Coverage for **exp_pro_gateway.py** : 73%

197 statements | 148 run | 49 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, threading.Thread):
```

```
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
47          # initialize the parent
48          super(ExpProGateway, self).__init__(
49              subscriber_list=subscriber_list,
50              back_plane_ip_address=kwargs[
51                  'back_plane_ip_address'],
52              subscriber_port=kwargs[
53                  'subscriber_port'],
54              publisher_port=kwargs[
55                  'publisher_port'],
56              process_name=kwargs[
57                  'process_name'],
58              board_type=kwargs['board_type'],
59          )
60          # get sensitivity levels for the analog inputs
61          sensitivity = kwargs['threshold']
62
63          # format is different if provided as default values vs.
64          # user entered values. format appropriately.
65          if isinstance(sensitivity, list):                                  65 ↛ 67
66              # convert string values to floats
67              sensitivity = [float(i) for i in sensitivity]
68          else:
69              sensitivity = [float(i) for i in sensitivity.split(',')]
70
71          if len(sensitivity) != 4:                                          71 ↛ 72
72              raise RuntimeError('You must specify 4 thresholds')
73
```

```
74          # the explorer analog input code sends data
75          # too fast to process properly, so using
76          # the lock solves this issue.
77
78          self.the_lock = threading.RLock()
79
80          # get the report topic passed in
81          self.report_topic = (kwargs['report_topic'])
82
83          # A map of gpio pins to input channel numbers
84          self.gpio_input_pins = {23: 1, 22: 2, 24: 3, 25: 4}
85
86          # A map of digital output and led object to gpio pins
87          self.digital_output_pins = {4: eh.light.blue,
88                                      17: eh.light.yellow,
89                                      27: eh.light.red,
90                                      5: eh.light.green,
91                                      6: eh.output.one,
92                                      12: eh.output.two,
93                                      13: eh.output.three,
94                                      16: eh.output.four
95                                      }
96
97          # enable all of the digital inputs and assign
98          # a callback for when the pin goes high
99          eh.input.one.on_high(self.input_callback_high)
100         eh.input.two.on_high(self.input_callback_high)
101         eh.input.three.on_high(self.input_callback_high)
102         eh.input.four.on_high(self.input_callback_high)
103
104         # assign a callback for when a pin goes low
105         eh.input.one.on_low(self.input_callback_low)
106         eh.input.two.on_low(self.input_callback_low)
107         eh.input.three.on_low(self.input_callback_low)
108         eh.input.four.on_low(self.input_callback_low)
109
110         # enable touch pins with callback
111         eh.touch.pressed(self.touch_pressed)
112         eh.touch.released(self.touch_released)
113
```

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

```
154         a digital input pin. A report is published reflecting
155         the change of pin state for the pin.
156         :param data: callback data
157         """
158         with self.the_lock:
159             timestamp = self.get_time_stamp()
160             # translate pin number
161             if data.pin in self.gpio_input_pins:
162                 pin = self.gpio_input_pins[data.pin]
163                 payload = {'report': 'digital_input', 'pin': pin,
164                            'value': 1, 'timestamp': timestamp}
165                 self.publish_payload(payload, self.report_topic)
166             else:
167                 raise RuntimeError('unknown input pin: ', data.pin)
168 
169     def input_callback_low(self, data):
170         """
171         This method is called by pigpio when it detects a change for
172         a digital input pin. A report is published reflecting
173         the change of pin state for the pin.
174         :param data: callback data
175         """
176         with self.the_lock:
177             timestamp = self.get_time_stamp()
178             # translate pin number
179             if data.pin in self.gpio_input_pins:
180                 pin = self.gpio_input_pins[data.pin]
181                 payload = {'report': 'digital_input', 'pin': pin,
182                            'value': 0, 'timestamp': timestamp}
183                 self.publish_payload(payload, self.report_topic)
184             else:
185                 raise RuntimeError('unknown input pin: ', data.pin)
186 
187     def analog_in1(self, data, value):
188         with self.the_lock:
189             # explorer sometimes sends bogus data - just ignore it
190             if value > 5.1:
191                 return
192             else:
193                 self.publish_analog_data(1, value)
```

```
194
195     def analog_in2(self, data, value):
196         with self.the_lock:
197             # explorer sometimes sends bogus data - just ignore it
198             if value > 5.1:
199                 return
200             else:
201                 self.publish_analog_data(2, value)
202
203     def analog_in3(self, data, value):
204         with self.the_lock:
205             # explorer sometimes sends bogus data - just ignore it
206             if value > 5.1:
207                 return
208             else:
209                 self.publish_analog_data(3, value)
210
211     def analog_in4(self, data, value):
212         with self.the_lock:
213             # explorer sometimes sends bogus data - just ignore it
214             if value > 5.1:
215                 return
216             else:
217                 self.publish_analog_data(4, value)
218
219     def publish_analog_data(self, pin, value):
220         # timestamp = self.get_time_stamp()
221         timestamp = self.get_time_stamp()
222         payload = {'report': 'analog_input', 'pin': pin,
223                    'value': value, 'timestamp': timestamp}
224         self.publish_payload(payload, self.report_topic)
225
226     def additional_banyan_messages(self, topic, payload):
227         """
228         This method will pass any messages not handled by this class to the
229         specific gateway class. Must be overwritten by the hardware gateway
230         class.
231         :param topic: message topic
232         :param payload: message payload
233         """
```

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

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

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

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

```
394         :param payload: {"command": "set_mode_digital_output",
395                          "pin": PIN, "tag":”TAG” }
396         """
397         # self.pi.set_mode(payload['pin'], pigpio.OUTPUT)
398         pass
399
400     def set_mode_pwm(self, topic, payload):
401         """
402          This method sets a GPIO pin capable of PWM for PWM operation.
403          :param topic: message topic
404          :param payload: {"command": "set_mode_pwm", "pin": “PIN”, "tag":”TAG” }
405          """
406         raise NotImplementedError
407
408     def set_mode_i2c(self, topic, payload):
409         """
410         :param topic: message topic
411         :param payload: message payload
412         """
413         raise NotImplementedError
414
415     def set_mode_servo(self, topic, payload):
416         """
417         {'command': 'set_mode_servo', 'pin': 1}
418
419         :param topic: message topic
420         :param payload: message payload
421         """
422         pass
423
424     def set_mode_sonar(self, topic, payload):
425         """
426         :param topic: message topic
427         :param payload: message payload
428         """
429         raise NotImplementedError
430
431     def set_mode_stepper(self, topic, payload):
432         """
433          - mode does not need to set - the stepper objects
```

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

```
474     parser.add_argument("-l", dest="subscriber_list",
475                         default="to_hardware", nargs='+',
476                         help="Banyan topics space delimited: topic1 topic2 "
477                              "topic3")
478     parser.add_argument("-n", dest="process_name", default="CrickitGateway",
479                         help="Set process name in banner")
480     parser.add_argument("-p", dest="publisher_port", default='43124',
481                         help="Publisher IP port")
482     parser.add_argument("-r", dest="report_topic", default='report_from_hardware',
483                         help="Topic to publish reports from hardware.")
484     parser.add_argument("-s", dest="subscriber_port", default='43125',
485                         help="Subscriber IP port")
486     parser.add_argument("-t", dest="threshold", default="4.99, 4.99, 4.99, 4.99",
487                         nargs="+", help="A space delimited list of analog input sensitivities. Must contain 4 values '
488                                         "between 0.0 and 5.0")
489
490     args = parser.parse_args()
491     if args.back_plane_ip_address == 'None':                                    491 ↛ 493
492         args.back_plane_ip_address = None
493     if args.board_type == 'None':                                               493 ↛ 495
494         args.back_plane_ip_address = None
495     args.enable_analog_input = args.enable_analog_input.lower()
496     if args.enable_analog_input == 'true':                                      496 ↛ 499
497         args.enable_analog_input = True
498     else:
499         args.enable_analog_input = False
500     kw_options = {
501         'enable_analog_input': args.enable_analog_input,
502         'back_plane_ip_address': args.back_plane_ip_address,
503         'publisher_port': args.publisher_port,
504         'subscriber_port': args.subscriber_port,
505         'process_name': args.process_name,
506         # 'loop_time': float(args.loop_time),
507         'report_topic': args.report_topic,
508         'board_type': args.board_type,
509         'threshold': args.threshold}
510
511     try:
512         app = ExpProGateway(args.subscriber_list, **kw_options)
513     except KeyboardInterrupt:
```

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

« index coverage.py v4.5.4, created at 2019-08-28 09:30