

## Coverage for **/home/pi/banyan-bot-blue/banyan\_assets/exp\_pro\_gateway.py** : 71%

180 statements   133 run   47 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 threshold levels for the analog inputs
61 |     self.threshold = kwargs['threshold']
62 |
63 |     # format is different if provided as default values vs.
64 |     # user entered values. format appropriately.
65 |     if isinstance(self.threshold, list):
66 |         # convert string values to floats
67 |         self.threshold = [float(i) for i in self.threshold]
68 |     else:
69 |         self.threshold = [float(i) for i in self.threshold.split(',')]
70 |
71 |     if len(self.threshold) != 4:
72 |         raise RuntimeError('You must specify 4 thresholds')
```

65 → 67

71 → 72

```
73
74 |     self.last_analog_value = [0.0, 0.0, 0.0, 0, 0]
75
76 |     # the explorer analog input code sends data
77 |     # too fast to process properly, so using
78 |     # the lock solves this issue.
79
80 |     # self.the_lock = threading.RLock()
81
82 |     # get the report topic passed in
83 |     self.report_topic = (kwargs['report_topic'])
84
85 |     # A map of gpio pins to input channel numbers
86 |     self.gpio_input_pins = {23: 1, 22: 2, 24: 3, 25: 4}
87
88 |     # A map of digital output and led object to gpio pins
89 |     self.digital_output_pins = {4: eh.light.blue,
90 |                                17: eh.light.yellow,
91 |                                27: eh.light.red,
92 |                                5: eh.light.green,
93 |                                6: eh.output.one,
94 |                                12: eh.output.two,
95 |                                13: eh.output.three,
96 |                                16: eh.output.four
97 |                                }
98 |     # analog input objects
99 |     self.analog_input_objects = [eh.analog.one, eh.analog.two,
100 |                                  eh.analog.three, eh.analog.four]
101
102 |     # enable all of the digital inputs and assign
103 |     # a callback for when the pin goes high
104 |     eh.input.one.on_high(self.input_callback_high, 30)
105 |     eh.input.two.on_high(self.input_callback_high, 30)
106 |     eh.input.three.on_high(self.input_callback_high, 30)
107 |     eh.input.four.on_high(self.input_callback_high, 30)
108
109 |     # assign a callback for when a pin goes low
110 |     eh.input.one.on_low(self.input_callback_low, 30)
111 |     eh.input.two.on_low(self.input_callback_low, 30)
112 |     eh.input.three.on_low(self.input_callback_low, 30)
```

```
113 |         eh.input.four.on_low(self.input_callback_low, 30)
114 |
115 |         # enable touch pins with callback
116 |         eh.touch.pressed(self.touch_pressed)
117 |         eh.touch.released(self.touch_released)
118 |
119 |         threading.Thread.__init__(self)
120 |         self.daemon = True
121 |
122 |         # start the thread to perform analog input polling
123 |         self.start()
124 |
125 |         # start the banyan receive loop
126 |         try:
127 |             self.receive_loop()
128 |         except KeyboardInterrupt:
129 |             self.clean_up()
130 |             sys.exit(0)
131 |
132 |     def init_pins_dictionary(self):
133 |         """
134 |         We will not be using this for this gateway, so just pass.
135 |         """
136 |         pass
137 |
138 |     def touch_pressed(self, pin, state):
139 |         timestamp = self.get_time_stamp()
140 |
141 |         payload = {'report': 'touch', 'pin': pin,
142 |                   'value': 1, 'timestamp': timestamp}
143 |         self.publish_payload(payload, self.report_topic)
144 |
145 |     def touch_released(self, pin, state):
146 |         timestamp = self.get_time_stamp()
147 |
148 |         payload = {'report': 'touch', 'pin': pin,
149 |                   'value': 0, 'timestamp': timestamp}
150 |         self.publish_payload(payload, self.report_topic)
151 |
152 |     def input_callback_high(self, data):
```

```
153     """
154     This method is called by pigpio when it detects a change for
155     a digital input pin. A report is published reflecting
156     the change of pin state for the pin.
157     :param data: callback data
158     """
159
160     timestamp = self.get_time_stamp()
161     # translate pin number
162     if data.pin in self.gpio_input_pins:
163         pin = self.gpio_input_pins[data.pin]
164         payload = {'report': 'digital_input', 'pin': pin,
165                   'value': 1, 'timestamp': timestamp}
166         self.publish_payload(payload, self.report_topic)
167     else:
168         raise RuntimeError('unknown input pin: ', data.pin)
169
170 def input_callback_low(self, data):
171     """
172     This method is called by pigpio when it detects a change for
173     a digital input pin. A report is published reflecting
174     the change of pin state for the pin.
175     :param data: callback data
176     """
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 run(self):
188     """
189     The input polling thread. Only report changes in input.
190
191     :return:
192     """
```

```
193     num_inputs = len(self.analog_input_objects)
194     while True:
195         for index, analog_input_object in enumerate(self.analog_input_objects):
196             value = analog_input_object.read()
197             if value < 6.5:
198                 if self.last_analog_value[index] != value:
199                     if abs(self.last_analog_value[index] - value) > self.threshold[index]:
200                         self.last_analog_value[index] = value
201                         timestamp = self.get_time_stamp()
202                         payload = {'report': 'analog_input', 'pin': index + 1,
203                                 'value':
204                                     value, 'timestamp': timestamp}
205                         self.publish_payload(payload, self.report_topic)
206
207     def additional_banyan_messages(self, topic, payload):
208         """
209         This method will pass any messages not handled by this class to the
210         specific gateway class. Must be overwritten by the hardware gateway
211         class.
212         :param topic: message topic
213         :param payload: message payload
214         """
215
216         # dc motor commands
217         if payload['command'] == 'dc_motor_forward':
218             speed = payload['speed'] * 100
219             if payload['motor'] == 1:
220                 eh.motor.one.speed(speed)
221             elif payload['motor'] == 2:
222                 eh.motor.two.speed(speed)
223             else:
224                 raise RuntimeError('unknown motor number')
225
226         elif payload['command'] == 'dc_motor_reverse':
227             speed = payload['speed'] * 100
228             if payload['motor'] == 1:
229                 eh.motor.one.speed(speed)
230             elif payload['motor'] == 2:
231                 eh.motor.two.speed(speed)
232             else:
```

```
233 |         raise RuntimeError('unknown motor')
234 |
235 |     else:
236 |         raise RuntimeError('Unknown motor command')
237 |
238 | def analog_write(self, topic, payload):
239 |     """
240 |
241 |     :param topic: message topic
242 |     :param payload: message payload
243 |     """
244 |     raise NotImplementedError
245 |
246 | def digital_read(self, pin):
247 |     """
248 |
249 |     :param pin:
250 |     """
251 |     raise NotImplementedError
252 |
253 | def digital_write(self, topic, payload):
254 |     """
255 |     Set a signal, specified by its pin number in the payload,
256 |     to the value specified in the payload.
257 |
258 |     Typical message: to_hardware {'command': 'digital_write', 'value': 0, 'pin': 0}
259 |
260 |     :param topic: message topic
261 |     :param payload: message payload
262 |     """
263 |     # we will use the fade function
264 |     pin = payload['pin']
265 |     value = payload['value']
266 |     if 0 <= value <= 100.0:
267 |         if pin in self.digital_output_pins:
268 |             output_object = self.digital_output_pins[pin]
269 |             output_object.pwm(eh.PULSE_FREQUENCY, value)
270 |         else:
271 |             raise RuntimeError('illegal digital output pin: ', pin)
272 |
```

266 → exi

267 → 271

```
273 |     def disable_analog_reporting(self, topic, payload):
274 |         """
275 |
276 |         :param topic: message topic
277 |         :param payload: message payload
278 |         """
279 |
280 |         raise NotImplementedError
281 |
282 |     def disable_digital_reporting(self, topic, payload):
283 |         """
284 |
285 |         :param topic: message topic
286 |         :param payload: message payload
287 |         """
288 |
289 |         raise NotImplementedError
290 |
291 |     def enable_analog_reporting(self, topic, payload):
292 |         """
293 |
294 |         :param topic: message topic
295 |         :param payload: message payload
296 |         """
297 |
298 |         raise NotImplementedError
299 |
300 |     def enable_digital_reporting(self, topic, payload):
301 |         """
302 |
303 |         :param topic: message topic
304 |         :param payload: message payload
305 |         """
306 |
307 |         raise NotImplementedError
308 |
309 |
310 |     def i2c_read(self, topic, payload):
311 |         """
312 |
313 |         :param topic: message topic
```



```
313         :param payload: message payload
314         """
315         raise NotImplementedError
316
317     def play_tone(self, topic, payload):
318         """
319         :param topic: message topic
320         :param payload: message payload
321         """
322         raise NotImplementedError
323
324     def pwm_write(self, topic, payload):
325         """
326         Set the specified drive pin to the specified pwm level
327
328         Typical message:
329         to_hardware {'pin': 0, 'command': 'pwm_write', 'value': 0.41}
330
331         :param topic: message topic
332         :param payload: message payload
333         """
334         raise NotImplementedError
335
336     def servo_position(self, topic, payload):
337         """
338         Set servo angle for the specified servo
339
340         Typical message:
341         to_hardware {'command': 'servo_position', 'position': 114, 'pin': 1}
342
343         :param topic: message topic
344         :param payload: message payload
345         """
346         raise NotImplementedError
347
348     def set_mode_analog_input(self, topic, payload):
349         """
350         Set a signal to analog input
351
352         Typical message:
```

```
353         to_hardware {'command': 'set_mode_analog_input', 'pin': 5}
354
355         :param topic: message topic
356         :param payload: message payload
357         """
358         pass
359
360     def set_mode_digital_input(self, topic, payload):
361         """
362         This method sets a pin as digital input.
363         :param topic: message topic
364         :param payload: {"command": "set_mode_digital_input", "pin": "PIN", "tag": "TAG" }
365         """
366         pass
367
368     def set_mode_digital_input_pullup(self, topic, payload):
369         pass
370
371     def set_mode_digital_output(self, topic, payload):
372         """
373         This method sets a pin as a digital output pin.
374         :param topic: message topic
375         :param payload: {"command": "set_mode_digital_output",
376                         "pin": PIN, "tag": "TAG" }
377         """
378         # self.pi.set_mode(payload['pin'], pigpio.OUTPUT)
379         pass
380
381     def set_mode_pwm(self, topic, payload):
382         """
383         This method sets a GPIO pin capable of PWM for PWM operation.
384         :param topic: message topic
385         :param payload: {"command": "set_mode_pwm", "pin": "PIN", "tag": "TAG" }
386         """
387         raise NotImplementedError
388
389     def set_mode_i2c(self, topic, payload):
390         """
391         :param topic: message topic
392         :param payload: message payload
```

```
393         """
394         raise NotImplementedError
395
396     def set_mode_servo(self, topic, payload):
397         """
398         {'command': 'set_mode_servo', 'pin': 1}
399
400         :param topic: message topic
401         :param payload: message payload
402         """
403         pass
404
405     def set_mode_sonar(self, topic, payload):
406         """
407         :param topic: message topic
408         :param payload: message payload
409         """
410         raise NotImplementedError
411
412     def set_mode_stepper(self, topic, payload):
413         """
414         - mode does not need to set - the stepper objects
415         are used directly.
416         :param topic: message topic
417         :param payload: message payload
418         """
419         raise NotImplementedError
420
421     def set_mode_tone(self, topic, payload):
422         """
423
424         :param topic: message topic
425         :param payload: message payload
426         """
427         raise NotImplementedError
428
429     def stepper_write(self, topic, payload):
430         """
431         - stepper objects are handled directly
432         :param topic: message topic
```

```
433         :param payload: message payload
434         """
435         raise NotImplementedError
436
437     def get_time_stamp(self):
438         """
439         Get the time of the pin change occurrence
440         :return: Time stamp
441         """
442         t = time.time()
443         return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(t))
444
445
446 def exp_pro_gateway():
447     parser = argparse.ArgumentParser()
448     parser.add_argument("-a", dest="enable_analog_input", default="false",
449                         help="Set to True to enable analog input")
450     parser.add_argument("-b", dest="back_plane_ip_address", default="None",
451                         help="None or IP address used by Back Plane")
452     parser.add_argument("-d", dest="board_type", default="None",
453                         help="This parameter identifies the target GPIO "
454                             "device")
455     parser.add_argument("-l", dest="subscriber_list",
456                         default="to_hardware", nargs='+',
457                         help="Banyan topics space delimited: topic1 topic2 "
458                             "topic3")
459     parser.add_argument("-n", dest="process_name", default="ExpProGateway",
460                         help="Set process name in banner")
461     parser.add_argument("-p", dest="publisher_port", default='43124',
462                         help="Publisher IP port")
463     parser.add_argument("-r", dest="report_topic", default='report_from_hardware',
464                         help="Topic to publish reports from hardware.")
465     parser.add_argument("-s", dest="subscriber_port", default='43125',
466                         help="Subscriber IP port")
467     parser.add_argument("-t", dest="threshold", default="0.3, 0.3, 0.3, 0.3",
468                         nargs="+", help="A space delimited list of analog input sensitivities. Must contain 4 values "
469                             "between 0.0 and 5.0")
470
471     args = parser.parse_args()
472     if args.back_plane_ip_address == 'None':
```

472 → 474

```
473 |         args.back_plane_ip_address = None
474 |     if args.board_type == 'None':                                474 → 476
475 |         args.back_plane_ip_address = None
476 |     args.enable_analog_input = args.enable_analog_input.lower()
477 |     if args.enable_analog_input == 'true':                      477 → 478
478 |         args.enable_analog_input = True
479 |     else:
480 |         args.enable_analog_input = False
481 |     kw_options = {
482 |         'enable_analog_input': args.enable_analog_input,
483 |         'back_plane_ip_address': args.back_plane_ip_address,
484 |         'publisher_port': args.publisher_port,
485 |         'subscriber_port': args.subscriber_port,
486 |         'process_name': args.process_name,
487 |         # 'loop_time': float(args.loop_time),
488 |         'report_topic': args.report_topic,
489 |         'board_type': args.board_type,
490 |         'threshold': args.threshold}
491 |
492 |     try:
493 |         app = ExpProGateway(args.subscriber_list, **kw_options)
494 |     except KeyboardInterrupt:
495 |         sys.exit()
496 |
497 |     # noinspection PyUnusedLocal
498 |     def signal_handler(sig, frame):
499 |         print("Control-C detected. See you soon.")
500 |         app.clean_up()
501 |         sys.exit(0)
502 |
503 |     # listen for SIGINT
504 |     signal.signal(signal.SIGINT, signal_handler)
505 |     signal.signal(signal.SIGTERM, signal_handler)
506 |
507 |
508 | if __name__ == '__main__':                                      508 → exi
509 |     # replace with name of function you defined above
510 |     exp_pro_gateway()
```

« *index*    *coverage.py v4.5.4, created at 2019-08-29 13:47*