Coverage for **/home/pi/banyan-bot-blue/banyan_assets/exp_pro_gateway.py** : 75%

198 statements | 154 run | 44 missing | 0 excluded | 11 partial

```
1  #!/usr/bin/env python3
2
3  """
4  exp_pro_gateway.py
5
6   Copyright (c) 2017-2019 Alan Yorinks All right reserved.
7
8   Python Banyan is free software; you can redistribute it and/or
9   modify it under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE
10  Version 3 as published by the Free Software Foundation; either
11  or (at your option) any later version.
12  This library is distributed in the hope that it will be useful,
13  but WITHOUT ANY WARRANTY; without even the implied warranty of
14  MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
15  General Public License for more details.
16
17  You should have received a copy of the GNU AFFERO GENERAL PUBLIC LICENSE
18  along with this library; if not, write to the Free Software
19  Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA
20
21 """
22 import argparse
23 import signal
24 import sys
25 import threading
26 import time
27
28 import explorerhat as eh
29 from python_banyan.gateway_base import GatewayBase
30
31
32 # noinspection PyMethodMayBeStatic,PyMethodMayBeStatic,SpellCheckingInspection,DuplicatedCode
```

```
33  class ExpProGateway(GatewayBase):
34      """
35      A OneGPIO type gateway for the Pimoroni Explorer Hat Pro
36      """
37
38      # noinspection PyDefaultArgument,PyRedundantParentheses
39      def __init__(self, *subscriber_list, **kwargs):
40          """
41          :param subscriber_list: a tuple or list of topics to be subscribed to
42          :param kwargs: contains the following parameters:
43
44          see the argparse section at the bottom of this file.
45          """
46          # initialize the parent
47          super(ExpProGateway, self).__init__(
48              subscriber_list=subscriber_list,
49              back_plane_ip_address=kwargs[
50                  'back_plane_ip_address'],
51              subscriber_port=kwargs[
52                  'subscriber_port'],
53              publisher_port=kwargs[
54                  'publisher_port'],
55              process_name=kwargs[
56                  'process_name'],
57              board_type=kwargs['board_type'],
58          )
59          # get threshold levels for the analog inputs
60          self.threshold = kwargs['threshold']
61
62          # format is different if provided as default values vs.
63          # user entered values. format appropriately.
64          if isinstance(self.threshold, list):                    64 ↛ 66
65              # convert string values to floats
66              self.threshold = [float(i) for i in self.threshold]
67          else:
68              self.threshold = [float(i) for i in self.threshold.split(',')]
69
70          if len(self.threshold) != 4:                            70 ↛ 71
71              raise RuntimeError('You must specify 4 thresholds')
72
```

```
73          self.enable_analog_input = kwargs['enable_analog_input']
74
75          # the explorer analog input code sends data
76          # too fast to process properly, so using
77          # the lock solves this issue.
78
79          self.the_lock = threading.RLock()
80
81          # get the report topic passed in
82          self.report_topic = (kwargs['report_topic'])
83
84          # A map of gpio pins to input channel numbers
85          self.gpio_input_pins = {23: 1, 22: 2, 24: 3, 25: 4}
86
87          # A map of digital output and led object to gpio pins
88          self.digital_output_pins = {4: eh.light.blue,
89                                      17: eh.light.yellow,
90                                      27: eh.light.red,
91                                      5: eh.light.green,
92                                      6: eh.output.one,
93                                      12: eh.output.two,
94                                      13: eh.output.three,
95                                      16: eh.output.four
96                                      }
97          # analog input objects
98          self.analog_input_objects = [eh.analog.one, eh.analog.two,
99                                       eh.analog.three, eh.analog.four]
100
101         # enable all of the digital inputs and assign
102         # a callback for when the pin goes high
103         eh.input.one.on_high(self.input_callback_high, 60)
104         eh.input.two.on_high(self.input_callback_high, 60)
105         eh.input.three.on_high(self.input_callback_high, 60)
106         eh.input.four.on_high(self.input_callback_high, 60)
107
108         # assign a callback for when a pin goes low
109         eh.input.one.on_low(self.input_callback_low, 60)
110         eh.input.two.on_low(self.input_callback_low, 60)
111         eh.input.three.on_low(self.input_callback_low, 60)
112         eh.input.four.on_low(self.input_callback_low, 60)
```

```
113
114         # enable touch pins with callback
115         eh.touch.pressed(self.touch_pressed)
116         eh.touch.released(self.touch_released)
117
118         # enable analog inputs if user selected to do so
119         # when instantiating ExpProGateway
120         if kwargs['enable_analog_input']:                                    120 ↛ 127
121             eh.analog.one.changed(self.analog_in1, self.threshold[0])
122             eh.analog.two.changed(self.analog_in2, self.threshold[1])
123             eh.analog.three.changed(self.analog_in3, self.threshold[2])
124             eh.analog.four.changed(self.analog_in4, self.threshold[3])
125
126         # start the banyan receive loop
127         try:
128             self.receive_loop()
129         except KeyboardInterrupt:
130             self.clean_up()
131             sys.exit(0)
132
133     def init_pins_dictionary(self):
134         """
135         We will not be using this for this gateway, so just pass.
136         """
137         pass
138
139     def touch_pressed(self, pin, state):
140         timestamp = self.get_time_stamp()
141
142         with self.the_lock:
143             payload = {'report': 'touch', 'pin': pin,
144                        'value': 1, 'timestamp': timestamp}
145             self.publish_payload(payload, self.report_topic)
146
147     def touch_released(self, pin, state):
148         timestamp = self.get_time_stamp()
149
150         with self.the_lock:
151             payload = {'report': 'touch', 'pin': pin,
152                        'value': 0, 'timestamp': timestamp}
```

```
153             self.publish_payload(payload, self.report_topic)
154
155     def input_callback_high(self, data):
156         """
157         This method is called by pigpio when it detects a change for
158         a digital input pin. A report is published reflecting
159         the change of pin state for the pin.
160         :param data: callback data
161         """
162         with self.the_lock:
163             timestamp = self.get_time_stamp()
164             # translate pin number
165             if data.pin in self.gpio_input_pins:
166                 pin = self.gpio_input_pins[data.pin]
167                 payload = {'report': 'digital_input', 'pin': pin,
168                            'value': 1, 'timestamp': timestamp}
169                 self.publish_payload(payload, self.report_topic)
170             else:
171                 raise RuntimeError('unknown input pin: ', data.pin)
172
173     def input_callback_low(self, data):
174         """
175         This method is called by pigpio when it detects a change for
176         a digital input pin. A report is published reflecting
177         the change of pin state for the pin.
178         :param data: callback data
179         """
180         with self.the_lock:
181             timestamp = self.get_time_stamp()
182             # translate pin number
183             if data.pin in self.gpio_input_pins:
184                 pin = self.gpio_input_pins[data.pin]
185                 payload = {'report': 'digital_input', 'pin': pin,
186                            'value': 0, 'timestamp': timestamp}
187                 self.publish_payload(payload, self.report_topic)
188             else:
189                 raise RuntimeError('unknown input pin: ', data.pin)
190
191     def analog_in1(self, data, value):
192         with self.the_lock:
```

```
193             # explorer sometimes sends bogus data - just ignore it
194             if value > 5.1:
195                 return
196             else:
197                 self.publish_analog_data(1, value)
198
199     def analog_in2(self, data, value):
200         with self.the_lock:
201             # explorer sometimes sends bogus data - just ignore it
202             if value > 5.1:
203                 return
204             else:
205                 self.publish_analog_data(2, value)
206
207     def analog_in3(self, data, value):
208         with self.the_lock:
209             # explorer sometimes sends bogus data - just ignore it
210             if value > 5.1:
211                 return
212             else:
213                 self.publish_analog_data(3, value)
214
215     def analog_in4(self, data, value):
216         with self.the_lock:
217             # explorer sometimes sends bogus data - just ignore it
218             if value > 5.1:
219                 return
220             else:
221                 self.publish_analog_data(4, value)
222
223     def publish_analog_data(self, pin, value):
224         # timestamp = self.get_time_stamp()
225         timestamp = self.get_time_stamp()
226         payload = {'report': 'analog_input', 'pin': pin,
227                    'value': value, 'timestamp': timestamp}
228         self.publish_payload(payload, self.report_topic)
229
230     def additional_banyan_messages(self, topic, payload):
231         """
232         This method will pass any messages not handled by this class to the
```

```
233         specific gateway class. Must be overwritten by the hardware gateway
234         class.
235         :param topic: message topic
236         :param payload: message payload
237         """
238 
239         # dc motor commands
240         if payload['command'] == 'dc_motor_forward':
241             speed = payload['speed'] * 100
242             if payload['motor'] == 1:
243                 eh.motor.one.speed(speed)
244             elif payload['motor'] == 2:                                  244 ↛ 247
245                 eh.motor.two.speed(speed)
246             else:
247                 raise RuntimeError('unknown motor number')
248 
249         elif payload['command'] == 'dc_motor_reverse':                   249 ↛ 259
250             speed = payload['speed'] * 100
251             if payload['motor'] == 1:
252                 eh.motor.one.speed(speed)
253             elif payload['motor'] == 2:                                  253 ↛ 256
254                 eh.motor.two.speed(speed)
255             else:
256                 raise RuntimeError('unknown motor')
257 
258         else:
259             raise RuntimeError('Unknown motor command')
260 
261     def analog_write(self, topic, payload):
262         """
263 
264         :param topic: message topic
265         :param payload: message payload
266         """
267         raise NotImplementedError
268 
269     def digital_read(self, pin):
270         """
271 
272         :param pin:
```

```
273         """
274         raise NotImplementedError
275 
276     def digital_write(self, topic, payload):
277         """
278         Set a signal, specified by its pin number in the payload,
279         to the value specified in the payload.
280 
281         Typical message: to_hardware {'command': 'digital_write', 'value': 0, 'pin': 0}
282 
283         :param topic: message topic
284         :param payload: message payload
285         """
286         # we will use the fade function
287         pin = payload['pin']
288         value = payload['value']
289         if pin in self.digital_output_pins:                                   289 ↛ 293
290             output_object = self.digital_output_pins[pin]
291             output_object.fade(0, value, .0001)
292         else:
293             raise RuntimeError('illegal digital output pin: ', pin)
294 
295     def disable_analog_reporting(self, topic, payload):
296         """
297 
298 
299         :param topic: message topic
300         :param payload: message payload
301         """
302         raise NotImplementedError
303 
304     def disable_digital_reporting(self, topic, payload):
305         """
306         :param topic: message topic
307         :param payload: message payload
308         """
309         raise NotImplementedError
310 
311     def enable_analog_reporting(self, topic, payload):
312         """
```

```
313         :param topic: message topic
314         :param payload: message payload
315         """
316         raise NotImplementedError
317 
318     def enable_digital_reporting(self, topic, payload):
319         """
320         :param topic: message topic
321         :param payload: message payload
322         """
323         raise NotImplementedError
324 
325     def i2c_read(self, topic, payload):
326         """
327         :param topic: message topic
328         :param payload: message payload
329         """
330         raise NotImplementedError
331 
332     def i2c_write(self, topic, payload):
333         """
334         :param topic: message topic
335         :param payload: message payload
336         """
337         raise NotImplementedError
338 
339     def play_tone(self, topic, payload):
340         """
341         :param topic: message topic
342         :param payload: message payload
343         """
344         raise NotImplementedError
345 
346     def pwm_write(self, topic, payload):
347         """
348         Set the specified drive pin to the specified pwm level
349 
350         Typical message:
351         to_hardware {'pin': 0, 'command': 'pwm_write', 'value': 0.41}
352 
```

```
353         :param topic: message topic
354         :param payload: message payload
355         """
356         raise NotImplementedError
357 
358     def servo_position(self, topic, payload):
359         """
360         Set servo angle for the specified servo
361 
362         Typical message:
363         to_hardware {'command': 'servo_position', 'position': 114, 'pin': 1}
364 
365         :param topic: message topic
366         :param payload: message payload
367         """
368         raise NotImplementedError
369 
370     def set_mode_analog_input(self, topic, payload):
371         """
372         Set a signal to analog input
373 
374         Typical message:
375         to_hardware {'command': 'set_mode_analog_input', 'pin': 5}
376 
377         :param topic: message topic
378         :param payload: message payload
379         """
380         pass
381 
382     def set_mode_digital_input(self, topic, payload):
383         """
384         This method sets a pin as digital input.
385         :param topic: message topic
386         :param payload: {"command": "set_mode_digital_input", "pin": “PIN”, "tag":”TAG” }
387         """
388         pass
389 
390     def set_mode_digital_input_pullup(self, topic, payload):
391         pass
392 
```

```
393     def set_mode_digital_output(self, topic, payload):
394         """
395         This method sets a pin as a digital output pin.
396         :param topic: message topic
397         :param payload: {"command": "set_mode_digital_output",
398                          "pin": PIN, "tag":”TAG” }
399         """
400         # self.pi.set_mode(payload['pin'], pigpio.OUTPUT)
401         pass
402 
403     def set_mode_pwm(self, topic, payload):
404         """
405          This method sets a GPIO pin capable of PWM for PWM operation.
406          :param topic: message topic
407          :param payload: {"command": "set_mode_pwm", "pin": “PIN”, "tag":”TAG” }
408          """
409         raise NotImplementedError
410 
411     def set_mode_i2c(self, topic, payload):
412         """
413         :param topic: message topic
414         :param payload: message payload
415         """
416         raise NotImplementedError
417 
418     def set_mode_servo(self, topic, payload):
419         """
420         {'command': 'set_mode_servo', 'pin': 1}
421 
422         :param topic: message topic
423         :param payload: message payload
424         """
425         pass
426 
427     def set_mode_sonar(self, topic, payload):
428         """
429         :param topic: message topic
430         :param payload: message payload
431         """
432         raise NotImplementedError
```

```
433
434     def set_mode_stepper(self, topic, payload):
435         """
436          - mode does not need to set - the stepper objects
437         are used directly.
438         :param topic: message topic
439         :param payload: message payload
440         """
441         raise NotImplementedError
442
443     def set_mode_tone(self, topic, payload):
444         """
445
446         :param topic: message topic
447         :param payload: message payload
448         """
449         raise NotImplementedError
450
451     def stepper_write(self, topic, payload):
452         """
453          - stepper objects are handled directly
454         :param topic: message topic
455         :param payload: message payload
456         """
457         raise NotImplementedError
458
459     def get_time_stamp(self):
460         """
461         Get the time of the pin change occurence
462         :return: Time stamp
463         """
464         t = time.time()
465         return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(t))
466
467
468 def exp_pro_gateway():
469     parser = argparse.ArgumentParser()
470     parser.add_argument("-a", dest="enable_analog_input", default="false",
471                         help="Set to True to enable analog input")
472     parser.add_argument("-b", dest="back_plane_ip_address", default="None",
```

```
473                         help="None or IP address used by Back Plane")
474     parser.add_argument("-d", dest="board_type", default="None",
475                         help="This parameter identifies the target GPIO "
476                              "device")
477     parser.add_argument("-l", dest="subscriber_list",
478                         default="to_hardware", nargs='+',
479                         help="Banyan topics space delimited: topic1 topic2 "
480                              "topic3")
481     parser.add_argument("-n", dest="process_name", default="ExpProGateway",
482                         help="Set process name in banner")
483     parser.add_argument("-p", dest="publisher_port", default='43124',
484                         help="Publisher IP port")
485     parser.add_argument("-r", dest="report_topic", default='report_from_hardware',
486                         help="Topic to publish reports from hardware.")
487     parser.add_argument("-s", dest="subscriber_port", default='43125',
488                         help="Subscriber IP port")
489     parser.add_argument("-t", dest="threshold", default="0.3, 0.3, 0.3, 0.3",
490                         nargs="+", help="A space delimited list of analog input sensitivities. Must contain 4 values '
491                                         "between 0.0 and 5.0")
492
493     args = parser.parse_args()
494     if args.back_plane_ip_address == 'None':                                    494 ↛ 496
495         args.back_plane_ip_address = None
496     if args.board_type == 'None':                                               496 ↛ 498
497         args.back_plane_ip_address = None
498     args.enable_analog_input = args.enable_analog_input.lower()
499     if args.enable_analog_input == 'true':                                      499 ↛ 502
500         args.enable_analog_input = True
501     else:
502         args.enable_analog_input = False
503     kw_options = {
504         'enable_analog_input': args.enable_analog_input,
505         'back_plane_ip_address': args.back_plane_ip_address,
506         'publisher_port': args.publisher_port,
507         'subscriber_port': args.subscriber_port,
508         'process_name': args.process_name,
509         # 'loop_time': float(args.loop_time),
510         'report_topic': args.report_topic,
511         'board_type': args.board_type,
512         'threshold': args.threshold}
```

```
513
514     try:
515         app = ExpProGateway(args.subscriber_list, **kw_options)
516     except KeyboardInterrupt:
517         sys.exit()
518
519     # noinspection PyUnusedLocal
520     def signal_handler(sig, frame):
521         print("Control-C detected. See you soon.")
522         app.clean__up()
523         sys.exit(0)
524
525     # listen for SIGINT
526     signal.signal(signal.SIGINT, signal_handler)
527     signal.signal(signal.SIGTERM, signal_handler)
528
529
530 if __name__ == '__main__':                                    530 ↛ exi
531     # replace with name of function you defined above
532     exp_pro_gateway()
```

« index coverage.py v4.5.4, created at 2019-09-03 17:52