```
### Script Generated by Control Surface Studio for Python 2.7 (resorted to default: no)
from __future__ import division
import Live
from _Framework.ControlSurface import ControlSurface
from _Framework.Layer import Layer
from _Framework.DeviceComponent import DeviceComponent
from _Framework.MixerComponent import MixerComponent
from _Framework.SliderElement import SliderElement
from _Framework.TransportComponent import TransportComponent
from _Framework.InputControlElement import *
from _Framework.ButtonElement import ButtonElement
from _Framework.ButtonMatrixElement import ButtonMatrixElement
from _Framework.SessionComponent import SessionComponent
from _Framework.EncoderElement import *
from Launchpad.ConfigurableButtonElement import ConfigurableButtonElement
import time
from itertools import imap, chain
from _Framework.Util import find_if
import collections
try:
    from user import *
except ImportError:
    pass
class css_atcoperator_imported_1(ControlSurface):
    def __init__(self, c_instance):
        super(css_atcoperator_imported_1, self).__init__(c_instance)
        with self.component_guard():
            global _map_modes
            _map_modes = Live.MidiMap.MapMode
            self.current_track_offset = 0
            self.current_scene_offset = 0
            global mixer
            num_tracks = 128
            num_returns = 24
            if hasattr(self, 'modifierList'):
                self.modifierList()
            if hasattr(self, 'customLists'):
                self.customLists()
            self._settings()
            self._inputs()
            self.turn_inputs_off()
            self.mixer = MixerComponent(num_tracks, num_returns)
            global active_mode
            self.debug_on = False
            self.mode_list()
            self.set_active_mode(self.modes[0])
            self.listening_to_tracks()
```

```
48              self.song().add_tracks_listener(self.listening_to_tracks)
49              self.song().add_tracks_listener(self._on_tracks_changed)
50              self.song().add_scenes_listener(self._on_scenes_changed)
51              self.all_track_device_listeners()
52
… self.song().view.add_selected_parameter_listener(self.
… _on_selected_parameter_changed)
53              self.create_clip_slot_map()
54              try:
55                  self.user = user(self)
56              except:
57                  pass
58              self.call_script_reaction(None, None,
… 'script_was_initialised')
59      def modifierList(self):
60          global modifiers
61          self.modifiers = {}
62          self.modifiers["m1"] = {"value": 0}
63          self.modifiers["m2"] = {"value": 0}
64          self.modifiers["m3"] = {"value": 0}
65          self.modifiers["m4"] = {"value": 0}
66          self.modifiers["m5"] = {"value": 0}
67          self.modifiers["m6"] = {"value": 0}
68          self.modifiers["m7"] = {"value": 0}
69          self.modifiers["m8"] = {"value": 0}
70          self.modifiers["m9"] = {"value": 0}
71          self.modifiers["m10"] = {"value": 0}
72          self.modifiers["m11"] = {"value": 0}
73          self.modifiers["m12"] = {"value": 0}
74          self.modifiers["m13"] = {"value": 0}
75          self.modifiers["m14"] = {"value": 0}
76          self.modifiers["m15"] = {"value": 0}
77          self.modifiers["m16"] = {"value": 0}
78          self.modifiers["m17"] = {"value": 0}
79          self.modifiers["m18"] = {"value": 0}
80          self.modifiers["m19"] = {"value": 0}
81          self.modifiers["m20"] = {"value": 0}
82      def customLists(self):
83          global lists
84          self.lists = {}
85          self.lists["list1"] = {"value": []}
86          self.lists["list2"] = {"value": []}
87          self.lists["list3"] = {"value": []}
88          self.lists["list4"] = {"value": []}
89          self.lists["list5"] = {"value": []}
90          self.lists["list6"] = {"value": []}
91          self.lists["list7"] = {"value": []}
92          self.lists["list8"] = {"value": []}
```

```
93          self.lists["list9"] = {"value": []}
94          self.lists["list10"] = {"value": []}
95      def _settings(self):
96          self.global_feedback = "default"
97          self.global_feedback_active = True
98          self.global_LED_on = 127
99          self.global_LED_off = 0
100         self.controller_LED_on = 127
101         self.controller_LED_off = 0
102         self.led_on = self.controller_LED_on
103         self.led_off = self.controller_LED_off
104     def mode_list(self):
105         global modes
106         self.mode_conf = 8367
107         self.modes = {}
108         self.modes[0] = "1"
109     def _inputs(self):
110         self.input_map = [
111             "midi_note_ch_15_val_1",
112             "midi_note_ch_15_val_2",
113             "midi_note_ch_15_val_3",
114             "midi_note_ch_15_val_4"]
115         self.midi_note_ch_15_val_1 = ConfigurableButtonElement(True,
… MIDI_NOTE_TYPE, 15, 1)
116         self.midi_note_ch_15_val_1.set_on_off_values(self.led_on,
… self.led_off)
117
… self.midi_note_ch_15_val_1.add_value_listener(self.placehold_listener,
… identify_sender= False)
118         self.midi_note_ch_15_val_1.pre_val = 0
119         self.midi_note_ch_15_val_1.cur_val = 0
120         self.midi_note_ch_15_val_2 = ConfigurableButtonElement(True,
… MIDI_NOTE_TYPE, 15, 2)
121         self.midi_note_ch_15_val_2.set_on_off_values(self.led_on,
… self.led_off)
122
… self.midi_note_ch_15_val_2.add_value_listener(self.placehold_listener,
… identify_sender= False)
123         self.midi_note_ch_15_val_2.pre_val = 0
124         self.midi_note_ch_15_val_2.cur_val = 0
125         self.midi_note_ch_15_val_3 = ConfigurableButtonElement(True,
… MIDI_NOTE_TYPE, 15, 3)
126         self.midi_note_ch_15_val_3.set_on_off_values(self.led_on,
… self.led_off)
127
… self.midi_note_ch_15_val_3.add_value_listener(self.placehold_listener,
… identify_sender= False)
128         self.midi_note_ch_15_val_3.pre_val = 0
```

```
129         self.midi_note_ch_15_val_3.cur_val = 0
130         self.midi_note_ch_15_val_4 = ConfigurableButtonElement(True,
…   MIDI_NOTE_TYPE, 15, 4)
131         self.midi_note_ch_15_val_4.set_on_off_values(self.led_on,
…   self.led_off)
132 
…   self.midi_note_ch_15_val_4.add_value_listener(self.placehold_listener,
…   identify_sender= False)
133         self.midi_note_ch_15_val_4.pre_val = 0
134         self.midi_note_ch_15_val_4.cur_val = 0
135     def _mode1(self):
136         self.show_message("Mode 1 is active")
137         num_tracks = 128
138         num_scenes = 1
139         track_offset = self.current_track_offset
140         scene_offset = self.current_scene_offset
141         combination_mode = "off"
142         feedbackArr = {}
143         feedbackArr["ClipRecording"] = None
144         feedbackArr["ClipStarted"] = None
145         feedbackArr["ClipStopped"] = None
146         feedbackArr["ClipTriggeredPlay"] = None
147         feedbackArr["ClipTriggeredRecord"] = None
148         feedbackArr["NoScene"] = None
149         feedbackArr["RecordButton"] = None
150         feedbackArr["Scene"] = None
151         feedbackArr["SceneTriggered"] = None
152         feedbackArr["StopAllOff"] = None
153         feedbackArr["StopAllOn"] = None
154         feedbackArr["StopClip"] = None
155         feedbackArr["StopClipTriggered"] = None
156         feedbackArr["StopTrackPlaying"] = None
157         feedbackArr["StopTrackStopped"] = None
158         clips = []
159         stop_all = self.midi_note_ch_15_val_1
160         stop_tracks = []
161         scene_launch = [self.midi_note_ch_15_val_2]
162         self.session_box(num_tracks, num_scenes, track_offset,
…   scene_offset, clips, stop_all, stop_tracks, scene_launch, feedbackArr,
…   combination_mode)
163 
…   self.midi_note_ch_15_val_4.add_value_listener(self.
…   midi_note_ch_15_val_4_mode1_listener,identify_sender= False)
164 
…   self.midi_note_ch_15_val_3.add_value_listener(self.
…   midi_note_ch_15_val_3_mode1_listener,identify_sender= False)
165 
…   self.midi_note_ch_15_val_2.add_value_listener(self.
```

```
midi_note_ch_15_val_2_mode1_listener,identify_sender= False)
        self._mode1_configs()
        self._mode1_led_listeners()
    def _remove_mode1(self):
        self.show_message("Mode 1 is removed")
        self.turn_inputs_off()
        combination_mode = "off"
        self.remove_session_box(combination_mode)

self.midi_note_ch_15_val_4.remove_value_listener(self.
midi_note_ch_15_val_4_mode1_listener)

self.midi_note_ch_15_val_3.remove_value_listener(self.
midi_note_ch_15_val_3_mode1_listener)

self.midi_note_ch_15_val_2.remove_value_listener(self.
midi_note_ch_15_val_2_mode1_listener)
        self._remove_mode1_led_listeners()
    def midi_note_ch_15_val_4_mode1_listener(self, value):
        self.midi_note_ch_15_val_4.cur_val = value
        if not hasattr(self.midi_note_ch_15_val_4, "pre_val"):
            self.midi_note_ch_15_val_4.pre_val = None
        if not hasattr(self.midi_note_ch_15_val_4, "prev_press_time"):
            self.midi_note_ch_15_val_4.prev_press_time = time.time()
        self.pick_brain(self.session_box_navigation_next_id_3)
        self.midi_note_ch_15_val_4.pre_val = value
        self.midi_note_ch_15_val_4.prev_press_time = time.time()
    def midi_note_ch_15_val_3_mode1_listener(self, value):
        self.midi_note_ch_15_val_3.cur_val = value
        if not hasattr(self.midi_note_ch_15_val_3, "pre_val"):
            self.midi_note_ch_15_val_3.pre_val = None
        if not hasattr(self.midi_note_ch_15_val_3, "prev_press_time"):
            self.midi_note_ch_15_val_3.prev_press_time = time.time()
        self.pick_brain(self.session_box_navigation_prev_id_7)
        self.midi_note_ch_15_val_3.pre_val = value
        self.midi_note_ch_15_val_3.prev_press_time = time.time()
    def midi_note_ch_15_val_2_mode1_listener(self, value):
        self.midi_note_ch_15_val_2.cur_val = value
        if not hasattr(self.midi_note_ch_15_val_2, "pre_val"):
            self.midi_note_ch_15_val_2.pre_val = None
        if not hasattr(self.midi_note_ch_15_val_2, "prev_press_time"):
            self.midi_note_ch_15_val_2.prev_press_time = time.time()
        self.pick_brain(self.session_box_navigation_advance_id_5)
        self.midi_note_ch_15_val_2.pre_val = value
        self.midi_note_ch_15_val_2.prev_press_time = time.time()
    def _mode1_configs(self):
        self.mode_1_configs_map = [
            "session_box_navigation_next_id_3",
```

```
207             "session_box_navigation_prev_id_7",
208             "session_box_navigation_advance_id_5"]
209         self.session_box_navigation_next_id_3 = {}
210         self.session_box_navigation_next_id_3["attached_to"] =
…   "midi_note_ch_15_val_4"
211         self.session_box_navigation_next_id_3["module"] = "self"
212         self.session_box_navigation_next_id_3["element"] =
…   "scroll_sess_offset"
213         self.session_box_navigation_next_id_3["output_type"] = "func"
214         self.session_box_navigation_next_id_3["func_arg"] = "cnfg"
215         self.session_box_navigation_next_id_3["tracks_scenes"] = "scenes"
216         self.session_box_navigation_next_id_3["ui_listener"] = "offset"
217         self.session_box_navigation_next_id_3["feedback_brain"] =
…   "feedback_sessbox_nav"
218         self.session_box_navigation_next_id_3["ctrl_type"] = "increment"
219         self.session_box_navigation_next_id_3["enc_first"] = 127
220         self.session_box_navigation_next_id_3["enc_second"] = 0
221         self.session_box_navigation_next_id_3["steps"] = 1
222         self.session_box_navigation_next_id_3["switch_type"] = "momentary"
223
…   self.session_box_navigation_next_id_3["LED_mapping_type_needs_feedback"] =
…   "1"
224         self.session_box_navigation_next_id_3["LED_feedback"] = "default"
225         self.session_box_navigation_next_id_3["LED_feedback_active"] = "1"
226         self.session_box_navigation_next_id_3["LED_on"] = "127"
227         self.session_box_navigation_next_id_3["LED_off"] = "0"
228
…   self.session_box_navigation_next_id_3["LED_send_feedback_to_selected"] =
…   ["midi_note_ch_15_val_4"]
229         self.session_box_navigation_next_id_3["json_id"] = 3
230         self.session_box_navigation_next_id_3["mapping_name"] = "Session
…   Box Navigation NEXT"
231         self.session_box_navigation_next_id_3["mapping_type"] = "Session
…   Box Navigation"
232         self.session_box_navigation_next_id_3["parent_json_id"] = 1
233         self.session_box_navigation_next_id_3["parent_name"] =
…   "mode_1_id_1"
234         self.session_box_navigation_prev_id_7 = {}
235         self.session_box_navigation_prev_id_7["attached_to"] =
…   "midi_note_ch_15_val_3"
236         self.session_box_navigation_prev_id_7["module"] = "self"
237         self.session_box_navigation_prev_id_7["element"] =
…   "scroll_sess_offset"
238         self.session_box_navigation_prev_id_7["output_type"] = "func"
239         self.session_box_navigation_prev_id_7["func_arg"] = "cnfg"
240         self.session_box_navigation_prev_id_7["tracks_scenes"] = "scenes"
241         self.session_box_navigation_prev_id_7["ui_listener"] = "offset"
242         self.session_box_navigation_prev_id_7["feedback_brain"] =
```

```
242… "feedback_sessbox_nav"
243         self.session_box_navigation_prev_id_7["ctrl_type"] = "decrement"
244         self.session_box_navigation_prev_id_7["enc_first"] = 127
245         self.session_box_navigation_prev_id_7["enc_second"] = 0
246         self.session_box_navigation_prev_id_7["steps"] = 1
247         self.session_box_navigation_prev_id_7["switch_type"] = "momentary"
248
…  self.session_box_navigation_prev_id_7["LED_mapping_type_needs_feedback"] =
…  "1"
249         self.session_box_navigation_prev_id_7["LED_feedback"] = "default"
250         self.session_box_navigation_prev_id_7["LED_feedback_active"] = "1"
251         self.session_box_navigation_prev_id_7["LED_on"] = "127"
252         self.session_box_navigation_prev_id_7["LED_off"] = "0"
253
…  self.session_box_navigation_prev_id_7["LED_send_feedback_to_selected"] =
…  ["midi_note_ch_15_val_3"]
254         self.session_box_navigation_prev_id_7["json_id"] = 7
255         self.session_box_navigation_prev_id_7["mapping_name"] = "Session
…  Box Navigation PREV"
256         self.session_box_navigation_prev_id_7["mapping_type"] = "Session
…  Box Navigation"
257         self.session_box_navigation_prev_id_7["parent_json_id"] = 1
258         self.session_box_navigation_prev_id_7["parent_name"] =
…  "mode_1_id_1"
259         self.session_box_navigation_advance_id_5 = {}
260         self.session_box_navigation_advance_id_5["attached_to"] =
…  "midi_note_ch_15_val_2"
261         self.session_box_navigation_advance_id_5["module"] = "self"
262         self.session_box_navigation_advance_id_5["element"] =
…  "scroll_sess_offset"
263         self.session_box_navigation_advance_id_5["output_type"] = "func"
264         self.session_box_navigation_advance_id_5["func_arg"] = "cnfg"
265         self.session_box_navigation_advance_id_5["tracks_scenes"] =
…  "scenes"
266         self.session_box_navigation_advance_id_5["ui_listener"] = "offset"
267         self.session_box_navigation_advance_id_5["feedback_brain"] =
…  "feedback_sessbox_nav"
268         self.session_box_navigation_advance_id_5["ctrl_type"] =
…  "increment"
269         self.session_box_navigation_advance_id_5["enc_first"] = 127
270         self.session_box_navigation_advance_id_5["enc_second"] = 0
271         self.session_box_navigation_advance_id_5["steps"] = 1
272         self.session_box_navigation_advance_id_5["switch_type"] =
…  "momentary"
273
…  self.session_box_navigation_advance_id_5["LED_mapping_type_needs_feedback"
…  ] = "1"
274         self.session_box_navigation_advance_id_5["LED_feedback"] =
```

```
"default"
        self.session_box_navigation_advance_id_5["LED_feedback_active"] =
"1"
        self.session_box_navigation_advance_id_5["LED_on"] = "127"
        self.session_box_navigation_advance_id_5["LED_off"] = "0"

self.session_box_navigation_advance_id_5["LED_send_feedback_to_selected"]
= ["midi_note_ch_15_val_2"]
        self.session_box_navigation_advance_id_5["json_id"] = 5
        self.session_box_navigation_advance_id_5["mapping_name"] =
"Session Box Navigation Advance"
        self.session_box_navigation_advance_id_5["mapping_type"] =
"Session Box Navigation"
        self.session_box_navigation_advance_id_5["parent_json_id"] = 1
        self.session_box_navigation_advance_id_5["parent_name"] =
"mode_1_id_1"
    def _mode1_led_listeners(self):
        try:
            self._mode1_fire_all_feedback()
        except:
            self.log("_mode1_led_listeners tried to call
_mode1_fire_all_feedback but it does not exist")
        try:
            self.song().add_tracks_listener(self._all_tracks_listener)
        except:
            self.log("_mode1_led_listeners tried to call
add_tracks_listener but it does not exist")
        try:
            self.all_track_device_listeners()
        except:
            self.log("_mode1_led_listeners tried to call
all_track_device_listeners but it does not exist")
        try:
            self._mode1_ui_listeners()
        except:
            self.log("_mode1_led_listeners tried to call
_mode1_ui_listeners but it does not exist")
        self.track_feedback(1)
        self.device_feedback(1)
        self.mode_device_bank_leds(1)
    def _remove_mode1_led_listeners(self):
        try:
            self.song().remove_tracks_listener(self._all_tracks_listener)
        except:
            self.log("_remove_mode1_led_listeners tried to call
remove_tracks_listener but it does not exist")
        try:
            self._remove_all_track_device_listeners()
```

```
311             except:
312                 self.log("_remove_mode1_led_listeners tried to call
…   _remove_all_track_device_listeners but it does not exist")
313             try:
314                 self._remove_mode1_ui_listeners()
315             except:
316                 self.log("_remove_mode1_led_listeners tried to call
…   _remove_mode1_ui_listeners but it does not exist")
317         def _mode1_ui_listeners(self):
318             try:
319
…   self._session.add_offset_listener(self.
…   session_box_navigation_next_id_3_led_listener)
320             except:
321                 self.log("_mode1_ui_listeners: self._session does not exist")
322             try:
323
…   self._session.add_offset_listener(self.
…   session_box_navigation_prev_id_7_led_listener)
324             except:
325                 self.log("_mode1_ui_listeners: self._session does not exist")
326             try:
327
…   self._session.add_offset_listener(self.
…   session_box_navigation_advance_id_5_led_listener)
328             except:
329                 self.log("_mode1_ui_listeners: self._session does not exist")
330         def _remove_mode1_ui_listeners(self):
331             try:
332
…   self._session.remove_offset_listener(self.
…   session_box_navigation_next_id_3_led_listener)
333             except:
334                 self.log("remove__mode1_ui_listeners: self._session does not
…   exist")
335             try:
336
…   self._session.remove_offset_listener(self.
…   session_box_navigation_prev_id_7_led_listener)
337             except:
338                 self.log("remove__mode1_ui_listeners: self._session does not
…   exist")
339             try:
340
…   self._session.remove_offset_listener(self.
…   session_box_navigation_advance_id_5_led_listener)
341             except:
342                 self.log("remove__mode1_ui_listeners: self._session does not
```

```
exist")
    def _mode1_fire_all_feedback(self):
        self.session_box_navigation_next_id_3_led_listener()
        self.session_box_navigation_prev_id_7_led_listener()
        self.session_box_navigation_advance_id_5_led_listener()
    def session_box_navigation_next_id_3_led_listener(self):
        self.feedback_brain(self.session_box_navigation_next_id_3)
    def session_box_navigation_prev_id_7_led_listener(self):
        self.feedback_brain(self.session_box_navigation_prev_id_7)
    def session_box_navigation_advance_id_5_led_listener(self):
        self.feedback_brain(self.session_box_navigation_advance_id_5)
    ################## CORE: Python 2.7 #################
    def get_value_from_ranges(self, a1, b2, c3, d4, e5, f6, g7, h8, i9,
j10, k11):
        logging = a1
        steps = b2
        round_down = c3
        current_input_value = d4
        i = {}
        i["minimum"] = e5
        i["maximum"] = f6
        i["decimal_places"] = g7
        i["steps"] = steps
        i["distance"] = i["maximum"] - i["minimum"]
        i["speed"] = i["distance"] / i["steps"]
        inn = self.step_values(i)
        o = {}
        o["minimum"] = h8
        o["maximum"] = i9
        o["decimal_places"] = j10
        o["reverse_mode"] = k11
        o["steps"] = steps
        o["distance"] = o["maximum"] - o["minimum"]
        o["speed"] = o["distance"] / o["steps"]
        out = self.step_values(o)
        closest_inn = self.f_n(inn, current_input_value, round_down)
        relative_out_value = out[closest_inn['index']]
        ret = {}
        ret['in'] = inn
        ret['selected_in'] = closest_inn
        ret['out'] = out
        ret["selected_out"] = relative_out_value
        if(logging == True):
            if(round_down == False):
                rounding = "up"
                rou_symb = str(">")
            else:
                rounding = "down"
```

```
389                 rou_symb = str("<")
390             log_arr = []
391             log_arr.append("In: " + str(current_input_value) )
392             log_arr.append("Out: " + str(ret["selected_out"]) )
393             log_arr.append("Steps: " + str(steps) )
394             log_arr.append("Rounding: " + str(rounding) )
395             log_arr.append("Rev: " + str(o["reverse_mode"]) )
396             log_str = ' '.join(log_arr)
397             table_arr = []
398             table_arr.append(str("<table class='rangeValueTable'>") )
399             step_arr = []
400             count = 0
401             for item in ret['in']:
402                 if(count==ret['selected_in']['index']):
403                     td = "<td class='slctd'>"
404                 else:
405                     td = "<td>"
406                 step_arr.append(td + str(count) + "</td>")
407                 count = count + 1
408             step_str = ''.join(step_arr)
409             in_arr = []
410             count = 0
411             for item in ret['in']:
412                 td = "<td>"
413                 if(count==ret['selected_in']['index']):
414                     td = "<td class='slctd'>"
415                 in_arr.append(td + str(item) + "</td>")
416                 count = count + 1
417             in_str = ''.join(in_arr)
418 
419             out_arr = []
420             count = 0
421             for item in ret['out']:
422                 td = "<td>"
423                 if(count==ret['selected_in']['index']):
424                     td = "<td class='slctd'>"
425                 out_arr.append(td + str(item) + "</td>")
426                 count = count + 1
427             out_str = ''.join(out_arr)
428             rev = ""
429             if o["reverse_mode"] == True:
430                 rev = "(Rev)"
431 
432             table_arr.append(str("<tr><td class='hd'>Steps</td>" +
  … step_str + "</tr>") )
433             table_arr.append(str("<tr><td class='hd'>In (" + rou_symb +
  … str(current_input_value) + ")</td>" + in_str + "</tr>") )
434             table_arr.append(str("<tr><td class='hd'>Out " + rev + "</td>"
```

```
434… + out_str + "</tr>") )
435             table_arr.append(str("</table>") )
436             table_str = ''.join(table_arr)
437             self.log_message("csslog: " + str(table_str) )
438         return ret["selected_out"]
439
440     def f_n(self, array, current_val, round_down = True):
441         i = 0
442         nearest = {}
443         nearest['index'] = None
444         nearest['value'] = None
445         prev_idx = i
446         prev_val = array[0]
447         for array_val in array:
448             if array_val == current_val:
449                 nearest['index'] = i
450                 nearest['value'] = array_val
451                 break
452             elif current_val > prev_val and current_val < array_val:
453                 if round_down is True:
454                     nearest['index'] = prev_idx
455                     nearest['value'] = prev_val
456                 else:
457                     nearest['index'] = i
458                     nearest['value'] = array_val
459                 break
460             else:
461                 prev_val = array_val
462                 prev_idx = i
463                 i = i + 1
464         return nearest;
465     def placehold_listener(self, value):
466         return
467     def pick_brain(self, obj):
468         cnfg = obj.copy()
469         if cnfg["output_type"] == "val":
470                 self.val_brain(cnfg)
471         elif cnfg["output_type"] == "func":
472             self.func_brain(cnfg)
473         elif cnfg["output_type"] == "bool":
474             self.bool_brain(cnfg)
475     def should_it_fire(self, cnfg):
476         controller = getattr(self, cnfg["attached_to"])
477         cnfg["value"] = controller.cur_val
478         cnfg["pre_val"] = controller.pre_val
479         cnfg["prev_press_time"] = controller.prev_press_time
480         timenow = time.time()
481         fire = 0;
```

```
482         if (cnfg["ctrl_type"] == "on/off" or cnfg["ctrl_type"] ==
… "increment" or cnfg["ctrl_type"] == "decrement"):
483             if(cnfg["switch_type"] == "delay"):
484                 if((cnfg["value"] == cnfg["enc_second"]) and (timenow -
… cnfg["prev_press_time"]) > cnfg["delay_amount"]):
485                     fire = 1;
486             elif(cnfg["switch_type"] == "toggle"):
487                 if cnfg["value"] == cnfg["enc_first"] or cnfg["value"] ==
… cnfg["enc_second"]:
488                     fire = 1;
489             elif (cnfg["switch_type"] == "momentary" and cnfg["value"] ==
… cnfg["enc_first"]):
490                 fire = 1;
491         elif cnfg["ctrl_type"] == "absolute":
492             if cnfg["value"] >= cnfg["enc_first"] and cnfg["value"] <=
… cnfg["enc_second"]:
493                 fire = 1;
494         elif cnfg["ctrl_type"] == "relative":
495             if cnfg["value"] == cnfg["enc_first"] or cnfg["value"] ==
… cnfg["enc_second"]:
496                 fire = 1;
497         return fire
498     def bool_brain(self, cnfg):
499         method_to_call = getattr(eval(cnfg["module"]), cnfg["element"])
500         fire = self.should_it_fire(cnfg)
501         if fire == 1:
502             if cnfg["element"] == "solo" and self.song().exclusive_solo:
503                 for index in range(len(self.song().tracks)):
504                     self.song().tracks[index].solo = False
505                 for index in range(len(self.song().return_tracks)):
506                     self.song().return_tracks[index].solo = False
507             if cnfg["element"] == "arm" and self.song().exclusive_arm:
508                 for index in range(len(self.song().tracks)):
509                     self.song().tracks[index].arm = False
510             if method_to_call is False:
511                 setattr(eval(cnfg["module"]), cnfg["element"], True)
512             else:
513                 setattr(eval(cnfg["module"]), cnfg["element"], False)
514     def func_brain(self, cnfg):
515         fire = self.should_it_fire(cnfg)
516         if fire == 1:
517             method_to_call = getattr(eval(cnfg["module"]),
… cnfg["element"])
518             if cnfg["func_arg"] != "" and cnfg["func_arg"] != "cnfg":
519                 method_to_call(cnfg["func_arg"])
520             elif cnfg["func_arg"] == "cnfg":
521                 method_to_call(cnfg)
522             else:
```

```
523                 method_to_call()
524     def val_brain(self, cnfg):
525         try:
526             cnfg["current_position"] = getattr(eval(cnfg["module"]),
…   cnfg["element"])
527         except:
528             self.show_message("This control does not exist in your
…   session")
529             return
530         self._parameter_to_map_to = eval(cnfg["module"])
531         if cnfg["ctrl_type"] != "on/off" and
…   hasattr(self._parameter_to_map_to, "max") and
…   hasattr(self._parameter_to_map_to, "min"):
532             param_range = self._parameter_to_map_to.max -
…   self._parameter_to_map_to.min
533             if cnfg.has_key("minimum"):
534                 usermin = cnfg["minimum"] / 100.;
535                 min_value = float(usermin * param_range)
536                 cnfg["minimum"] = min_value +
…   self._parameter_to_map_to.min
537             if cnfg.has_key("maximum") and cnfg["mapping_type"] !=
…   "On/Off":
538                 usermax = cnfg["maximum"] / 100.;
539                 max_value = float(usermax * param_range)
540                 cnfg["maximum"] = max_value +
…   self._parameter_to_map_to.min
541         controller = getattr(self, cnfg["attached_to"])
542         cnfg["value"] = controller.cur_val
543         cnfg["pre_val"] = controller.pre_val
544         if cnfg.has_key("decimal_places"):
545             cnfg["current_position"] = round(cnfg["current_position"],
…   cnfg["decimal_places"])
546         if cnfg["ctrl_type"] == "absolute":
547             cnfg["steps"] = (cnfg["enc_second"] - cnfg["enc_first"])
548         if cnfg["ctrl_type"] != "on/off":
549             cnfg["distance"] = cnfg["maximum"] - cnfg["minimum"]
550             cnfg["speed"] = cnfg["distance"] / cnfg["steps"]
551             cnfg["step_values"] = self.step_values(cnfg)
552             cnfg["velocity_seq"] = self._velocity_seq(cnfg)
553
554         if int(cnfg["current_position"]) < int(cnfg["minimum"]) or
…   int(cnfg["current_position"]) > int(cnfg["maximum"]):
555             new_val = self.snap_to_max_min(cnfg)
556         elif cnfg["ctrl_type"] == "absolute":
557             new_val = self.absolute_decision(cnfg)
558         elif cnfg["ctrl_type"] == "relative":
559             new_val = self.relative_decision(cnfg)
560         elif cnfg["ctrl_type"] == "on/off" or cnfg["ctrl_type"] ==
```

```
"increment" or cnfg["ctrl_type"] == "decrement":
            new_val = self.button_decision(cnfg)
        try:
            setattr(eval(cnfg["module"]), cnfg["element"], new_val)
        except:
            return
    def snap_to_max_min(self, cnfg):

        if(cnfg["enc_first"] < cnfg["enc_second"]):
            enc_lowest = cnfg["enc_first"];
            enc_highest = cnfg["enc_second"]
        else:
            enc_lowest = cnfg["enc_second"];
            enc_highest = cnfg["enc_first"]

        if cnfg["snap_to"] == True and (cnfg["value"] <= enc_lowest or
cnfg["value"] >= enc_highest):
            if int(cnfg["current_position"]) < int(cnfg["minimum"]):
                new_val = cnfg["minimum"]
                self.log("snapped to min")
            elif int(cnfg["current_position"]) > int(cnfg["maximum"]):
                new_val = cnfg["maximum"]
                self.log("snapped to max")
        else:
            new_val = cnfg["current_position"]
            self.show_message("remotify: snapping is off for this control.
Check min / max values")
        return new_val
    def step_values(self, cnfg):
        calc = []
        for i in range(0, cnfg["steps"] +1):
            val = (i * cnfg["speed"]) + cnfg["minimum"]
            if cnfg.has_key("decimal_places"):
                val = round(val, cnfg["decimal_places"])
                if cnfg["decimal_places"] is 0:
                    val = int(val)
            calc.append(val)
        if "reverse_mode" in cnfg and cnfg["reverse_mode"] is True:
            calc = list(reversed(calc))
        return calc
    def relative_decision(self, cnfg):
        fire = 0
        new_val = cnfg["current_position"]
        if cnfg["value"] == cnfg["enc_second"]:
            max_min = "max"
            fire = 1
        elif cnfg["value"] == cnfg["enc_first"]:
            max_min = "min"
```

```
606             fire = 1
607         if fire == 0:
608             return new_val
609         if cnfg["current_position"] in cnfg["step_values"]:
610             current_pos_index =
… cnfg["step_values"].index(cnfg["current_position"])
611 
612             feedback = current_pos_index / cnfg["steps"] * 127
613             feedback = round(feedback, 0)
614             method_to_call = getattr(self, cnfg["attached_to"])
615             incr_index = current_pos_index + 1
616             decr_index = current_pos_index - 1
617             if max_min == "max" and incr_index < len(cnfg["step_values"]):
618                 incr = cnfg["step_values"][incr_index]
619                 while incr == cnfg["current_position"]:
620                     incr_index = incr_index + 1
621                     if incr_index < len(cnfg["step_values"]):
622                         incr = cnfg["step_values"][incr_index]
623                     else:
624                         break
625                 new_val = incr
626             elif max_min == "min" and decr_index >= 0:
627                 decr = cnfg["step_values"][decr_index]
628                 new_val = decr
629             return new_val
630         else:
631             new_val = self.step_in_line(cnfg, max_min)
632             return new_val
633         return new_val
634     def percent_as_value(self, param, percentage):
635         param =         eval(param)
636         if hasattr(param, 'max') and hasattr(param, 'min'):
637             param_range = param.max - param.min
638             val = percentage * param_range / 100
639             return val
640         else:
641             self.log("param does not have min and/or max attribute(s)")
642     def button_decision(self, cnfg):
643         new_val = cnfg["current_position"]
644         fire = self.should_it_fire(cnfg)
645         if fire == 0:
646             return new_val;
647         if cnfg["ctrl_type"] == "on/off":
648             if(cnfg["switch_type"] == "toggle"):
649                 if cnfg["value"] == cnfg["enc_first"]:
650                     new_val = cnfg["maximum"]
651                     return new_val
652                 elif cnfg["value"] == cnfg["enc_second"]:
```

```
653                     new_val = cnfg["minimum"]
654                     return new_val
655             elif(cnfg["switch_type"] == "momentary"):
656                 if(cnfg["current_position"] == cnfg["maximum"]):
657                     new_val = cnfg["minimum"]
658                 else:
659                     new_val = cnfg["maximum"]
660                 return new_val
661             elif(cnfg["switch_type"] == "delay"):
662                 if(cnfg["current_position"] == cnfg["maximum"]):
663                     new_val = cnfg["minimum"]
664                 elif (cnfg["current_position"] == cnfg["minimum"]):
665                     new_val = cnfg["maximum"]
666                 return new_val
667             else:
668                 self.log("neither momentary or toggle were set for on off
… button")
669                 return new_val
670         if cnfg["current_position"] in cnfg["step_values"]:
671             current_pos_index =
… cnfg["step_values"].index(cnfg["current_position"])
672             incr_index = current_pos_index + 1
673             decr_index = current_pos_index - 1
674             if cnfg["ctrl_type"] ==  "increment" and incr_index <
… len(cnfg["step_values"]):
675                 incr = cnfg["step_values"][incr_index]
676                 new_val = incr
677             elif cnfg["ctrl_type"] == "decrement" and decr_index >= 0:
678                 decr = cnfg["step_values"][decr_index]
679                 new_val = decr
680             return new_val
681         else:
682             if cnfg["ctrl_type"] ==  "increment":
683                 max_min = "max"
684             elif cnfg["ctrl_type"] == "decrement": max_min = "min"
685             new_val = self.step_in_line(cnfg, max_min)
686             return new_val
687         return new_val
688     def step_in_line(self, cnfg, max_min):
689         previous = int()
690         step_num = 0
691         speed = 0
692         for step_val in cnfg["step_values"]:
693             step_num += 1
694             if cnfg["current_position"] > previous and
… cnfg["current_position"] < step_val:
695                 if max_min == "min":
696                     speed = cnfg["current_position"] - previous
```

```
                    new_val = previous
                elif max_min == "max":
                    speed = step_val - cnfg["current_position"]
                    new_val = step_val
                break
            previous = step_val
        return new_val
    def absolute_decision(self, cnfg):
        if(cnfg["enc_first"] > cnfg["enc_second"]):
            self.log("enc_first is higher than enc_second, needs to be
… lower")
        new_val = cnfg["current_position"]
        if cnfg["pre_val"] is None:
            return new_val
        ######### Get pre_val details from list values #########
        ######### ######### ######### ######## ######
        if cnfg["pre_val"] in cnfg["velocity_seq"]:
            cnfg["previous_step_num"] =
… cnfg["velocity_seq"].index(cnfg["pre_val"])
            cnfg["previous_step_value"] =
… cnfg["step_values"][cnfg["previous_step_num"]]
        else:
            cnfg["previous_step_value"] = None
        ######### get value details from list #########
        ######### ######### ######### ######### ######
        if cnfg["value"] in cnfg["velocity_seq"]:
            cnfg["step_num"] = cnfg["velocity_seq"].index(cnfg["value"])
            cnfg["step_value"] = cnfg["step_values"][cnfg["step_num"]]
        else:
            cnfg["step_num"] = None
            cnfg["step_value"] = None

        ######### MAX OR MIN ########
        ######### ######### #########
        if cnfg["reverse_mode"] is False:
            if cnfg["value"] > cnfg["pre_val"]: max_min = "max"
            elif cnfg["value"] < cnfg["pre_val"]: max_min = "min"
        elif cnfg["reverse_mode"] is True:
            if cnfg["value"] > cnfg["pre_val"]: max_min = "min"
            elif cnfg["value"] < cnfg["pre_val"]: max_min = "max"
        inside_outside = self.inside_outside_checks(cnfg)
        if inside_outside is not False:
            self.log("inside outside was not false")
            return inside_outside
        ######### straight assign or takeover #########
        ######### ######### ######### ######### #######
        if cnfg["previous_step_value"] == cnfg["current_position"]:
            new_val = cnfg["step_value"]
```

```
742         elif cnfg["takeover_mode"] == "None":
743             new_val = cnfg["step_value"]
744         elif cnfg["takeover_mode"] == "Pickup":
745             new_val = self.pickup(cnfg, max_min)
746         elif cnfg["takeover_mode"] == "Value scaling": new_val =
…   self.value_scaling(cnfg, max_min)
747         else: self.log("nothing got decided")
748 
749         return new_val
750     def inside_outside_checks(self, cnfg):
751         new_val = cnfg["current_position"]
752         if cnfg["reverse_mode"] is False:
753             minimum = cnfg["minimum"]
754             maximum = cnfg["maximum"]
755         elif cnfg["reverse_mode"] is True:
756             minimum = cnfg["maximum"]
757             maximum = cnfg["minimum"]
758         ######### was outside and is still outside ######
759         ######### ######### ######### ######### #########
760         if (cnfg["pre_val"] < cnfg["enc_first"] and cnfg["value"] <
…   cnfg["enc_first"]):
761             self.log("was below and still below")
762             return new_val
763         elif (cnfg["pre_val"] > cnfg["enc_second"] and cnfg["value"] >
…   cnfg["enc_second"]):
764             self.log("was above and still above")
765             return new_val
766         ## 1. Going Below
767         if (cnfg["pre_val"] >= cnfg["enc_first"] and cnfg["value"] <
…   cnfg["enc_first"]):
768             self.log("going below enter")
769             if cnfg["takeover_mode"] == "Pickup":
770                 if cnfg["reverse_mode"] is False and
…   cnfg["current_position"] > cnfg["previous_step_value"]:
771                     return new_val
772                 elif cnfg["reverse_mode"] is True and
…   cnfg["current_position"] < cnfg["previous_step_value"]:
773                     return new_val
774             if cnfg["reverse_mode"] is False:
775                 new_val = minimum
776                 self.log("going below 1")
777                 return new_val
778             elif cnfg["reverse_mode"] is True:
779                 new_val = minimum
780                 self.log("going below 2")
781                 return new_val
782         ## 2. Going Above
783         if (cnfg["pre_val"] <= cnfg["enc_second"] and cnfg["value"] >
```

```
783… cnfg["enc_second"]):
784             if cnfg["takeover_mode"] == "Pickup":
785                 self.log("THIS SHOULD FIRE 1")
786                 if cnfg["reverse_mode"] is False and
…   cnfg["current_position"] < cnfg["previous_step_value"]:
787                     self.log("THIS SHOULD FIRE 2")
788                     return new_val
789                 elif cnfg["reverse_mode"] is True and
…   cnfg["current_position"] > cnfg["previous_step_value"]:
790                     return new_val
791             if cnfg["reverse_mode"] is False:
792                 new_val = maximum
793                 self.log("going above 1")
794                 return new_val
795             elif cnfg["reverse_mode"] is True:
796                 new_val = maximum
797                 self.log("going above 2")
798                 return new_val
799         #########  >>0<< Coming inside ########
800         ######### ######### ######### #########
801         if (cnfg["pre_val"] < cnfg["enc_first"] and cnfg["value"] >=
…   cnfg["enc_first"]):
802             self.log("come in from below")
803
804         elif (cnfg["pre_val"] > cnfg["enc_second"] and cnfg["value"] <=
…   cnfg["enc_second"]):
805             self.log("coming in from above")
806         return False
807     def _velocity_seq(self,cnfg):
808         number_of_steps = cnfg['enc_second'] - cnfg['enc_first']
809         arr = []
810         i = 0
811         sequence_num = cnfg['enc_first']
812         while i <= number_of_steps:
813             arr.append(sequence_num)
814             i += 1
815             sequence_num += 1
816         return arr
817     def pickup(self, cnfg, max_min):
818         new_val = cnfg["current_position"]
819         found = False
820         if cnfg["previous_step_value"] is None:
821             self.log("just entered")
822
823             if cnfg["reverse_mode"] is False:
824                 if cnfg["pre_val"] < cnfg["enc_first"] and
…   cnfg["step_value"] > cnfg["current_position"]:
825                     new_val = cnfg["step_value"]
```

```
826                     found = True
827                     self.log("pickup 1 found")
828                 elif cnfg["pre_val"] > cnfg["enc_second"] and
…   cnfg["step_value"] < cnfg["current_position"]:
829                     new_val = cnfg["step_value"]
830                     found = True
831                     self.log("pickup 2 found")
832             elif cnfg["reverse_mode"] is True:
833                 if cnfg["pre_val"] < cnfg["enc_first"] and
…   cnfg["step_value"] < cnfg["current_position"]:
834                     new_val = cnfg["step_value"]
835                     found = True
836                     self.log("pickup 3 found")
837                 elif cnfg["pre_val"] > cnfg["enc_second"] and
…   cnfg["step_value"] > cnfg["current_position"]:
838                     new_val = cnfg["step_value"]
839                     found = True
840                     self.log("pickup 4 found")
841
842         else:
843             self.log("we were already in here")
844
845             if cnfg["previous_step_value"] < cnfg["current_position"] and
…   cnfg["step_value"] > cnfg["current_position"]:
846                 new_val = cnfg["step_value"]
847                 found = True
848                 self.log("pickup 4 found")
849             elif cnfg["previous_step_value"] > cnfg["current_position"]
…   and cnfg["step_value"] < cnfg["current_position"] :
850                 new_val = cnfg["step_value"]
851                 found = True
852                 self.log("pickup 5 found")
853             else:
854                 self.log("waiting for pickup")
855         if found is False:
856             msg = "remotify says: waiting for pickup " +
…   str(cnfg["step_value"]) + " >> " + str(cnfg["current_position"])
857             self.show_message(msg)
858         return new_val
859         step_num = cnfg["step_num"]
860         step_value = cnfg["step_value"]
861         remaining_steps = cnfg["steps"] - step_num
862         new_val = cnfg["current_position"]
863         distance_to_max = cnfg["maximum"] - cnfg["current_position"]
864         distance_to_min = cnfg["current_position"] - cnfg["minimum"]
865         speed_to_max = 0
866         speed_to_min = 0
867         if cnfg["current_position"] >= cnfg["minimum"] and
```

```
867… cnfg["current_position"] <= cnfg["maximum"]:
868             if max_min == "max" and distance_to_max > 0:
869                 if cnfg["reverse_mode"] is False and remaining_steps > 0:
…   speed_to_max = distance_to_max / remaining_steps
870                 elif cnfg["reverse_mode"] is True and step_num > 0:
…   speed_to_max = distance_to_max / step_num
871                 if speed_to_max is not 0: new_val = speed_to_max +
…   cnfg["current_position"]
872             elif max_min == "min" and distance_to_min > 0:
873                 if cnfg["reverse_mode"] is False and step_num > 0:
…   speed_to_min = distance_to_min / step_num
874                 elif cnfg["reverse_mode"] is True and remaining_steps > 0:
…   speed_to_min = distance_to_min / remaining_steps
875                 if speed_to_min is not 0: new_val =
…   cnfg["current_position"] - speed_to_min
876         return new_val
877     def value_scaling(self, cnfg, max_min):
878         step_num = cnfg["step_num"]
879         step_value = cnfg["step_value"]
880         remaining_steps = cnfg["steps"] - step_num
881         new_val = cnfg["current_position"]
882         distance_to_max = cnfg["maximum"] - cnfg["current_position"]
883         distance_to_min = cnfg["current_position"] - cnfg["minimum"]
884         speed_to_max = 0
885         speed_to_min = 0
886         if cnfg["current_position"] >= cnfg["minimum"] and
…   cnfg["current_position"] <= cnfg["maximum"]:
887             if max_min == "max" and distance_to_max > 0:
888                 if cnfg["reverse_mode"] is False and remaining_steps > 0:
…   speed_to_max = distance_to_max / remaining_steps
889                 elif cnfg["reverse_mode"] is True and step_num > 0:
…   speed_to_max = distance_to_max / step_num
890                 if speed_to_max is not 0: new_val = speed_to_max +
…   cnfg["current_position"]
891             elif max_min == "min" and distance_to_min > 0:
892                 if cnfg["reverse_mode"] is False and step_num > 0:
…   speed_to_min = distance_to_min / step_num
893                 elif cnfg["reverse_mode"] is True and remaining_steps > 0:
…   speed_to_min = distance_to_min / remaining_steps
894                 if speed_to_min is not 0: new_val =
…   cnfg["current_position"] - speed_to_min
895         return new_val
896     def track_num(self, track_num):
897         if ((hasattr(self, '_session')) and (self._session is not None)):
898             track_num = track_num + self._session._track_offset
899         else:
900             track_num = track_num
901         return track_num
```

```
902     def scene_num(self, scene_num):
903         if ((hasattr(self, '_session')) and (self._session is not None)):
904             scene_num = scene_num + self._session._scene_offset
905         else:
906             scene_num = scene_num
907         return scene_num
908     def log_cnfg_settings(self, cnfg):
909         for i in cnfg:
910             text = i + ": " + str(cnfg[i])
911             self.log(text)
912     def dump(self, obj):
913         for attr in dir(obj):
914             try:
915                 self.log_message("csslog: %s" % (attr))
916             except:
917                 self.log_message("next")
918     def log(self, msg):
919         if self.debug_on is True:
920             self.log_message("csslog:" + str(msg))
921     def pret(self, ugly):
922         for key,value in sorted(ugly.items()):
923             self.log_message(key)
924             self.log_message(value)
925             self.log_message("")
926 
927     ################## Extra Functions: Python 2.7 #################
928     def get_list(self, list_name):
929         try:
930             if list_name in self.lists:
931                 return self.lists[list_name]["value"]
932             else:
933                 self.log_message('csslog: The custom list "' +
… str(list_name) + '" does not exist')
934                 return False
935         except Exception as e:
936             self.log_message('csslog: There was an error getting a custom
… list with "get_list", ' + str(e))
937             return False
938     def get_list_length(self, list_name):
939         theList = self.get_list(list_name)
940         if theList is False:
941             return False
942         return len(theList)
943     def get_list_item(self, list_name, item_num):
944         try:
945             theList = self.get_list(list_name)
946             if theList is False:
947                 return False
```

```
948
949             list_len = self.get_list_length(list_name)
950             if list_len is False:
951                 return
952             if list_len >= item_num:
953                 return theList[item_num - 1]
954             else:
955                 self.log_message('csslog: Custom list "' + str(list_name)
… + '" does not have ' + str(item_num) + ' items')
956                 return False
957         except Exception as e:
958             self.log_message('csslog: There was an error in
… "get_list_item"', str(e))
959             return False
960     def add_to_list(self, list_name, value_to_add, position):
961         try:
962             theList = self.get_list(list_name)
963             if theList is False:
964                 return False
965             list_len = self.get_list_length(list_name)
966             if position is None or position > list_len:
967                 position = list_len
968             theList.insert(position, value_to_add)
969         except Exception as e:
970             self.log_message('csslog: There was an error in
… "add_to_list()", ' + str(e))
971             return False;
972     def remove_from_list(self, list_name, position):
973         try:
974             theList = self.get_list(list_name)
975             if theList is False:
976                 return False
977             list_len = self.get_list_length(list_name)
978             if list_len == 0:
979                 self.log_message("csslog: Nothing to delete from list '" +
… str(list_name) + "' as it's already empty")
980                 return
981             if position > list_len:
982                 self.log_message("csslog: Custom list '" + str(list_name)
… + "' does not contain " + str(position) + " items")
983                 return False
984             if position is None:
985                 position = list_len
986             theList.pop(position)
987         except Exception as e:
988             self.log_message('csslog: There was an error in
… "remove_from_list", ' + str(e))
989             return False;
```

```
    def clear_list(self, list_name):
        try:
            theList = self.get_list(list_name)
            if theList is False:
                return False
            del theList[:]
        except Exception as e:
            self.log_message('csslog: There was an error in "clear_list",
' + str(e))
            return False;
    def get_num_of_tracks(self, track_slug):
        try:
            s = "self.song()." + track_slug
            s = eval(s)
            return len(s)
        except:
            self.log_message('There was an error in get_num_of_tracks()')
            return -1

    def get_num_of_scenes(self):
        try:
            s = "self.song().scenes"
            s = eval(s)
            return len(s)
        except:
            self.log_message('There was an error in get_num_of_scenes()')
            return -1

    def get_num_of_devices(self, track_slug):
        try:
            s = "self.song()." + track_slug + ".devices"
            s = eval(s)
            return len(s)
        except:
            self.log_message('There was an error in get_num_of_devices()')
            return -1

    def get_selected_track_num(self):
        track = self.song().view.selected_track
        track = self.tuple_index(self.song().tracks, track)
        return track

    def get_selected_scene_num(self):
        scene = self.song().view.selected_scene
        scene = self.tuple_index(self.song().scenes, scene)
        return scene

    def get_selected_device_num(self, track_slug):
```

```
1037            try:
1038                device_list = "self.song()." + track_slug + ".devices"
1039                selected_device = "self.song()." + track_slug +
…   ".view.selected_device"
1040                s = self.tuple_index(eval(device_list), eval(selected_device))
1041                if(s == False and s != 0):
1042                    s = -1
1043                return s
1044            except:
1045                self.log_message('csslog: There was an error in
…   get_num_of_devices()')
1046                return -1
1047        def get_active_mode_id(self):
1048            global active_mode
1049            return active_mode
1050        def get_sessbox_track_offset(self):
1051            if hasattr(self, '_session') and self._session is not None:
1052                return self._session._track_offset
1053            else:
1054                return -1
1055        def get_sessbox_scene_offset(self):
1056            if hasattr(self, '_session') and self._session is not None:
1057                return self._session._scene_offset
1058            else:
1059                return -1
1060        def get_sessbox_last_track_number(self):
1061            if hasattr(self, '_session') and self._session is not None:
1062                last_track = self._session._track_offset +
…   self._session.width()
1063                return last_track
1064            else:
1065                return -1
1066        def get_sessbox_last_scene_number(self):
1067            if hasattr(self, '_session') and self._session is not None:
1068                last_scene = self._session._scene_offset +
…   self._session.height()
1069                return last_scene
1070            else:
1071                return -1
1072        def get_sessbox_width(self):
1073            if hasattr(self, '_session') and self._session is not None:
1074                return self._session.width()
1075            else:
1076                return -1
1077        def get_sessbox_height(self):
1078            if hasattr(self, '_session') and self._session is not None:
1079                return self._session.height()
1080            else:
```

```
1081            return -1
1082
1083    def get_sessbox_is_active(self):
1084        if hasattr(self, '_session') and self._session is not None:
1085            return True
1086        else:
1087            return False
1088    def set_highlighted_track(self, n):
1089        self.song().view.selected_track = self.song().tracks[n]
1090
1091    def set_highlighted_scene(self, n):
1092        self.song().view.selected_scene = self.song().scenes[n]
1093    def set_sessionbox_offsets(self, track_offset, scene_offset):
1094        if hasattr(self, '_session') and self._session is not None:
1095            self._session.set_offsets(track_offset, scene_offset)
1096    def set_sessionbox_combo_mode(self, combo):
1097        if hasattr(self, '_session') and self._session is not None:
1098            if combo == True:
1099                self._session._link
1100            elif combo == False:
1101                self._session._unlink
1102    def _quantizeDict(self):
1103        grid_setting =
… str(self.song().view.highlighted_clip_slot.clip.view.grid_quantization)
1104        is_it_triplet =
… self.song().view.highlighted_clip_slot.clip.view.grid_is_triplet
1105        if (is_it_triplet is True):
1106            grid_setting += "_triplet"
1107        RecordingQuantization = Live.Song.RecordingQuantization
1108        quantDict = {}
1109        quantDict["g_thirtysecond"] =
… RecordingQuantization.rec_q_thirtysecond
1110        quantDict["g_sixteenth"] = RecordingQuantization.rec_q_sixtenth
1111        quantDict["g_eighth"] = RecordingQuantization.rec_q_eight
1112        quantDict["g_quarter"] = RecordingQuantization.rec_q_quarter
1113        quantDict["g_eighth_triplet"] =
… RecordingQuantization.rec_q_eight_triplet
1114        quantDict["g_sixteenth_triplet"] =
… RecordingQuantization.rec_q_sixtenth_triplet
1115        return quantDict[grid_setting];
1116    def _arm_follow_track_selection(self):
1117        for track in self.song().tracks:
1118            if track.can_be_armed:
1119                track.arm = False
1120        if self.song().view.selected_track.can_be_armed:
1121            self.song().view.selected_track.arm = True
1122    def turn_inputs_off(self):
1123        send_feedback = False
```

```
1124            if hasattr(self, "global_feedback"):
1125                if self.global_feedback == "custom":
1126                    if self.global_feedback_active == True:
1127                        send_feedback = True
1128                elif hasattr(self, "controller_LED_on") and hasattr(self,
…   "controller_LED_off"):
1129                    send_feedback = True
1130            if send_feedback == True:
1131                for input_name in self.input_map:
1132                    input_ctrl = getattr(self, input_name)
1133                    input_ctrl.send_value(self.led_off)
1134        def feedback_brain(self, obj):
1135            cnfg = obj.copy()
1136            try:
1137                method_to_call = getattr(self, cnfg["feedback_brain"])
1138                method_to_call(cnfg)
1139            except:
1140                return
1141        def feedback_bool(self, feedback_to):
1142            control =   eval("self." + str(feedback_to["attached_to"]))
1143            param =         eval(feedback_to["module"] + "." +
…   feedback_to["ui_listener"])
1144            ctrl_on =   self.feedback_which_ctrl_on_off(feedback_to, "on")
1145            ctrl_off =  self.feedback_which_ctrl_on_off(feedback_to, "off")
1146            if(feedback_to["mapping_type"] == "Mute"):
1147                if param == False:
1148                    send_val = ctrl_on
1149                elif param == True:
1150                    send_val = ctrl_off
1151            else:
1152                if param == True:
1153                    send_val = ctrl_on
1154                elif param == False:
1155                    send_val = ctrl_off
1156            self.feedback_handler(feedback_to, send_val)
1157        def feedback_on_off(self, feedback_to):
1158            param =         eval(feedback_to["module"])
1159            ctrl_on =   self.feedback_which_ctrl_on_off(feedback_to, "on")
1160            ctrl_off =  self.feedback_which_ctrl_on_off(feedback_to, "off")
1161            param_value = round(param.value,2)
1162            mapping_type = str(feedback_to["mapping_type"])
1163            if feedback_to.has_key("maximum") and
…   feedback_to.has_key("minimum"):
1164                max_val = feedback_to["maximum"]
1165                min_val = feedback_to["minimum"]
1166            elif hasattr(param, "max") and hasattr(param, "min"):
1167                max_val = param.max
1168                max_val = round(max_val,2)
```

```
1169            min_val = param.min
1170            min_val = round(min_val,2)
1171        else:
1172            self.log_message(str(param) + " does not have a max/min
…   param")
1173            return
1174        send_val = None
1175        if param_value == max_val:
1176            send_val = ctrl_on
1177        elif param_value == min_val:
1178            send_val = ctrl_off
1179        if send_val is not None:
1180            self.feedback_handler(feedback_to, send_val)
1181        else:
1182            return
1183    def feedback_increment(self, feedback_to):
1184        control =   eval("self." + str(feedback_to["attached_to"]))
1185        param =         eval(feedback_to["module"])
1186        mapping_type = str(feedback_to["mapping_type"])
1187        ctrl_on =   self.feedback_which_ctrl_on_off(feedback_to, "on")
1188        ctrl_off =  self.feedback_which_ctrl_on_off(feedback_to, "off")
1189        snapping = feedback_to["snap_to"]
1190        mapping_type = str(feedback_to["mapping_type"])
1191        if feedback_to.has_key("maximum") and
…   feedback_to.has_key("minimum"):
1192            max_val = feedback_to["maximum"]
1193            min_val = feedback_to["minimum"]
1194            if mapping_type != "On/Off":
1195                max_val = self.percent_as_value(feedback_to["module"],
…   feedback_to["maximum"])
1196                min_val = self.percent_as_value(feedback_to["module"],
…   feedback_to["minimum"])
1197        elif hasattr(param, "max") and hasattr(param, "min"):
1198            max_val = param.max
1199            min_val = param.min
1200        else:
1201            self.log_message(str(param) + " does not have a max/min
…   param")
1202            return
1203        if snapping == False and param.value < min_val:
1204            send_val = ctrl_off
1205        elif param.value < max_val:
1206            send_val = ctrl_on
1207        else:
1208            send_val = ctrl_off
1209        self.feedback_handler(feedback_to, send_val)
1210    def feedback_decrement(self, feedback_to):
1211        control =   eval("self." + str(feedback_to["attached_to"]))
```

```
1212            param =         eval(feedback_to["module"])
1213            mapping_type = str(feedback_to["mapping_type"])
1214            ctrl_on =   self.feedback_which_ctrl_on_off(feedback_to, "on")
1215            ctrl_off =  self.feedback_which_ctrl_on_off(feedback_to, "off")
1216            snapping = feedback_to["snap_to"]
1217            if feedback_to.has_key("maximum") and
…   feedback_to.has_key("minimum"):
1218                max_val = feedback_to["maximum"]
1219                min_val = feedback_to["minimum"]
1220                if mapping_type != "On/Off":
1221                    max_val = self.percent_as_value(feedback_to["module"],
…   feedback_to["maximum"])
1222                    min_val = self.percent_as_value(feedback_to["module"],
…   feedback_to["minimum"])
1223            elif hasattr(param, "max") and hasattr(param, "min"):
1224                max_val = param.max
1225                min_val = param.min
1226            else:
1227                self.log_message(str(param) + " does not have a max/min
…   param")
1228                return
1229            if snapping == False and param.value > max_val:
1230                send_val = ctrl_off
1231            elif param.value > min_val:
1232                send_val = ctrl_on
1233            else:
1234                send_val = ctrl_off
1235            self.feedback_handler(feedback_to, send_val)
1236        def feedback_which_ctrl_on_off(self, feedback_to, on_off):
1237            if feedback_to["LED_feedback"] == "default":
1238                ctrl_on = self.led_on
1239                ctrl_off = self.led_off
1240            elif feedback_to["LED_feedback"] == "custom":
1241                if feedback_to["ctrl_type"] == "on/off" or
…   feedback_to["ctrl_type"] == "increment" or feedback_to["ctrl_type"] ==
…   "decrement":
1242                    ctrl_on = feedback_to["LED_on"]
1243                    ctrl_off = feedback_to["LED_off"]
1244                elif feedback_to["ctrl_type"] == "absolute" or
…   feedback_to["ctrl_type"] == "relative":
1245                    ctrl_on = feedback_to["enc_first"]
1246                    ctrl_off = feedback_to["enc_second"]
1247            if on_off == "on":
1248                value = ctrl_on
1249            elif on_off == "off":
1250                value = ctrl_off
1251            return value;
1252        def feedback_range(self, feedback_to):
```

```
1253         if feedback_to['ctrl_type'] == "on/off":
1254             self.feedback_on_off(feedback_to)
1255         elif feedback_to['ctrl_type'] == "increment":
1256             self.feedback_increment(feedback_to)
1257         elif feedback_to['ctrl_type'] == "decrement":
1258             self.feedback_decrement(feedback_to)
1259         control =   eval("self." + str(feedback_to["attached_to"]))
1260         param =         eval(feedback_to["module"])
1261         ctrl_min =  feedback_to["minimum"]
1262         ctrl_max =  feedback_to["maximum"]
1263         ctrl_type = feedback_to["ctrl_type"]
1264         default_ctrl_first = 0
1265         default_ctrl_last = 127
1266         if ctrl_type == "relative":
1267             crl_reverse = False
1268             ctrl_first = 0
1269             ctrl_last = 127
1270         else:
1271             crl_reverse = feedback_to["reverse_mode"]
1272             ctrl_first = feedback_to["enc_first"]
1273             ctrl_last = feedback_to["enc_second"]
1274         param_range = param.max - param.min
1275         orig_param_range = param.max - param.min
1276         param_range = ctrl_max * orig_param_range / 100
1277         ctrl_min_as_val = ctrl_min * orig_param_range / 100
1278         param_range = param_range - ctrl_min_as_val
1279         param_value = param.value - ctrl_min_as_val
1280 
1281         if orig_param_range == 2.0 and param.min == -1.0:
1282             param_value = param_value + 1
1283         percentage_control_is_at = param_value / param_range * 100
1284         ctrl_range = ctrl_last - ctrl_first
1285         percentage_of_ctrl_range = ctrl_range * percentage_control_is_at /
…  100 + ctrl_first
1286         percentage_of_ctrl_range = round(percentage_of_ctrl_range,0)
1287         if crl_reverse == True:
1288             percentage_of_ctrl_range = ctrl_range -
…  percentage_of_ctrl_range
1289         self.feedback_handler(feedback_to, percentage_of_ctrl_range)
1290     def feedback_a_b_crossfade_assign(self, feedback_to):
1291         assigned_val = eval(str(feedback_to['parent_track']) +
…  ".mixer_device.crossfade_assign")
1292         if(assigned_val == 0):
1293             send_val = feedback_to["LED_on"]
1294         elif(assigned_val == 1):
1295             send_val = feedback_to["LED_off"]
1296         elif(assigned_val == 2):
1297             send_val = feedback_to["LED_assigned_to_b"]
```

```
1298            else:
1299                send_val = 0
1300            self.feedback_handler(feedback_to, send_val)
1301        def feedback_handler(self, config, send_val):
1302            send_feedback = False
1303            if config.has_key("LED_feedback"):
1304                if config["LED_feedback"] == "custom":
1305                    if config["LED_feedback_active"] == "1" or
…   config["LED_feedback_active"] == "true":
1306                        send_feedback = True
1307                elif hasattr(self, "global_feedback"):
1308                    if self.global_feedback == "custom":
1309                        if self.global_feedback_active == True:
1310                            send_feedback = True
1311                    elif hasattr(self, "controller_LED_on") and hasattr(self,
…   "controller_LED_off"):
1312                        send_feedback = True
1313                if send_feedback == True:
1314                    if config["LED_feedback"] == "custom":
1315                        for item in config["LED_send_feedback_to_selected"]:
1316                            feedback_control =  eval("self." + str(item))
1317                            feedback_control.send_value(send_val)
1318                    else:
1319                        control =   eval("self." + str(config["attached_to"]))
1320                        control.send_value(send_val)
1321                else:
1322                    self.log("feedback_handler says 'not sending led
…   feedback'")
1323        def sess_highlight_banking_calculate(self, feedback_to,
…   num_of_tracks_scenes, offset_is_at):
1324            ctrl_first = feedback_to["enc_first"]
1325            ctrl_last = feedback_to["enc_second"]
1326            ctrl_range = ctrl_last - ctrl_first
1327            if feedback_to['ctrl_type'] == "absolute" or
…   feedback_to['ctrl_type'] == "relative":
1328                percentage_control_is_at = offset_is_at / num_of_tracks_scenes
…   * 100
1329                velocity_val = ctrl_range * percentage_control_is_at / 100 +
…   ctrl_first
1330                velocity_val = int(velocity_val)
1331            elif feedback_to['ctrl_type'] == "on/off" or
…   feedback_to['ctrl_type'] == "increment":
1332                if offset_is_at == num_of_tracks_scenes:
1333                    velocity_val = feedback_to["LED_on"]
1334                else:
1335                    velocity_val = feedback_to["LED_off"]
1336            elif feedback_to['ctrl_type'] == "decrement":
1337                if offset_is_at == 0:
```

```
1338                velocity_val = feedback_to["LED_off"]
1339            else:
1340                velocity_val = feedback_to["LED_on"]
1341        if feedback_to['ctrl_type'] == "absolute" and
…   feedback_to["reverse_mode"] == True:
1342            velocity_val = ctrl_range - velocity_val
1343        self.feedback_handler(feedback_to, velocity_val)
1344    def feedback_scroll_mode_selector(self, feedback_to):
1345        global active_mode
1346        num_of_tracks_scenes = len(self.modes) - 1
1347        count = 0
1348        for mode_num in self.modes.values():
1349            if mode_num == active_mode:
1350                offset_is_at = count
1351                break
1352            count += 1
1353        self.sess_highlight_banking_calculate(feedback_to,
…   num_of_tracks_scenes, offset_is_at)
1354    def feedback_scroll_mode_selector_select(self, feedback_to):
1355        global active_mode
1356        mode_to_select = int(feedback_to["func_arg"])
1357        if int(active_mode) == mode_to_select:
1358            self.feedback_handler(feedback_to, feedback_to["LED_on"])
1359        else:
1360            self.feedback_handler(feedback_to, feedback_to["LED_off"])
1361    def feedback_param_banking_select(self, feedback_to):
1362        if type(feedback_to["banking_number"]) == str:
1363            banking_number =
…   self.get_modifier_value(feedback_to["banking_number"])
1364        else:
1365            banking_number = feedback_to["banking_number"] - 1
1366        parent_device_id = feedback_to["parent_device_id"]
1367        offset_is_at = getattr(self, "device_id_" + str(parent_device_id)
…   + "_active_bank")
1368        if banking_number == offset_is_at:
1369            self.feedback_handler(feedback_to, feedback_to["LED_on"])
1370        else:
1371            self.feedback_handler(feedback_to, feedback_to["LED_off"])
1372    def feedback_param_banking(self, feedback_to):
1373        self.log_message("scroll banking fired")
1374        parent_device_id = feedback_to["parent_device_id"]
1375        bank_array = getattr(self, "device_id_" + str(parent_device_id) +
…   "_banks")
1376        num_of_tracks_scenes = len(bank_array) - 1
1377        offset_is_at = getattr(self, "device_id_" + str(parent_device_id)
…   + "_active_bank")
1378        self.sess_highlight_banking_calculate(feedback_to,
…   num_of_tracks_scenes, offset_is_at)
```

```
1379     def feedback_highlight_nav_select(self, feedback_to):
1380         tracks_or_scenes = feedback_to["tracks_scenes"]
1381         tracks_scene_num = int(feedback_to["highlight_number"])
1382         if tracks_or_scenes == "tracks":
1383             offset_is_at = int(self.selected_track_idx()) - 1
1384         elif tracks_or_scenes == "scenes":
1385             offset_is_at = int(self.selected_scene_idx()) - 1
1386         if tracks_scene_num == offset_is_at:
1387             self.feedback_handler(feedback_to, feedback_to["LED_on"])
1388         else:
1389             self.feedback_handler(feedback_to, feedback_to["LED_off"])
1390     def feedback_highlight_nav(self, feedback_to):
1391         tracks_or_scenes = feedback_to["tracks_scenes"]
1392         if tracks_or_scenes == "tracks":
1393             offset_is_at = int(self.selected_track_idx()) - 1
1394             num_of_tracks_scenes = int(len(self.song().tracks)) - 1
1395         elif tracks_or_scenes == "scenes":
1396             offset_is_at = int(self.selected_scene_idx()) - 1
1397             num_of_tracks_scenes = int(len(self.song().scenes)) - 1
1398         self.sess_highlight_banking_calculate(feedback_to,
…  num_of_tracks_scenes, offset_is_at)
1399     def feedback_sessbox_nav_select(self, feedback_to):
1400         try:
1401             self._session
1402         except:
1403             self.show_message("There's no Session Box to select for
…  feedback")
1404             return
1405         tracks_scene_num = int(feedback_to["highlight_number"])
1406         tracks_or_scenes = feedback_to["tracks_scenes"]
1407         if tracks_or_scenes == "tracks":
1408             offset_is_at = int(self._session.track_offset())
1409         elif tracks_or_scenes == "scenes":
1410             offset_is_at = int(self._session.scene_offset())
1411         if tracks_scene_num == offset_is_at:
1412             self.feedback_handler(feedback_to, feedback_to["LED_on"])
1413         else:
1414             self.feedback_handler(feedback_to, feedback_to["LED_off"])
1415     def feedback_sessbox_nav(self, feedback_to):
1416         try:
1417             self._session
1418         except:
1419             self.show_message("There's no Session Box to scroll for
…  feedback sir.")
1420             return
1421         tracks_or_scenes = feedback_to["tracks_scenes"]
1422         if tracks_or_scenes == "tracks":
1423             offset_is_at = int(self._session.track_offset())
```

```
1424            num_of_tracks_scenes = int(len(self.song().tracks)) - 1
1425        elif tracks_or_scenes == "scenes":
1426            offset_is_at = int(self._session.scene_offset())
1427            num_of_tracks_scenes = int(len(self.song().scenes)) - 1
1428        self.sess_highlight_banking_calculate(feedback_to,
…   num_of_tracks_scenes, offset_is_at)
1429    def feedback_tempo(self, feedback_to):
1430        control =   eval("self." + str(feedback_to["attached_to"]))
1431        param =         eval(feedback_to["module"])
1432        ctrl_min =  feedback_to["minimum"]
1433        ctrl_max =  feedback_to["maximum"]
1434        ctrl_type = feedback_to["ctrl_type"]
1435        ctrl_first = feedback_to["enc_first"]
1436        ctrl_last = feedback_to["enc_second"]
1437        default_ctrl_first = 0
1438        default_ctrl_last = 127
1439        crl_reverse = feedback_to["reverse_mode"]
1440        param_range = ctrl_max - ctrl_min
1441        param =         eval(feedback_to["module"] + "." +
…   feedback_to["ui_listener"])
1442        zero = ctrl_min
1443        if param < ctrl_min or param > ctrl_max:
1444            self.log("tempo is outside ctrl_min / ctrl_max")
1445        else:
1446            zerod_param = param - zero
1447            percentage_control_is_at = zerod_param / param_range * 100
1448        ctrl_range = ctrl_last - ctrl_first
1449        percentage_of_ctrl_range = ctrl_range * percentage_control_is_at /
…   100 + ctrl_first
1450        if crl_reverse == True:
1451            percentage_of_ctrl_range = ctrl_range -
…   percentage_of_ctrl_range
1452        self.feedback_handler(feedback_to, percentage_of_ctrl_range)
1453    def mode_device_bank_leds(self, mode_id):
1454        config_map = "mode_" + str(mode_id) + "_configs_map"
1455        config_map = getattr(self, config_map)
1456        for config_name in config_map:
1457            config = getattr(self, config_name)
1458            if config["mapping_type"] == "Parameter Bank":
1459                parent_id = config["parent_json_id"]
1460                bank_names_array_name = "device_id_" + str(parent_id) +
…   "_banks"
1461                active_bank_name = "device_id_" + str(parent_id) +
…   "_active_bank"
1462                bank_names_array = getattr(self, bank_names_array_name)
1463                active_bank = getattr(self, active_bank_name)
1464                for index, bank_name in enumerate(bank_names_array):
1465                    if bank_name == config_name:
```

```
1466                            if index == active_bank:
1467                                led_on = config["LED_on"]
1468                                self.feedback_handler(config, led_on)
1469                            else:
1470                                led_off = config["LED_off"]
1471                                self.feedback_handler(config, led_off)
1472        def bank_led_feedback(self, parent_device_id):
1473            global active_mode
1474            device = "device_id_" + str(parent_device_id);
1475            device_bank_array = getattr(self, device + "_banks")
1476            active_bank_idx = getattr(self, device + "_active_bank")
1477            device_bank_params = getattr(self, device + "_bank_parameters_" +
…   str(active_bank_idx))
1478            for index, val in enumerate(device_bank_array):
1479                bank_cnfg = getattr(self, val)
1480                bank_cnfg["LED_feedback"] = "custom";
1481                if index == active_bank_idx:
1482                        if bank_cnfg.has_key("LED_on"):
1483                            led_on = bank_cnfg["LED_on"]
1484                            self.feedback_handler(bank_cnfg, led_on)
1485                else:
1486                    if bank_cnfg.has_key("LED_off"):
1487                        led_off = bank_cnfg["LED_off"]
1488                        self.feedback_handler(bank_cnfg, led_off)
1489
1490            remove_mode = getattr(self, "_remove_mode" + active_mode +
…   "_ui_listeners")
1491            remove_mode()
1492            activate_mode = getattr(self, "_mode" + active_mode +
…   "_ui_listeners")
1493            activate_mode()
1494            for param in device_bank_params:
1495                fire_param_feedback = getattr(self, param + "_led_listener")
1496                fire_param_feedback()
1497        def device_feedback(self, mode_id=None):
1498            if (mode_id == None):
1499                global active_mode
1500                mode_id = active_mode
1501            config_map = "mode_" + str(mode_id) + "_configs_map"
1502            config_map = getattr(self, config_map)
1503            for config_name in config_map:
1504                config = getattr(self, config_name)
1505                if config.has_key("mapping_type") and config["mapping_type"]
…   == "Device":
1506                    led_on = config["LED_on"]
1507                    led_off = config["LED_off"]
1508                    try:
1509                        device = eval(config["module"])
```

```
1510                except:
1511                    self.feedback_handler(config, led_off)
1512                    return
1513                find = config["module"].find("selected_track")
1514                if find >= 0:
1515                    selected_device =
…   self.song().view.selected_track.view.selected_device
1516                    if device == selected_device:
1517                        self.feedback_handler(config, led_on)
1518                    else:
1519                        self.feedback_handler(config, led_off)
1520                else:
1521                    for parent_name in config_map:
1522                        parent_config = getattr(self, parent_name)
1523                        if parent_config["json_id"] ==
…   config["parent_json_id"]:
1524                            parent_track = parent_config["module"]
1525                            break
1526                    tracks_selected_device = eval(parent_track +
…   ".view.selected_device")
1527                    if device == tracks_selected_device:
1528                        self.feedback_handler(config, led_on)
1529                    else:
1530                        self.feedback_handler(config, led_off)
1531    def _on_selected_track_changed(self):
1532        global active_mode, prev_active_mode, modes
1533        self.log("selected track changed")
1534        remove_modex_led_listeners = "_remove_mode" + active_mode +
…   "_led_listeners"
1535        add_modex_led_listeners = "_mode" + active_mode + "_led_listeners"
1536        if(hasattr(self, remove_modex_led_listeners)):
1537            mode_to_call = getattr(self, remove_modex_led_listeners)
1538            mode_to_call()
1539        if(hasattr(self, add_modex_led_listeners)):
1540            mode_to_call = getattr(self, add_modex_led_listeners)
1541            mode_to_call()
1542            self.track_feedback()
1543            self.device_feedback()
1544        self.refresh_state()
1545    def track_feedback(self, mode_id=None):
1546        if (mode_id == None):
1547            global active_mode
1548            mode_id = active_mode
1549        config_map = "mode_" + str(mode_id) + "_configs_map"
1550        config_map = getattr(self, config_map)
1551        selected_track = self.song().view.selected_track
1552        for config_name in config_map:
1553            config = getattr(self, config_name)
```

```
1554                if config.has_key("mapping_type") and config["mapping_type"]
… == "Track":
1555                    led_on = config["LED_on"]
1556                    led_off = config["LED_off"]
1557                    try:
1558                        track = eval(config["module"])
1559                    except:
1560                        self.feedback_handler(config, led_off)
1561                        return
1562                    if track == selected_track:
1563                        self.feedback_handler(config, led_on)
1564                    else:
1565                        self.feedback_handler(config, led_off)
1566        def create_clip_slot_map(self):
1567            num_of_tracks = int(len(self.song().tracks))
1568            num_of_scenes = int(len(self.song().scenes))
1569            for track in xrange(0,num_of_tracks):
1570                for scene in xrange(0,num_of_scenes):
1571                    if(not
… self.song().tracks[track].clip_slots[scene].has_clip_has_listener(self.
… _on_clip_added_removed)):
1572                        try:
1573
… self.song().tracks[track].clip_slots[scene].add_has_clip_listener(self.
… _on_clip_added_removed)
1574                        except:
1575                            pass
1576        def _on_clip_added_removed(self):
1577            global active_mode
1578            self.log("a clip has been added or removed")
1579            updated_by = "_on_clip_added_removed"
1580            self._remove_custom_lom_listeners_handler(active_mode, updated_by)
1581            self._add_custom_lom_listeners_handler(active_mode, updated_by)
1582        def _on_tracks_changed(self):
1583            global active_mode
1584            self.log("tracks changed")
1585            updated_by = "_on_tracks_changed"
1586            self._remove_custom_lom_listeners_handler(active_mode, updated_by)
1587            self._add_custom_lom_listeners_handler(active_mode, updated_by)
1588            self.all_track_device_listeners()
1589            self.create_clip_slot_map()
1590        def _on_scenes_changed(self):
1591            global active_mode
1592            self.log("scenes changed")
1593            updated_by = "_on_scenes_changed"
1594            self._remove_custom_lom_listeners_handler(active_mode, updated_by)
1595            self._add_custom_lom_listeners_handler(active_mode, updated_by)
1596            self.create_clip_slot_map()
```

```
1597     def _on_devices_changed(self):
1598         global active_mode, prev_active_mode, modes
1599         self.log("devices changed")
1600         updated_by = "_on_devices_changed"
1601         self._remove_custom_lom_listeners_handler(active_mode, updated_by)
1602         self._add_custom_lom_listeners_handler(active_mode, updated_by)
1603         try:
1604             mode_to_call = getattr(self, "_remove_mode" + active_mode +
…"_led_listeners")
1605             mode_to_call()
1606             mode_to_call = getattr(self, "_mode" + active_mode +
…"_led_listeners")
1607             mode_to_call()
1608         except:
1609             pass
1610     def _on_selected_device_changed(self):
1611         global active_mode, prev_active_mode, modes
1612         self.log("selected device changed")
1613         try:
1614             mode_to_call = getattr(self, "_remove_mode" + active_mode +
…"_led_listeners")
1615             mode_to_call()
1616             mode_to_call = getattr(self, "_mode" + active_mode +
…"_led_listeners")
1617             mode_to_call()
1618             self.device_feedback()
1619             self.refresh_state()
1620         except:
1621             pass
1622     def _on_selected_parameter_changed(self):
1623         global active_mode
1624         self.log("selected parameter changed")
1625         if(hasattr(self.song().view.selected_parameter,
…"canonical_parent") and
…hasattr(self.song().view.selected_parameter.canonical_parent, "type")):
1626             updated_by = "_on_selected_parameter_changed"
1627             self._remove_custom_lom_listeners_handler(active_mode,
…updated_by)
1628             self._add_custom_lom_listeners_handler(active_mode,
…updated_by)
1629     def _on_selected_scene_changed(self):
1630         global active_mode, prev_active_mode, modes
1631         self.log("selected scene changed")
1632         remove_modex_led_listeners = "_remove_mode" + active_mode +
…"_led_listeners"
1633         add_modex_led_listeners = "_mode" + active_mode + "_led_listeners"
1634         if(hasattr(self, remove_modex_led_listeners)):
1635             mode_to_call = getattr(self, remove_modex_led_listeners)
```

```
1636                mode_to_call()
1637            if(hasattr(self, add_modex_led_listeners)):
1638                mode_to_call = getattr(self, add_modex_led_listeners)
1639                mode_to_call()
1640            self.refresh_state()
1641        def _all_tracks_listener(self):
1642            global active_mode, prev_active_mode, modes
1643            self.log("mode 1 tracks listener")
1644            mode_to_call = getattr(self, "_remove_mode" + active_mode +
…   "_led_listeners")
1645            mode_to_call()
1646            mode_to_call = getattr(self, "_mode" + active_mode +
…   "_led_listeners")
1647            mode_to_call()
1648        def all_track_device_listeners(self):
1649            numtracks = len(self.song().tracks)
1650            for index in range(numtracks):
1651                try:
1652
…   self.song().tracks[index].view.add_selected_device_listener(self.
…   _on_selected_device_changed)
1653
…   self.song().tracks[index].add_devices_listener(self._on_devices_changed)
1654                except:
1655                    pass
1656            num_returns = len(self.song().return_tracks)
1657            for index in range(num_returns):
1658                try:
1659
…   self.song().return_tracks[index].view.add_selected_device_listener(self.
…   _on_selected_device_changed)
1660
…   self.song().return_tracks[index].add_devices_listener(self.
…   _on_devices_changed)
1661                except:
1662                    pass
1663            try:
1664
…   self.song().master_track.view.add_selected_device_listener(self.
…   _on_selected_device_changed)
1665
…   self.song().master_track.add_devices_listener(self._on_devices_changed)
1666            except:
1667                pass
1668        def _remove_all_track_device_listeners(self):
1669            numtracks = len(self.song().tracks)
1670            for index in range(numtracks):
1671                try:
```

```
1672
…   self.song().tracks[index].view.remove_selected_device_listener(self.
…   _on_selected_device_changed)
1673
…   self.song().tracks[index].remove_devices_listener(self._on_devices_changed
…   )
1674            except:
1675                pass
1676        num_returns = len(self.song().return_tracks)
1677        for index in range(num_returns):
1678            try:
1679
…   self.song().return_tracks[index].view.remove_selected_device_listener(self
…   ._on_selected_device_changed)
1680
…   self.song().return_tracks[index].remove_devices_listener(self.
…   _on_devices_changed)
1681            except:
1682                pass
1683        try:
1684
…   self.song().master_track.view.remove_selected_device_listener(self.
…   _on_selected_device_changed)
1685
…   self.song().master_track.remove_devices_listener(self._on_devices_changed)
1686        except:
1687            pass
1688    ################################################
1689    ############# Extra Functions ##################
1690    ################################################
1691    def scroll_through_devices(self, cnfg):
1692        NavDirection = Live.Application.Application.View.NavDirection
1693        if cnfg["ctrl_type"] == "absolute":
1694            if cnfg["value"] > cnfg["pre_val"]:
1695                if cnfg["reverse_mode"] is False:
1696                    goto = "right"
1697                elif cnfg["reverse_mode"] is True:
1698                    goto = "left"
1699                times = 1;
1700            elif cnfg["value"] < cnfg["pre_val"]:
1701                if cnfg["reverse_mode"] is False:
1702                    goto = "left"
1703                elif cnfg["reverse_mode"] is True:
1704                    goto = "right"
1705                times = 1;
1706        elif cnfg["ctrl_type"] == "relative":
1707            if cnfg["enc_first"] == cnfg["value"]:
1708                goto = "left"
```

```
1709                times = cnfg["steps"];
1710            elif cnfg["enc_second"] == cnfg["value"]:
1711                goto = "right"
1712                times = cnfg["steps"];
1713        elif cnfg["ctrl_type"] == "on/off":
1714            if cnfg["enc_first"] == cnfg["value"]:
1715                    goto = "right"
1716            elif cnfg["enc_second"] == cnfg["value"]:
1717                    goto = "right"
1718        elif cnfg["ctrl_type"] == "increment":
1719            if cnfg["enc_first"] == cnfg["value"]:
1720                goto = "right"
1721                times = cnfg["steps"];
1722        elif cnfg["ctrl_type"] == "decrement":
1723            if cnfg["enc_first"] == cnfg["value"]:
1724                goto = "left"
1725                times = cnfg["steps"];
1726        if goto == "right":
1727            for x in range(0, times):
1728                self._scroll_device_chain(NavDirection.right)
1729        elif goto == "left":
1730            for x in range(0, times):
1731                self._scroll_device_chain(NavDirection.left)
1732    def _scroll_device_chain(self, direction):
1733        view = self.application().view
1734        if not view.is_view_visible('Detail') or not
…   view.is_view_visible('Detail/DeviceChain'):
1735            view.show_view('Detail')
1736            view.show_view('Detail/DeviceChain')
1737        else:
1738            view.scroll_view(direction, 'Detail/DeviceChain', False)
1739    def selected_device_idx(self):
1740        self._device =
…   self.song().view.selected_track.view.selected_device
1741        return self.tuple_index(self.song().view.selected_track.devices,
…   self._device)
1742    def selected_track_idx(self):
1743        self._track = self.song().view.selected_track
1744        self._track_num = self.tuple_index(self.song().tracks,
…   self._track)
1745        self._track_num = self._track_num + 1
1746        return self._track_num
1747    def selected_scene_idx(self):
1748        self._scene = self.song().view.selected_scene
1749        self._scene_num = self.tuple_index(self.song().scenes,
…   self._scene)
1750        self._scene_num = self._scene_num + 1
1751        return self._scene_num
```

```
1752     def tuple_index(self, tuple, obj):
1753         for i in xrange(0, len(tuple)):
1754             if (tuple[i] == obj):
1755                 return i
1756         return(False)
1757     def select_a_device(self, cnfg):
1758         parent_track = cnfg["parent_track"]
1759         device_chain = cnfg["device_chain"]
1760         chain_selector = "self.song().view.selected_track" + device_chain
1761         try:
1762             self.song().view.selected_track = eval(parent_track)
1763             try:
1764                 self.song().view.select_device(eval(chain_selector))
1765             except IndexError:
1766                 self.show_message("Device you are trying to select does
…  not exist on track.")
1767         except IndexError:
1768             self.show_message("Track does not exist for the device you are
…  selecting.")
1769     def a_b_crossfade_assign(self, cnfg):
1770         assignment_type = cnfg['assignment_type'];
1771         if(assignment_type == "Scroll"):
1772             goto = self.scroll_a_b_assign(cnfg);
1773             if goto > 2:
1774                 goto = 2
1775         elif cnfg["enc_first"] == cnfg["value"]:
1776             if assignment_type == "Select A":
1777                 goto = 0
1778             elif assignment_type == "Select None":
1779                 goto = 1
1780             elif assignment_type == "Select B":
1781                 goto = 2
1782             else:
1783                 goto = 0
1784         setattr(eval(str(cnfg['parent_track']) + ".mixer_device"),
…  "crossfade_assign", goto)
1785     def scroll_a_b_assign(self, cnfg):
1786         should_it_fire = self.should_it_fire(cnfg)
1787         if(should_it_fire != 1):
1788             return
1789         current_assigned_value = eval(str(cnfg['parent_track']) +
…  ".mixer_device.crossfade_assign")
1790         length = 3
1791         if cnfg["ctrl_type"] == "absolute":
1792             divider = (cnfg["enc_second"] - cnfg["enc_first"]) / length
1793             goto = int(cnfg["value"] / divider)
1794             if cnfg["reverse_mode"] is True:
1795                 if(goto >= 2):
```

```
1796                    goto = 0
1797                elif(goto == 0):
1798                    goto = 2
1799            goto = int(goto)
1800        elif cnfg["ctrl_type"] == "relative":
1801            self.log_message("csslog: relative");
1802            if cnfg["enc_first"] == cnfg["value"] and
… current_assigned_value > 0:
1803                goto = current_assigned_value - 1
1804            elif cnfg["enc_second"] == cnfg["value"] and
… current_assigned_value < 2:
1805                goto = current_assigned_value + 1
1806        elif cnfg["ctrl_type"] == "on/off":
1807            if current_assigned_value < 2:
1808                goto = current_assigned_value + 1
1809            elif current_assigned_value >= 2:
1810                goto = 0
1811        elif cnfg["ctrl_type"] == "increment":
1812            if current_assigned_value < 2:
1813                goto = current_assigned_value + 1
1814            else:
1815                goto = current_assigned_value
1816        elif cnfg["ctrl_type"] == "decrement":
1817            if current_assigned_value > 0:
1818                goto = current_assigned_value - 1
1819            else:
1820                goto = current_assigned_value
1821        return int(goto)
1822    def scroll_highlight(self, cnfg):
1823        if cnfg["tracks_scenes"] == "tracks":
1824            length = len(self.song().tracks) +
… len(self.song().return_tracks)
1825            selected = self.selected_track_idx() - 1
1826        elif cnfg["tracks_scenes"] == "scenes":
1827            length = len(self.song().scenes)
1828            selected = self.selected_scene_idx() - 1
1829        else:
1830            self.log("scroll_highlight error, tracks_scenes was not set")
1831        if cnfg["ctrl_type"] == "absolute":
1832            divider = (cnfg["enc_second"] - cnfg["enc_first"]) / length
1833            if cnfg["reverse_mode"] is False:
1834                goto = cnfg["value"] / divider
1835            elif cnfg["reverse_mode"] is True:
1836                goto = (divider * length) / cnfg["value"]
1837            goto = int(goto)
1838        elif cnfg["ctrl_type"] == "relative":
1839            if cnfg["enc_first"] == cnfg["value"]:
1840                goto = selected - cnfg["steps"]
```

```
1841                elif cnfg["enc_second"] == cnfg["value"]:
1842                    goto = selected + cnfg["steps"]
1843            elif cnfg["ctrl_type"] == "on/off":
1844                if cnfg["enc_first"] == cnfg["value"]:
1845                    goto = length
1846                elif cnfg["enc_second"] == cnfg["value"]:
1847                    goto = 0
1848            elif cnfg["ctrl_type"] == "increment":
1849                goto = selected + cnfg["steps"]
1850            elif cnfg["ctrl_type"] == "decrement":
1851                goto = selected - cnfg["steps"]
1852            if goto <= length and goto >= 0 and goto != selected:
1853                cnfg["highlight_number"] = goto
1854                self.select_highlight(cnfg)
1855        def select_sess_offset(self, cnfg):
1856            try:
1857                self._session
1858            except:
1859                self.show_message("There's no Session Box to select, buddy.")
1860                return
1861            tracks_scenes = cnfg["tracks_scenes"]
1862            track_offset = self._session.track_offset()
1863            scene_offset = self._session.scene_offset()
1864            if type(cnfg["highlight_number"]) == str:
1865                change_to = self.get_modifier_value(cnfg["highlight_number"])
1866            else:
1867                change_to = cnfg["highlight_number"]
1868            if tracks_scenes == "tracks":
1869                track_offset = change_to
1870            elif tracks_scenes == "scenes":
1871                scene_offset = change_to
1872            try:
1873                self._session.set_offsets(track_offset, scene_offset)
1874                self._session._reassign_scenes()
1875                self.set_highlighting_session_component(self._session)
1876                self.refresh_state()
1877                self.call_script_reaction(active_mode, None,
…   'session_box_position')
1878            except:
1879                self.show_message("unable to move session box there.")
1880        def scroll_sess_offset(self, cnfg):
1881            try:
1882                self._session
1883            except:
1884                self.show_message("There's no Session Box to scroll, buddy.")
1885                return
1886            tracks_scenes = cnfg["tracks_scenes"]
1887            track_offset = self._session.track_offset()
```

```
1888            scene_offset = self._session.scene_offset()
1889            if cnfg["tracks_scenes"] == "tracks":
1890                length = len(self.song().tracks)
1891                selected = track_offset
1892            elif cnfg["tracks_scenes"] == "scenes":
1893                length = len(self.song().scenes)
1894                selected = scene_offset
1895            else:
1896                self.log("scroll_sess_offset error, tracks_scenes was not
…   set")
1897            if cnfg["ctrl_type"] == "absolute":
1898                divider = (cnfg["enc_second"] - cnfg["enc_first"]) / length
1899                goto = cnfg["value"] / divider
1900                if cnfg["reverse_mode"] is True:
1901                    goto = length - goto
1902                goto = int(goto)
1903            elif cnfg["ctrl_type"] == "relative":
1904                if cnfg["enc_first"] == cnfg["value"]:
1905                    goto = selected - cnfg["steps"]
1906                elif cnfg["enc_second"] == cnfg["value"]:
1907                    goto = selected + cnfg["steps"]
1908            elif cnfg["ctrl_type"] == "on/off":
1909                if cnfg["enc_first"] == cnfg["value"] or cnfg["enc_second"] ==
…   cnfg["value"]:
1910                    if selected != 0 and selected != length - 1:
1911                        goto = length - 1
1912                    elif selected == 0:
1913                        goto = length - 1
1914                    else:
1915                        goto = 0
1916            elif cnfg["ctrl_type"] == "increment":
1917                goto = selected + cnfg["steps"]
1918            elif cnfg["ctrl_type"] == "decrement":
1919                goto = selected - cnfg["steps"]
1920            if(goto < 0):
1921                goto = 0
1922            if cnfg["tracks_scenes"] == "tracks":
1923                track_offset = goto
1924            elif cnfg["tracks_scenes"] == "scenes":
1925                scene_offset = goto
1926            try:
1927                self._session.set_offsets(track_offset, scene_offset)
1928                self._session._reassign_scenes()
1929                self.set_highlighting_session_component(self._session)
1930                self.refresh_state()
1931                self.call_script_reaction(active_mode, None,
…   'session_box_position')
1932            except:
```

```
1933            self.show_message("unable to move session box there.")
1934    def get_tracks_array(self):
1935        tracks_array = []
1936        count = 0
1937        for index in range(len(self.song().tracks)):
1938            tracks_array.append(self.song().tracks[count])
1939            count = count+1
1940        count = 0
1941        for index in range(len(self.song().return_tracks)):
1942            tracks_array.append(self.song().return_tracks[count])
1943            count = count+1
1944        tracks_array.append(self.song().master_track)
1945        return tracks_array
1946    def select_highlight(self, cnfg):
1947        tracks_scenes = cnfg["tracks_scenes"]
1948        if type(cnfg["highlight_number"]) == str:
1949            change_to = self.get_modifier_value(cnfg["highlight_number"])
1950        else:
1951            change_to = cnfg["highlight_number"]
1952        if tracks_scenes == "tracks":
1953            num_of_tracks_scenes = len(self.song().tracks) +
… len(self.song().return_tracks) + 1
1954        elif tracks_scenes == "scenes":
1955            num_of_tracks_scenes = len(self.song().scenes)
1956        if num_of_tracks_scenes >= change_to + 1:
1957            if tracks_scenes == "tracks":
1958                all_tracks_arr = self.get_tracks_array()
1959                self.song().view.selected_track =
… all_tracks_arr[change_to]
1960            elif tracks_scenes == "scenes":
1961                self.song().view.selected_scene =
… self.song().scenes[change_to]
1962        else:
1963            self.show_message("Your Session doesn't have " + str(change_to
… + 1) + " " + tracks_scenes)
1964    def scroll_active_device_bank(self, cnfg):
1965        device_id = cnfg["parent_device_id"]
1966        device = "device_id_" + str(device_id);
1967        active_bank = getattr(self, device + "_active_bank")
1968        banks = getattr(self, device + "_banks")
1969        length = len(banks) - 1
1970        if cnfg["ctrl_type"] == "absolute":
1971            divider = (cnfg["enc_second"] - cnfg["enc_first"]) / length
1972            if cnfg["reverse_mode"] is False:
1973                goto = cnfg["value"] / divider
1974            elif cnfg["reverse_mode"] is True:
1975                goto = (divider * length) / cnfg["value"]
1976            goto = int(goto)
```

```
1977            elif cnfg["ctrl_type"] == "relative":
1978                if cnfg["enc_first"] == cnfg["value"]:
1979                    goto = active_bank - 1
1980                elif cnfg["enc_second"] == cnfg["value"]:
1981                    goto = active_bank + 1
1982            elif cnfg["ctrl_type"] == "on/off":
1983                if cnfg["switch_type"] == "toggle":
1984                    if cnfg["enc_first"] == cnfg["value"]:
1985                        goto = length
1986                    elif cnfg["enc_second"] == cnfg["value"]:
1987                        goto = 0
1988                elif active_bank == length:
1989                    goto = 0
1990                else:
1991                    goto = length
1992            elif cnfg["ctrl_type"] == "increment":
1993                    goto = active_bank + 1
1994            elif cnfg["ctrl_type"] == "decrement":
1995                    goto = active_bank - 1
1996            if goto <= length and goto >= 0 and goto != active_bank:
1997                cnfg["banking_number"] = goto + 1
1998                self.change_active_device_bank(cnfg)
1999        def change_active_device_bank(self, cnfg):
2000            global active_mode
2001            device_id = cnfg["parent_device_id"]
2002            if type(cnfg["banking_number"]) == str:
2003                change_to_bank =
… self.get_modifier_value(cnfg["banking_number"])
2004            else:
2005                change_to_bank = cnfg["banking_number"] - 1
2006
2007            device = "device_id_" + str(device_id);
2008            bank_names = getattr(self, device + "_bank_names")
2009            length = len(bank_names) - 1;
2010            if change_to_bank <= length:
2011                setattr(self, device + "_active_bank", change_to_bank)
2012                self.bank_led_feedback(cnfg["parent_json_id"]);
2013                self.show_message("changed active bank to: " +
… bank_names[change_to_bank])
2014            elif change_to_bank > length:
2015                self.show_message("device does not have " + str(change_to_bank
… + 1) + " parameter banks set")
2016            fire_all_mode_feedback = getattr(self, "_mode" + active_mode +
… "_fire_all_feedback")
2017            fire_all_mode_feedback()
2018        def session_box(self, num_tracks, num_scenes, track_offset,
… scene_offset, clips, stop_all, stop_tracks, scene_launch, feedbackArr,
… combination_mode):
```

```
2019            self._session = SessionComponent(num_tracks, num_scenes)
2020            self._session.set_offsets(track_offset, scene_offset)
2021            self._session.add_offset_listener(self._on_session_offset_changes,
… identify_sender= False)
2022            self._session._reassign_scenes()
2023            self.set_highlighting_session_component(self._session)
2024            if clips:
2025                self._grid =
… ButtonMatrixElement(rows=[clips[(index*num_tracks):(index*num_tracks)+
… num_tracks] for index in range(num_scenes)])
2026                self._session.set_clip_launch_buttons(self._grid)
2027            if stop_all:
2028                self._session.set_stop_all_clips_button(stop_all)
2029            if stop_tracks:
2030                self._session.set_stop_track_clip_buttons(tuple(stop_tracks))
2031            if scene_launch:
2032                scene_launch_buttons =
… ButtonMatrixElement(rows=[scene_launch])
2033                self._session.set_scene_launch_buttons(scene_launch_buttons)
2034
… self._session.set_stop_clip_triggered_value(feedbackArr["StopClipTriggered
… "])
2035                self._session.set_stop_clip_value(feedbackArr["StopClip"])
2036            for scene_index in range(num_scenes):
2037                scene = self._session.scene(scene_index)
2038                scene.set_scene_value(feedbackArr["Scene"])
2039                scene.set_no_scene_value(feedbackArr["NoScene"])
2040                scene.set_triggered_value(feedbackArr["SceneTriggered"])
2041                for track_index in range(num_tracks):
2042                    clip_slot = scene.clip_slot(track_index)
2043
… clip_slot.set_triggered_to_play_value(feedbackArr["ClipTriggeredPlay"])
2044
… clip_slot.set_triggered_to_record_value(feedbackArr["ClipTriggeredRecord"]
… )
2045
… clip_slot.set_record_button_value(feedbackArr["RecordButton"])
2046                    clip_slot.set_stopped_value(feedbackArr["ClipStopped"])
2047                    clip_slot.set_started_value(feedbackArr["ClipStarted"])
2048
… clip_slot.set_recording_value(feedbackArr["ClipRecording"])
2049                for index in range(len(stop_tracks)):
2050                    stop_track_button = stop_tracks[index]
2051                    if feedbackArr["StopTrackPlaying"] and
… feedbackArr["StopTrackStopped"]:
2052
… stop_track_button.set_on_off_values(feedbackArr["StopTrackPlaying"],
… feedbackArr["StopTrackStopped"])
```

```
2053                if stop_all:
2054                    if feedbackArr["StopAllOn"] and feedbackArr["StopAllOff"]:
2055                        stop_all.set_on_off_values(feedbackArr["StopAllOn"],
…   feedbackArr["StopAllOff"])
2056            if combination_mode == "on":
2057                self._session._link()
2058            self.refresh_state()
2059        def _on_session_offset_changes(self):
2060            global active_mode
2061            updated_by = "_on_session_offset_changes"
2062            self._remove_custom_lom_listeners_handler(active_mode, updated_by)
2063            self._add_custom_lom_listeners_handler(active_mode, updated_by)
2064            self.log("sessionbox offset changed")
2065            try:
2066                remove_mode = getattr(self, "_remove_mode" + active_mode +
…   "_led_listeners")
2067                remove_mode()
2068                activate_mode = getattr(self, "_mode" + active_mode +
…   "_led_listeners")
2069                activate_mode()
2070            except:
2071                self.log("_on_session_offset_changes: could not remove / add
…   led_listeners")
2072                return;
2073        def remove_session_box(self, combination_mode):
2074            if hasattr(self, "_session"):
2075                self.current_track_offset = self._session._track_offset
2076                self.current_scene_offset = self._session._scene_offset
2077                self._session.set_clip_launch_buttons(None)
2078                self.set_highlighting_session_component(None)
2079                self._session.set_stop_all_clips_button(None)
2080                self._session.set_stop_track_clip_buttons(None)
2081                self._session.set_scene_launch_buttons(None)
2082                if combination_mode == "on":
2083                    self._session._unlink()
2084                self._session = None
2085        def scroll_modes(self, cnfg):
2086            controller = getattr(self, cnfg["attached_to"])
2087            cnfg["value"] = controller.cur_val
2088            if cnfg["ctrl_type"] == "absolute":
2089                divider = (cnfg["enc_second"] - cnfg["enc_first"]) /
…   (len(self.modes) - 1)
2090                if cnfg["reverse_mode"] is False:
2091                    goto = cnfg["value"] / divider
2092                elif cnfg["reverse_mode"] is True:
2093                    length = len(self.modes) - 1
2094                    goto = (divider * length) / cnfg["value"]
2095                goto = int(goto)
```

```
2096            elif cnfg["ctrl_type"] == "relative":
2097                if cnfg["enc_first"] == cnfg["value"]:
2098                    goto = self.key_num - 1
2099                elif cnfg["enc_second"] == cnfg["value"]:
2100                    goto = self.key_num + 1
2101            elif cnfg["ctrl_type"] == "on/off":
2102                if cnfg["enc_first"] == cnfg["value"]:
2103                    goto = len(self.modes) - 1
2104                elif cnfg["enc_second"] == cnfg["value"]:
2105                    goto = 0
2106            elif cnfg["ctrl_type"] == "increment":
2107                if cnfg["enc_first"] == cnfg["value"]:
2108                    goto = self.key_num + 1
2109            elif cnfg["ctrl_type"] == "decrement":
2110                if cnfg["enc_first"] == cnfg["value"]:
2111                    goto = self.key_num - 1
2112            if goto <= len(self.modes) and goto >= 0 and active_mode !=
…   self.modes[goto]:
2113                self.set_active_mode(self.modes[goto])
2114        def listening_to_tracks(self):
2115            global active_mode
2116            self.remove_listening_to_tracks()
2117            for index in range(len(self.song().tracks)):
2118                _track = self.song().tracks[index]
2119                if _track.can_be_armed and hasattr(self, "_mode" + active_mode
…   + "_arm_listener"):
2120                    _track.add_arm_listener(getattr(self, "_mode" +
…   active_mode + "_arm_listener"))
2121                if hasattr(self, "_mode" + active_mode + "_mute_listener"):
2122                    _track.add_mute_listener(getattr(self, "_mode" +
…   active_mode + "_mute_listener"))
2123                if hasattr(self, "_mode" + active_mode + "_solo_listener"):
2124                    _track.add_solo_listener(getattr(self, "_mode" +
…   active_mode + "_solo_listener"))
2125                if hasattr(self, "_mode" + active_mode + "_volume_listener"):
2126
…   _track.mixer_device.volume.add_value_listener(getattr(self, "_mode" +
…   active_mode + "_volume_listener"))
2127                if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2128
…   _track.mixer_device.panning.add_value_listener(getattr(self, "_mode" +
…   active_mode + "_panning_listener"))
2129                if hasattr(self, "_mode" + active_mode + "_send_listener"):
2130                    for send_index in range(len(_track.mixer_device.sends)):
2131
…   _track.mixer_device.sends[send_index].add_value_listener(getattr(self,
…   "_mode" + active_mode + "_send_listener"))
2132            for index in range(len(self.song().return_tracks)):
```

```
2133                _return_track = self.song().return_tracks[index]
2134                if hasattr(self, "_mode" + active_mode + "_mute_listener"):
2135                    _return_track.add_mute_listener(getattr(self, "_mode" +
…   active_mode + "_mute_listener"))
2136                if hasattr(self, "_mode" + active_mode + "_solo_listener"):
2137                    _return_track.add_solo_listener(getattr(self, "_mode" +
…   active_mode + "_solo_listener"))
2138                if hasattr(self, "_mode" + active_mode + "_volume_listener"):
2139
…   _return_track.mixer_device.volume.add_value_listener(getattr(self, "_mode"
…   + active_mode + "_volume_listener"))
2140                if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2141
…   _return_track.mixer_device.panning.add_value_listener(getattr(self,
…   "_mode" + active_mode + "_panning_listener"))
2142                if hasattr(self, "_mode" + active_mode + "_send_listener"):
2143                    for send_index in
…   range(len(_return_track.mixer_device.sends)):
2144
…   _return_track.mixer_device.sends[send_index].add_value_listener(getattr(
…   self, "_mode" + active_mode + "_send_listener"))
2145            _master = self.song().master_track
2146            if hasattr(self, "_mode" + active_mode + "_volume_listener"):
2147                _master.mixer_device.volume.add_value_listener(getattr(self,
…   "_mode" + active_mode + "_volume_listener"))
2148            if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2149                _master.mixer_device.panning.add_value_listener(getattr(self,
…   "_mode" + active_mode + "_panning_listener"))
2150        def remove_listening_to_tracks(self):
2151            global active_mode
2152            for index in range(len(self.song().tracks)):
2153                _track = self.song().tracks[index]
2154                if hasattr(self, "_mode" + active_mode + "_arm_listener"):
2155                    if _track.arm_has_listener(getattr(self, "_mode" +
…   active_mode + "_arm_listener")):
2156                        _track.remove_arm_listener(getattr(self, "_mode" +
…   active_mode + "_arm_listener"))
2157                if hasattr(self, "_mode" + active_mode + "_mute_listener"):
2158                    if _track.mute_has_listener(getattr(self, "_mode" +
…   active_mode + "_mute_listener")):
2159                        _track.remove_mute_listener(getattr(self, "_mode" +
…   active_mode + "_mute_listener"))
2160                if hasattr(self, "_mode" + active_mode + "_solo_listener"):
2161                    if _track.solo_has_listener(getattr(self, "_mode" +
…   active_mode + "_solo_listener")):
2162                        _track.remove_solo_listener(getattr(self, "_mode" +
…   active_mode + "_solo_listener"))
2163                if hasattr(self, "_mode" + active_mode + "_volume_listener"):
```

```
2164                 if
…   _track.mixer_device.volume.value_has_listener(getattr(self, "_mode" +
…   active_mode + "_volume_listener")):
2165
…   _track.mixer_device.volume.remove_value_listener(getattr(self, "_mode" +
…   active_mode + "_volume_listener"))
2166             if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2167                 if
…   _track.mixer_device.panning.value_has_listener(getattr(self, "_mode" +
…   active_mode + "_panning_listener")):
2168
…   _track.mixer_device.panning.remove_value_listener(getattr(self, "_mode" +
…   active_mode + "_panning_listener"))
2169             if hasattr(self, "_mode" + active_mode + "_send_listener"):
2170                 for send_index in range(len(_track.mixer_device.sends)):
2171                     if
…   _track.mixer_device.sends[send_index].value_has_listener(getattr(self,
…   "_mode" + active_mode + "_send_listener")):
2172
…   _track.mixer_device.sends[send_index].remove_value_listener(getattr(self,
…   "_mode" + active_mode + "_send_listener"))
2173         for index in range(len(self.song().return_tracks)):
2174             _return_track = self.song().return_tracks[index]
2175             if hasattr(self, "_mode" + active_mode + "_mute_listener"):
2176                 if _return_track.mute_has_listener(getattr(self, "_mode" +
…   active_mode + "_mute_listener")):
2177                     _return_track.remove_mute_listener(getattr(self,
…   "_mode" + active_mode + "_mute_listener"))
2178             if hasattr(self, "_mode" + active_mode + "_solo_listener"):
2179                 if _return_track.solo_has_listener(getattr(self, "_mode" +
…   active_mode + "_solo_listener")):
2180                     _return_track.remove_solo_listener(getattr(self,
…   "_mode" + active_mode + "_solo_listener"))
2181             if hasattr(self, "_mode" + active_mode + "_volume_listener"):
2182                 if
…   _return_track.mixer_device.volume.value_has_listener(getattr(self, "_mode"
…   + active_mode + "_volume_listener")):
2183
…   _return_track.mixer_device.volume.remove_value_listener(getattr(self,
…   "_mode" + active_mode + "_volume_listener"))
2184             if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2185                 if
…   _return_track.mixer_device.panning.value_has_listener(getattr(self,
…   "_mode" + active_mode + "_panning_listener")):
2186
…   _return_track.mixer_device.panning.remove_value_listener(getattr(self,
…   "_mode" + active_mode + "_panning_listener"))
2187             if hasattr(self, "_mode" + active_mode + "_send_listener"):
```

```
2188                 for send_index in
…  range(len(_return_track.mixer_device.sends)):
2189                     if
…  _return_track.mixer_device.sends[send_index].value_has_listener(getattr(
…  self, "_mode" + active_mode + "_send_listener")):
2190
…  _return_track.mixer_device.sends[send_index].remove_value_listener(getattr
…  (self, "_mode" + active_mode + "_send_listener"))
2191         _master = self.song().master_track
2192         if hasattr(self, "_mode" + active_mode + "_volume_listener"):
2193             if
…  _master.mixer_device.volume.value_has_listener(getattr(self, "_mode" +
…  active_mode + "_volume_listener")):
2194
…  _master.mixer_device.volume.remove_value_listener(getattr(self, "_mode" +
…  active_mode + "_volume_listener"))
2195         if hasattr(self, "_mode" + active_mode + "_panning_listener"):
2196             if
…  _master.mixer_device.panning.value_has_listener(getattr(self, "_mode" +
…  active_mode + "_panning_listener")):
2197
…  _master.mixer_device.panning.remove_value_listener(getattr(self, "_mode" +
…  active_mode + "_panning_listener"))
2198     def set_active_mode(self, activate_new_mode):
2199         global active_mode, prev_active_mode, modes
2200
2201         for number, mode_id in self.modes.items():
2202             if mode_id == activate_new_mode:
2203                 self.key_num = mode_id
2204         if(activate_new_mode == "Previous Mode"):
2205             if 'prev_active_mode' not in globals():
2206                 self.show_message("No previous mode is set yet.")
2207             else:
2208                 remove_mode = getattr(self, "_remove_mode" + active_mode)
2209                 remove_mode()
2210                 activate_new_mode = prev_active_mode
2211                 prev_active_mode = active_mode
2212                 self.call_script_reaction(prev_active_mode, None,
…  'mode_is_deactivated')
2213                 active_mode = activate_new_mode
2214                 mode_to_call = getattr(self, "_mode" + activate_new_mode)
2215                 mode_to_call()
2216                 self.call_script_reaction(activate_new_mode, None,
…  'mode_is_activated')
2217         else:
2218             if 'active_mode' in globals():
2219                 remove_mode = getattr(self, "_remove_mode" + active_mode)
2220                 remove_mode()
```

```
2221                prev_active_mode = active_mode
2222                self.call_script_reaction(prev_active_mode, None,
…   'mode_is_deactivated')
2223            active_mode = activate_new_mode
2224            mode_to_call = getattr(self, "_mode" + activate_new_mode)
2225            mode_to_call()
2226            self.call_script_reaction(activate_new_mode, None,
…   'mode_is_activated')
2227    def target_by_name(self, target_list, name):
2228        matches = filter(lambda t: t.display_name == name, target_list)
2229        if matches:
2230            return matches[0]
2231        return
2232    def _add_custom_lom_listeners_handler(self, mode_number,
…   updated_by=False):
2233        self.log("custom lom listeners refreshed")
2234        name_string = "_mode" + str(mode_number) + "_custom_lom_listeners"
2235        if hasattr(self, name_string):
2236            try:
2237                mode_to_call = getattr(self, name_string)
2238                mode_to_call(updated_by)
2239            except:
2240                self.log_message("csslog: unable to run " + name_string)
2241                pass
2242    def _remove_custom_lom_listeners_handler(self, mode_number,
…   updated_by=False):
2243        name_string = "_remove_mode" + str(mode_number) +
…   "_custom_lom_listeners"
2244        if hasattr(self, name_string):
2245            try:
2246                mode_to_call = getattr(self, name_string)
2247                mode_to_call(updated_by)
2248            except:
2249                self.log_message("csslog: unable to run " + name_string)
2250                pass
2251    def get_modifier_value(self, mod_name):
2252        return self.modifiers[mod_name]["value"]
2253    def set_modifier_value(self, mod_name, contents):
2254        global active_mode
2255        self.modifiers[mod_name]["value"] = contents
2256        self.call_script_reaction(active_mode, mod_name,
…   "modifier_was_updated")
2257    def call_script_reaction(self, mode_id, param2, reaction_name):
2258        one = "";
2259        two = "";
2260        three = "";
2261        if(mode_id!=None):
2262            one = "_mode_" + str(mode_id)
```

```
2263            if(param2!=None):
2264                two = "_" + str(param2)
2265            if(reaction_name!=None):
2266                three = "_" + str(reaction_name)
2267            reaction_method = one + two + three
2268            if hasattr(self, reaction_method):
2269                getattr(self, reaction_method)()
2270        def disconnect(self):
2271            self.call_script_reaction(None, None, 'script_is_disconnected')
2272            super(css_atcoperator_imported_1, self).disconnect()
```