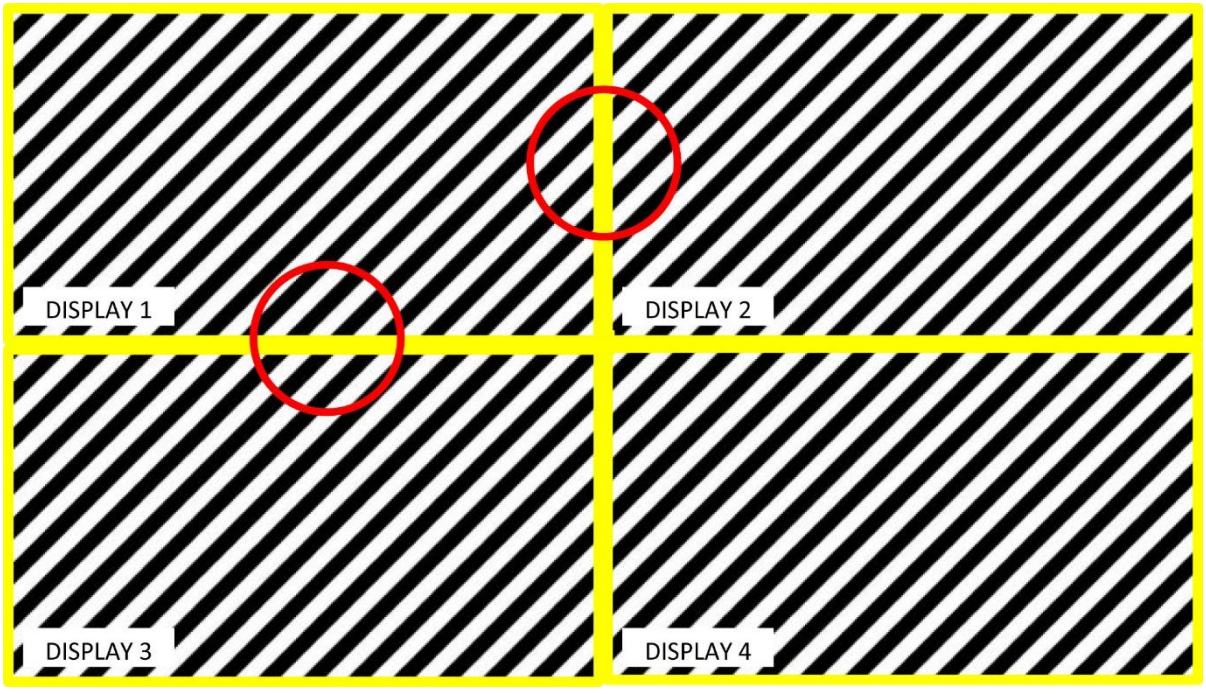
ToC

Bezel Compensation - walladv

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



without bezel compensation



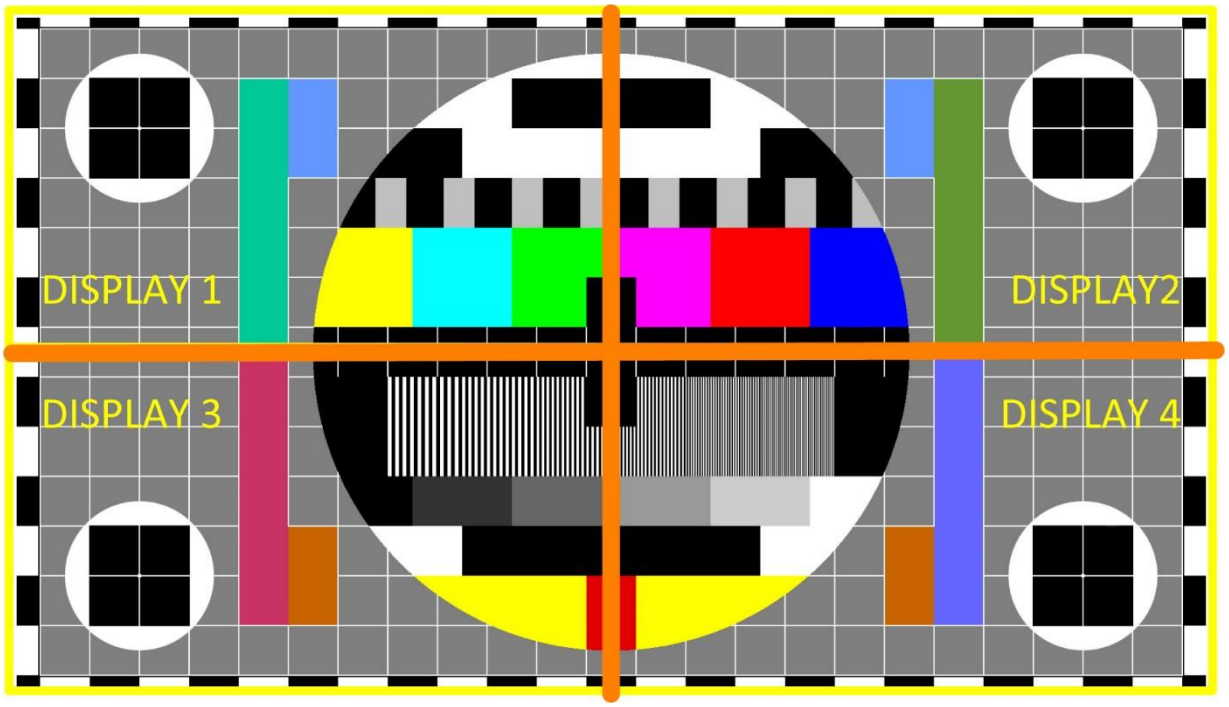
with bezel compensation

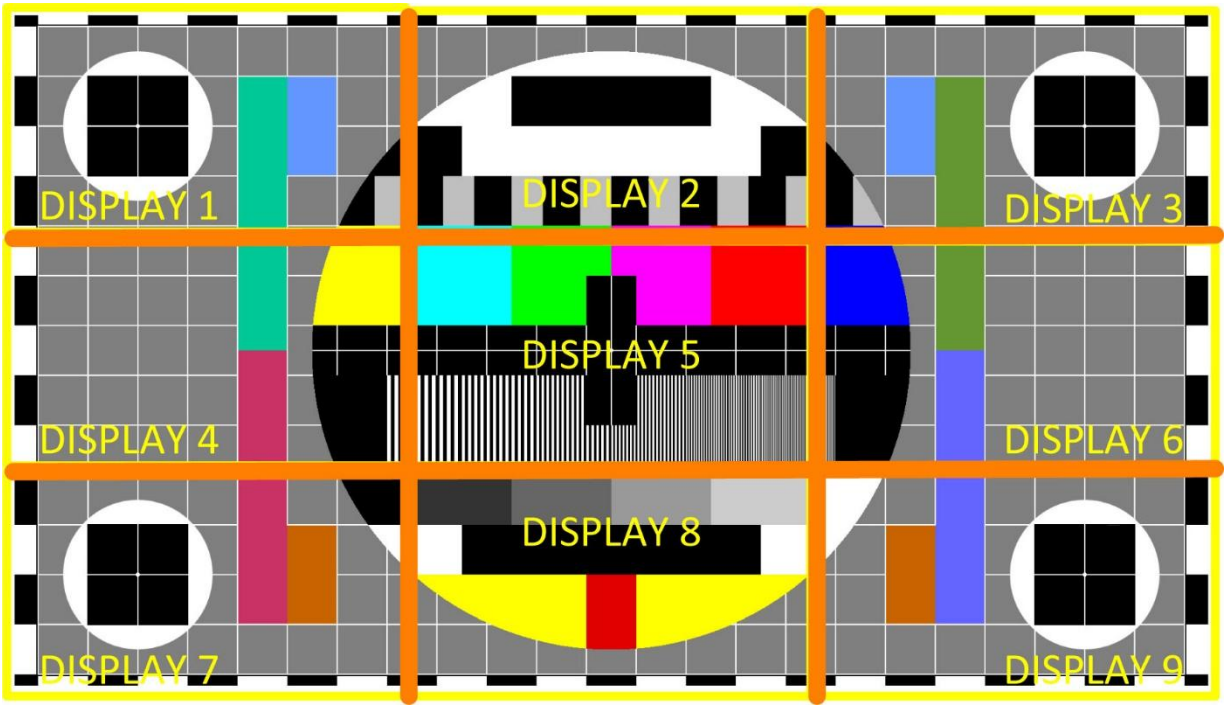
ToC

Bezel Compensation – walladv continued…

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

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





ToC

Bezel Compensation – walladv continued…

Definition of terms:

source_resolution_width = source video resolution width in pixels

source_resolution_height = source video resolution height in pixels

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

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

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

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

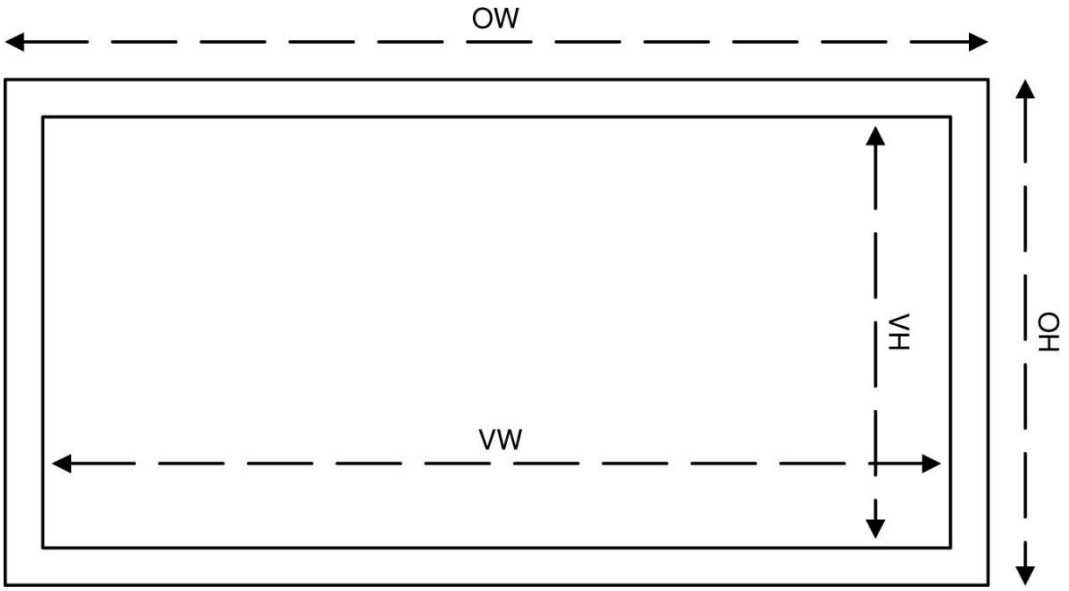
BWB = physical width of the displays bottom bezel in millimetres when using different bezel sizes

OW = physical overall width of the display in millimetres

VW = physical width of the displays viewable area in millimetres

OH = physical overall height of the display in millimetres

VH = physical height of the displays viewable area in millimetres



ToC

Bezel Compensation – walladv continued…

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

1) Uniform bezel size

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

ToC

Bezel Compensation – walladv continued…

2) Different bezel sizes

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

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

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

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

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

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

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

ToC

Bezel Compensation – walladv continued…

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

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

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

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

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

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

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

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

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

ToC

viewport - walladv

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

The **viewport_horiz** and **viewport_vert** arguments specify respectively the horizontal and vertical position of the viewports top-left corner on the screen, in pixels, with respect to the specified width.

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

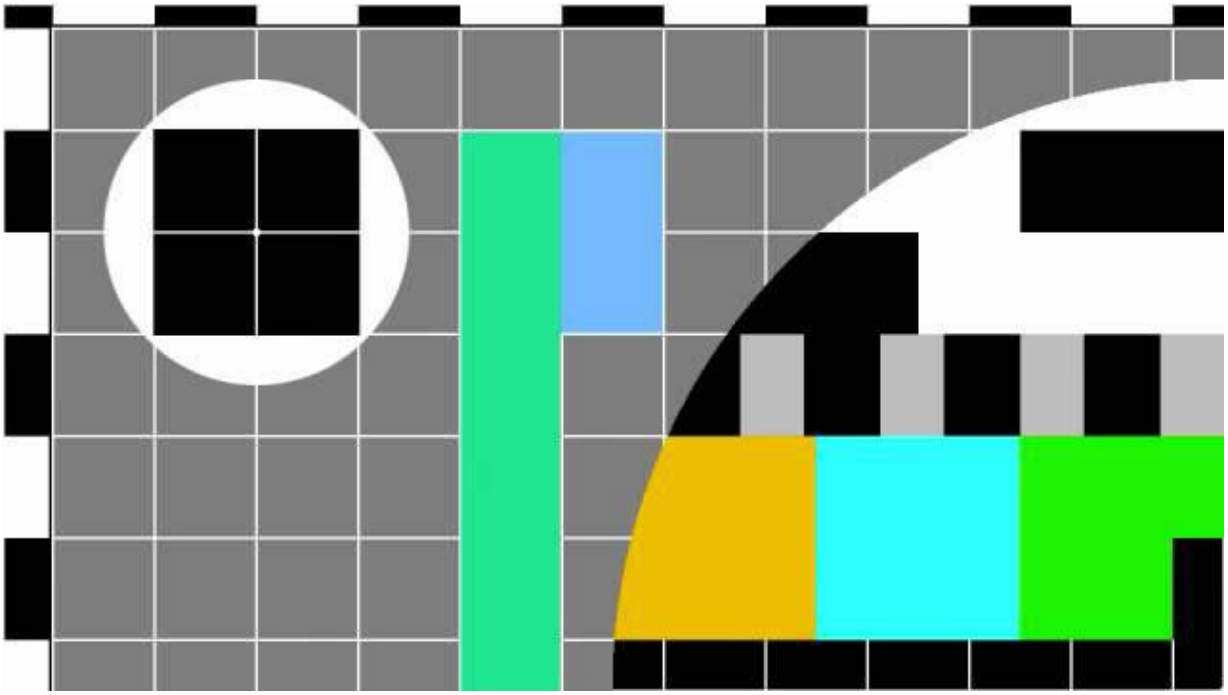
Original full screen 1920x1080 video source



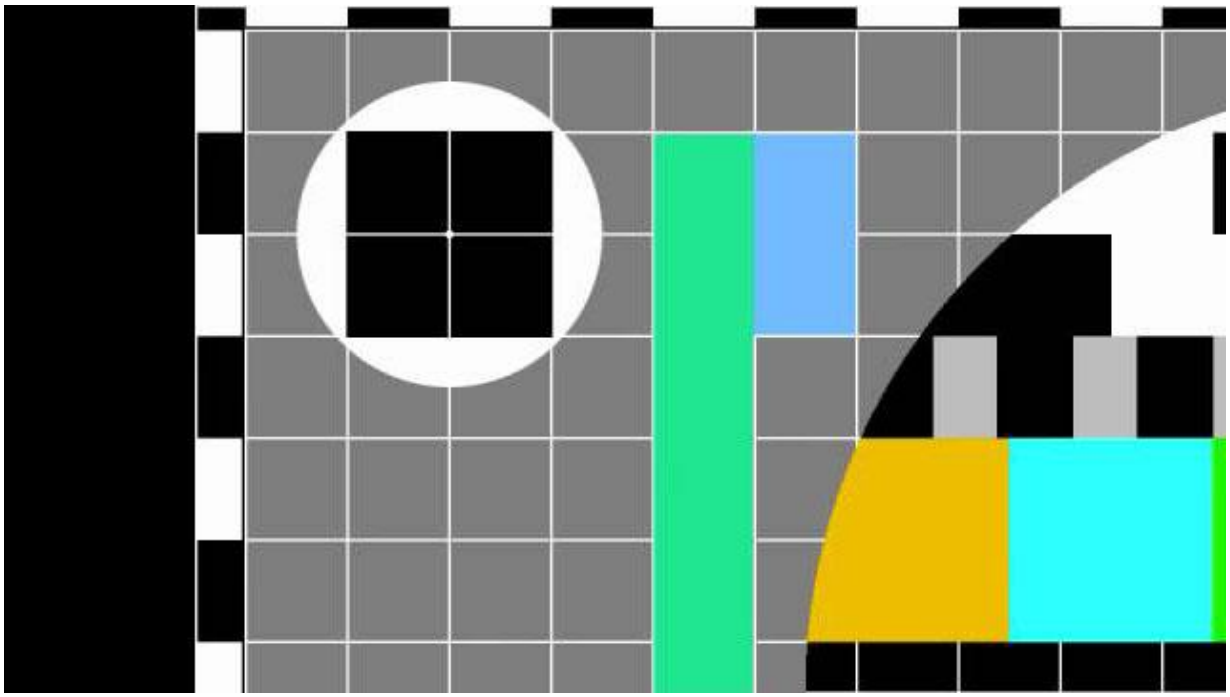
ToC

viewport – walladv continued…

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



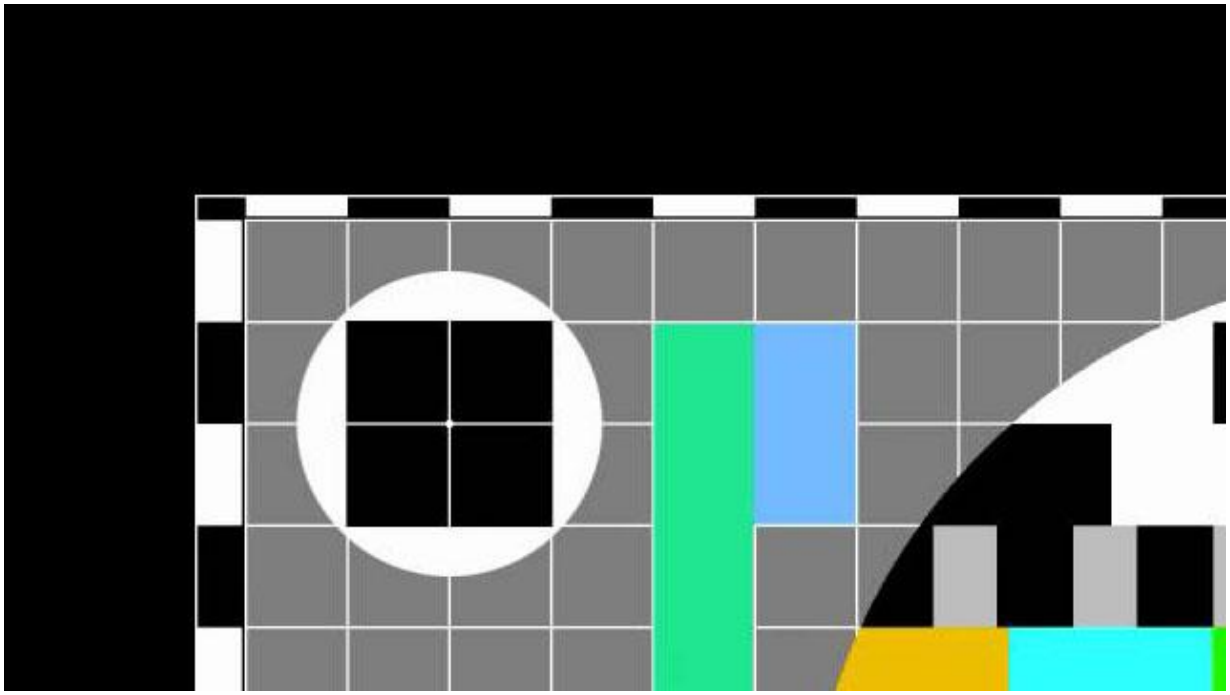
Add 100 pixels of black to the left and reduce viewport_width by 100 pixels.
join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 **100** 0 **1820** 1080 60<cr>



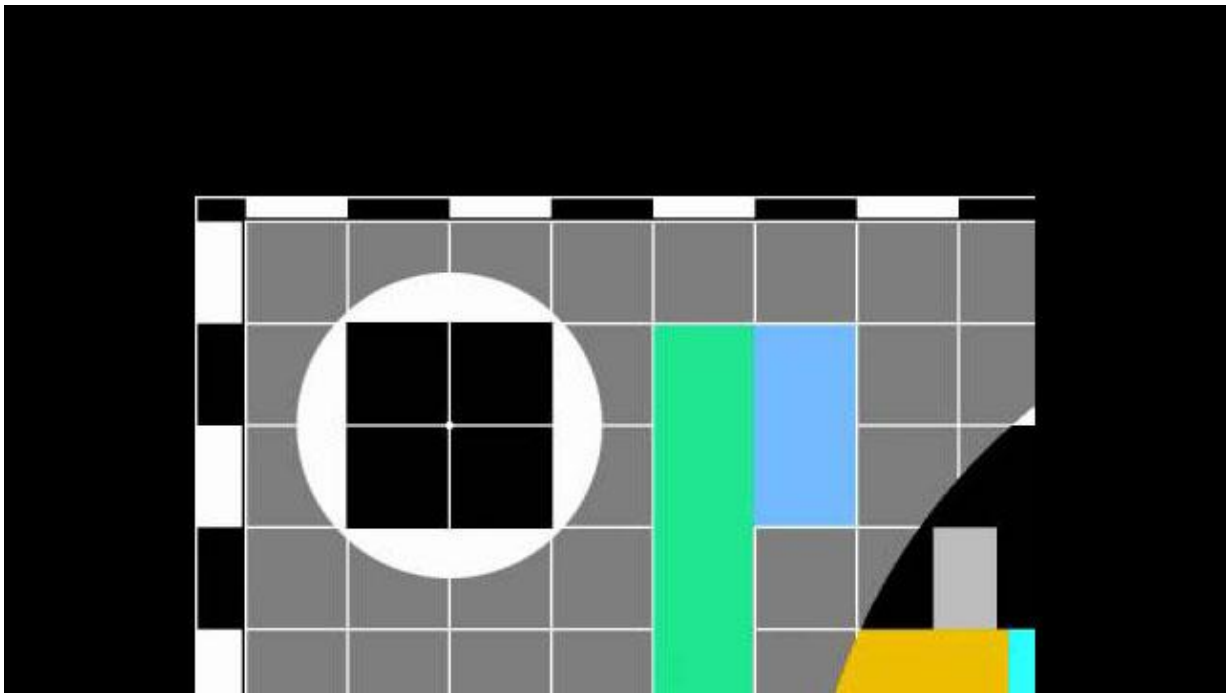
ToC

viewport – walladv continued…

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



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

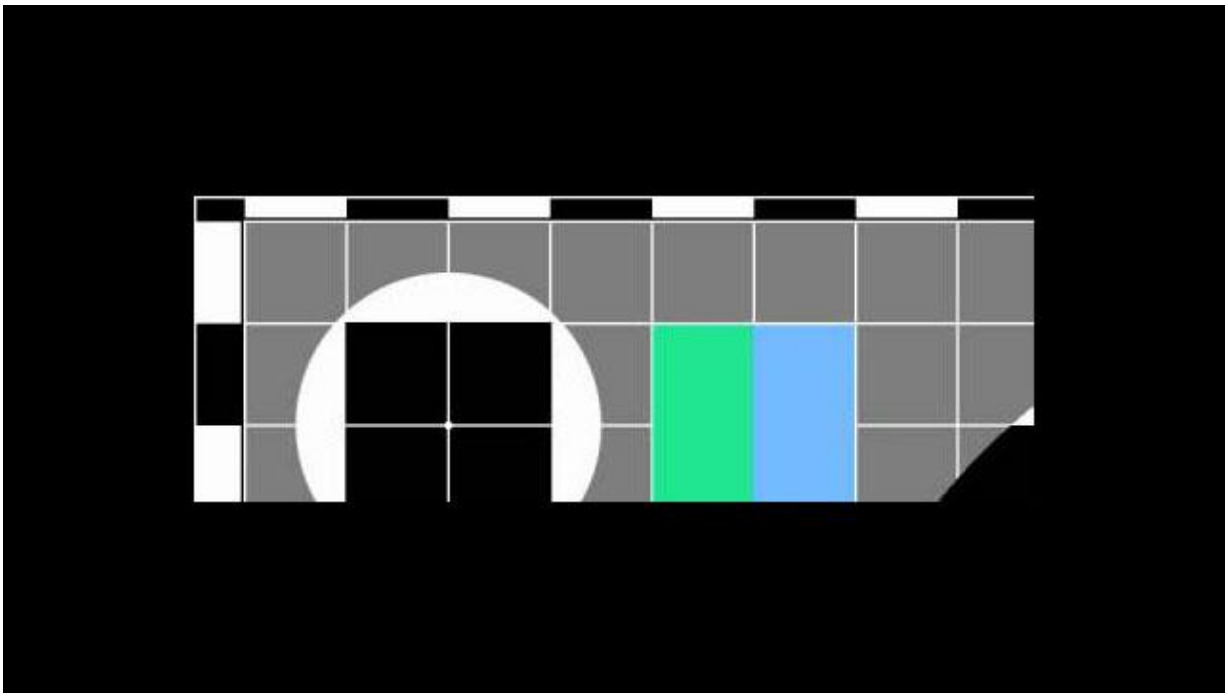


ToC

viewport – walladv continued…

Add 100 pixels of black to the bottom by reducing viewport_height by 100 pixels.

join walladv Encoder1 Decoder1 1920 1080 0 0 960 540 100 100 1720 **880** 60<cr>

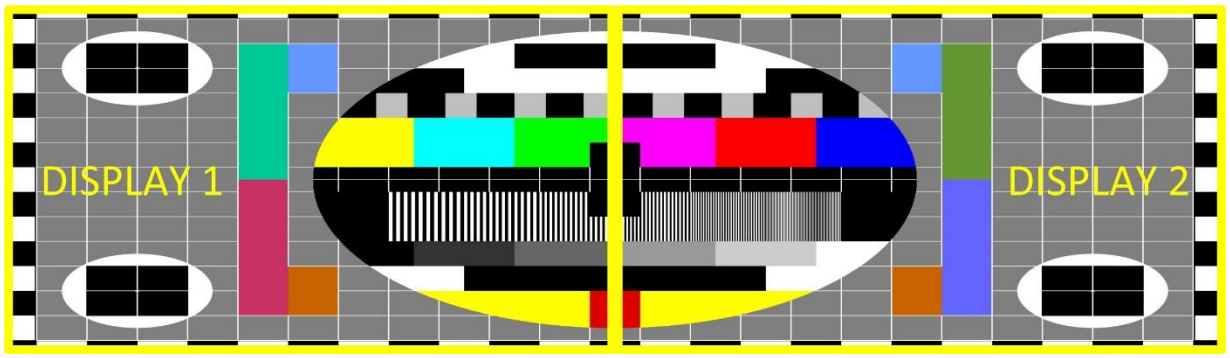


ToC

viewport – walladv continued…

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

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



DISPLAY 1

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

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

DISPLAY 2

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

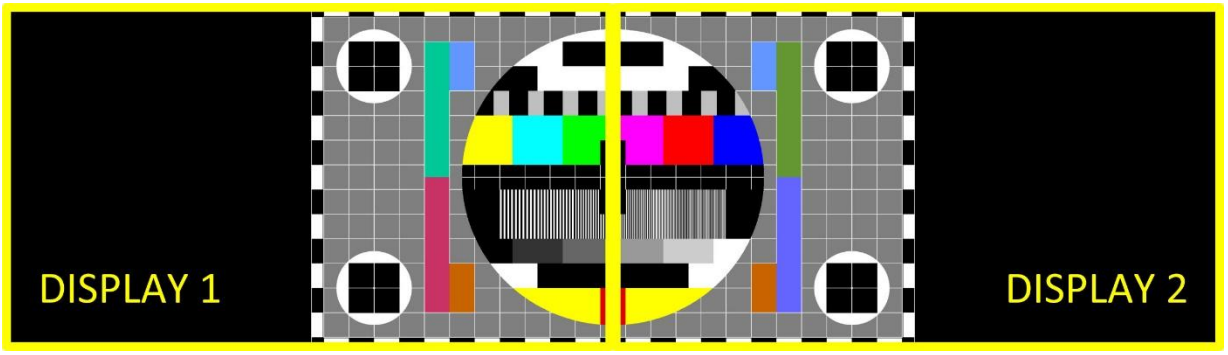
join walladv Encoder1 Decoder2 1920 1080 976 0 944 1080 0 0 1920 1080 60<cr>

ToC

viewport – walladv continued…

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

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



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

DISPLAY 1

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

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

DISPLAY 2

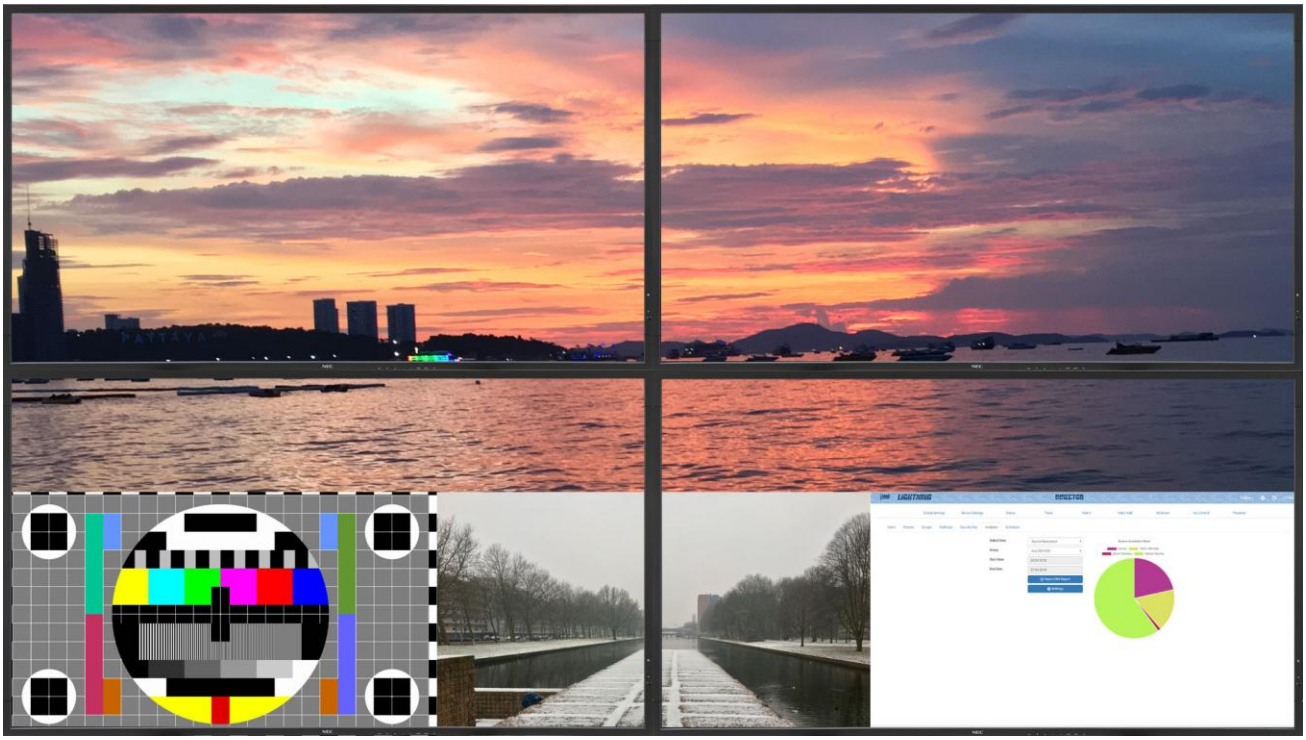
width = 1920
height = 1080
h_offset = ((horizontal resolution / number of horizontal displays) + Bezel) = ((1920 / 2) + 16) = 976
v_offset = 0
keep_width = ((horizontal resolution / number of horizontal displays) - Bezel) = ((1920 / 2) - 16) = 944
keep_height = 1080
viewport_horz = 0
viewport_vert =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

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

ToC

Appendix C - How to Video Wall with Multiview

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



ToC

Appendix C - How to Video Wall with Multiview continued…

1. Firstly a dedicated Decoder is required to create a HDMI video signal containing the Multiview Layout. Let's name this Decoder '*MVsourceOUT*'. The HDMI output of this Decoder is connected to a dedicated Encoder's HDMI input which will then convert the Multiview layout video into a stream accessible on the network for any other Decoder to display. Let's name this Encoder '*MVsourceIN*'.

Decoder "*MVsourceOUT*"



2. Now create a Video Wall preset using the required display layout. For this example a 2x2 Video Wall is used and the preset is saved as '*sample_VideoWall_2x2*'.

ToC

Appendix C - How to Video Wall with Multiview continued…

3. Now create a Multiview layout as required using Decoder '*MVsourceOUT*'. For this example layout #6 is used and the preset is saved as '*sample_Multiview_6*'.

4. Now these two presets just need to be executed. Whenever a change of the layout is required, only a different Multiview layout preset is required to be executed as the Video Wall configuration will remain the same.

ToC

Appendix D - How to HTTP request

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

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

Example 5: IFTTT – Webhooks – GET

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

Method
GET

Content Type
text/plain

Example 6: bt.tn – Auxiliary HTTP request GET

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

ToC

Appendix E – Preset Logic

Basic if else logic can be applied within a preset to allow you to build some *smarts* into your system. All 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…

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]
```

Appendix 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

Appendix 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")
```

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
}
```

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:

ToC

- 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
}
```

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") {
```

ToC

```
        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
0000006700000015006000170017001700 2F001700170017002F0017002F0017002F001700170017001700170017001700 2F001700170017002F0
017002F001700170017002F0017002F0017002F0017002F0017002F0204\x0D" contains "send ir success"

} elseif (<<button_name>> == "Btn12") {
    send tcp (get var myIP6980) 6980 "send ir encoder3
0000006700000015006000170017001700 2F001700170017002F0017002F0017002F001700170017001700170017001700 2F001700170017002F0
017002F001700170017002F0017002F0017002F0017002F0017002F0204\x0D" contains "send ir success"

} elseif (<<button_name>> == "Btn13") {
    send tcp (get var myIP6980) 6980 "send ir encoder3
0000006700000015006000170017001700 2F001700170017002F0017002F0017002F001700170017001700170017001700 2F001700170017002F0
017002F001700170017002F0017002F0017002F0017002F0017002F021A\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

Appendix 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

仕様は改良のため予告なく変更することがありますので、あらかじめご了承ください。

カナレ電気株式会社
https://www.canare.co.jp/
Copyright© Canare Electric Co., Ltd. All rights reserved

MBW319