Coverage for **/home/pi/bots-in-pieces-examples-master/banyan-bot-blue/banyan_assets/crickit_gateway.py** : 83%

236 statements | 200 run | 36 missing | 0 excluded | 14 partial

```
1   #!/usr/bin/env python3
2
3   """
4   crickit_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  from adafruit_crickit import crickit
29  from adafruit_motor import stepper
30  from python_banyan.gateway_base import GatewayBase
31
32
33  # noinspection PyMethodMayBeStatic,PyMethodMayBeStatic,SpellCheckingInspection
34  class CrickitGateway(GatewayBase, threading.Thread):
35      """
36      A OneGPIO type gateway for the Adafruit Crickit Hat for the Raspberry Pi
37      """
38
39      # noinspection PyDefaultArgument,PyRedundantParentheses
40      def __init__(self, *subscriber_list, **kwargs):
41          """
42          :param subscriber_list: a tuple or list of topics to be subscribed to
43          :param kwargs: contains the following parameters:
44
45          see the argparse section at the bottom of this file.
46          """
47
48          # virtual pin numbers for the cricket control objects
49          # this will serve as an index into the pins_dictionary
50
```

```
51          # pins  0 -  7: signals
52          self.SIGNAL_BASE = 0
53          self.SIGNAL_MAX = 7
54
55          self.TOUCH_BASE = 8
56          self.TOUCH_MAX = 11
57
58          # pins 12 - 15
59          self.DRIVE_BASE = 12
60          self.DRIVE_MAX = 15
61
62          # pins 16-17 servo
63          self.SERVO_BASE = 16
64          self.SERVO_MAX = 19
65
66          # pins 18 - 19 dc motor
67          self.MOTOR_BASE = 20
68          self.MOTOR_MAX = 21
69
70          # pins 20 - 21 stepper
71          self.STEPPER_BASE = 22
72          self.STEPPER_MAX = 23
73
74          # pin 22 neopixel
75          self.NEOPIXEL_BASE = 24
76
77          # initialize the parent
78          super(CrickitGateway, self).__init__(
79              subscriber_list=subscriber_list,
80              back_plane_ip_address=kwargs[
81                  'back_plane_ip_address'],
82              subscriber_port=kwargs[
83                  'subscriber_port'],
84              publisher_port=kwargs[
85                  'publisher_port'],
86              process_name=kwargs[
87                  'process_name'],
88              board_type=kwargs['board_type']
89          )
90
91          # get a seesaw object
92          self.ss = crickit.seesaw
93
94          threading.Thread.__init__(self)
95          self.daemon = True
96
97          # start the thread to perform input polling
98          self.start()
99
100         # start the banyan receive loop
101         try:
102             self.receive_loop()
103         except KeyboardInterrupt:
104             self.clean_up()
105             sys.exit(0)
106
107     def init_pins_dictionary(self):
108         """
109         Initialize the pins data structure. For this interface, it
```

```
110         is an array of dictionaries. To find the entry, use the pin
111         number as an index.
112 
113         Initialize the pins data structure. For this interface, it
114         is an array of dictionaries. To find the entry, use the pin
115         number as an index.
116 
117         pins  0 -  7: signals
118         pins  8 - 11: touch
119         pins 12 - 15: drive
120         pins 16 - 19: servo
121         pins 20 - 21: dc motor
122         pins 22 - 23 stepper
123         :return:
124 
125         """
126 
127         self.pins_dictionary = [
128             # SIGNALS - 0
129             {'crickit_object': crickit.SIGNAL1,
130              'modes': ['input', 'input_pullup', 'analog_input',
131                        'digital_output'],
132              'current_mode': None, 'enabled': False,
133              'last_value': None, 'callback': None
134              },
135 
136             {'crickit_object': crickit.SIGNAL2,
137              'modes': ['input', 'input_pullup', 'analog_input',
138                        'digital_output'],
139              'current_mode': None, 'enabled': False,
140              'last_value': None, 'callback': None
141              },
142 
143             {'crickit_object': crickit.SIGNAL3,
144              'modes': ['input', 'input_pullup', 'analog_input',
145                        'digital_output'],
146              'current_mode': None, 'enabled': False,
147              'last_value': None, 'callback': None
148              },
149 
150             {'crickit_object': crickit.SIGNAL4,
151              'modes': ['input', 'input_pullup', 'analog_input',
152                        'digital_output'],
153              'current_mode': None, 'enabled': False,
154              'last_value': None, 'callback': None
155              },
156 
157             {'crickit_object': crickit.SIGNAL5,
158              'modes': ['input', 'input_pullup', 'analog_input',
159                        'digital_output'],
160              'current_mode': None, 'enabled': False,
161              'last_value': None, 'callback': None
162              },
163 
164             {'crickit_object': crickit.SIGNAL6,
165              'modes': ['input', 'input_pullup', 'analog_input',
166                        'digital_output'],
167              'current_mode': None, 'enabled': False,
168              'last_value': None, 'callback': None
```

```
169              },
170
171             {'crickit_object': crickit.SIGNAL7,
172              'modes': ['input', 'input_pullup', 'analog_input',
173                        'digital_output'],
174              'current_mode': None, 'enabled': False,
175              'last_value': None, 'callback': None
176              },
177
178             {'crickit_object': crickit.SIGNAL8,
179              'modes': ['input', 'input_pullup', 'analog_input',
180                        'digital_output'],
181              'current_mode': None, 'enabled': False,
182              'last_value': None, 'callback': None
183              },
184
185             # TOUCH PADS - 8
186             {'crickit_object': crickit.touch_1,
187              'modes': ['input'],
188              'current_mode': None, 'enabled': False,
189              'last_value': None, 'callback': None
190              },
191
192             {'crickit_object': crickit.touch_2,
193              'modes': ['input'],
194              'current_mode': None, 'enabled': False,
195              'last_value': None, 'callback': None
196              },
197
198             {'crickit_object': crickit.touch_3,
199              'modes': ['input'],
200              'current_mode': None, 'enabled': False,
201              'last_value': None, 'callback': None
202              },
203
204             {'crickit_object': crickit.touch_4,
205              'modes': ['input'],
206              'current_mode': None, 'enabled': False,
207              'last_value': None, 'callback': None
208              },
209
210             # DRIVES - 12
211
212             {'crickit_object': crickit.drive_1,
213              'modes': ['pwm'], 'frequency': 1000,
214              'current_mode': None, 'enabled': False,
215              'last_value': None, 'callback': None
216              },
217
218             {'crickit_object': crickit.drive_2,
219              'modes': ['pwm'], 'frequency': 1000,
220              'current_mode': None, 'enabled': False,
221              'last_value': None, 'callback': None
222              },
223
224             {'crickit_object': crickit.drive_3,
225              'modes': ['pwm'], 'frequency': 1000,
226              'current_mode': None, 'enabled': False,
227              },
```

```
228
229             {'crickit_object': crickit.drive_4,
230              'modes': ['pwm'], 'frequency': 1000,
231              'current_mode': None, 'enabled': False,
232              },
233
234             # SERVOS - 16
235             {'crickit_object': crickit.servo_1,
236              'modes': ['servo'], 'frequency': 1000,
237              'current_mode': None, 'enabled': False,
238              'min_pulse': 500, 'max_pulse': 2500
239              },
240
241             {'crickit_object': crickit.servo_2,
242              'modes': ['servo'], 'frequency': 1000,
243              'current_mode': None, 'enabled': False,
244              'min_pulse': 500, 'max_pulse': 2500
245              },
246
247             {'crickit_object': crickit.servo_3,
248              'modes': ['servo'], 'frequency': 1000,
249              'current_mode': None, 'enabled': False,
250              'min_pulse': 500, 'max_pulse': 2500
251              },
252
253             {'crickit_object': crickit.servo_4,
254              'modes': ['servo'], 'frequency': 1000,
255              'current_mode': None, 'enabled': False,
256              'min_pulse': 500, 'max_pulse': 2500
257              },
258
259             # DC MOTORS - 20
260             {'crickit_object': crickit.dc_motor_1,
261              'modes': ['dc_motor'],
262              'current_mode': None, 'enabled': False,
263              },
264
265             {'crickit_object': crickit.dc_motor_2,
266              'modes': ['dc_motor'],
267              'current_mode': None, 'enabled': False,
268              },
269
270             # STEPPERS 23
271             {'crickit_object': crickit.stepper_motor,
272              'modes': ['stepper'],
273              'current_mode': None, 'enabled': False,
274              },
275
276             {'crickit_object': crickit.drive_stepper_motor,
277              'modes': ['drive_stepper'],
278              },
279         ]
280
281         # This is a workaround for an adafruit library anomaly -
282         # without these 2 lines, if a dc motor is connected,
283         # it will start spinning by itself.
284         stepper_motor = crickit.stepper_motor
285         stepper_motor.release()
286
```

```
287     def additional_banyan_messages(self, topic, payload):
288         """
289         This method will pass any messages not handled by this class to the
290         specific gateway class. Must be overwritten by the hardware gateway
291         class.
292         :param topic: message topic
293         :param payload: message payload
294         """
295 
296         # dc motor commands
297         if payload['command'] == 'dc_motor_forward' or payload['command'] == \
298                 'dc_motor_reverse':
299             self.dc_motor_move(payload['motor'] - 1, payload['speed'])
300 
301         # stepper commands
302         elif payload['command'] == 'stepper_drive_forward':
303             self.stepper_drive('drive', stepper.FORWARD, payload['steps'],
304                                payload['style'], payload['speed'])
305         elif payload['command'] == 'stepper_drive_reverse':
306             self.stepper_drive('drive', stepper.BACKWARD, payload['steps'],
307                                payload['style'], payload['speed'])
308         elif payload['command'] == 'stepper_forward':
309             self.stepper_drive('motor', stepper.FORWARD, payload['steps'],
310                                payload['style'], payload['speed'])
311         elif payload['command'] == 'stepper_reverse':
312             self.stepper_drive('motor', stepper.BACKWARD, payload['steps'],
313                                payload['style'], payload['speed'])
314         # pixel commands
315         elif payload['command'] == 'set_pixel':                                 315 ↛ 31
316             self.neo_pixel_control(payload['number_of_pixels'], payload['pixel_position'],
317                                    payload['red'], payload['green'], payload['blue'])
318         else:
319             raise RuntimeError('Unknown command: ', payload['command'])
320 
321     def stepper_drive(self, port, direction, number_of_steps, the_style, inter_step_delay):
322         """
323         This method control both drive and motor port steppers
324 
325         Typical command:
326         from_crickit_gui {'steps': '100', 'command': 'stepper_reverse',
327                           'speed': 0.0294, 'style': 'Double'}
328         from_crickit_gui {'steps': '100', 'command': 'stepper_drive_forward',
329                           'speed': 0.0, 'style': 'Single'}
330 
331         :param port: drive or motor port
332         :param direction: direction to move
333         :param number_of_steps: steps to move
334         :param the_style: Single, Double or Interleave
335         :param inter_step_delay: time between steps
336         """
337         if port == 'drive':
338             stepper_motor = crickit.drive_stepper_motor
339         else:
340             stepper_motor = crickit.stepper_motor
341 
342         if the_style == 'Double':
343             the_style = stepper.DOUBLE
344         elif the_style == 'Interleave':
345             the_style = stepper.INTERLEAVE
```

```
346         else:
347             the_style = stepper.SINGLE
348 
349         if direction == stepper.FORWARD:
350             for steps in range(int(number_of_steps)):
351                 stepper_motor.onestep(direction=stepper.FORWARD, style=the_style)
352                 time.sleep(inter_step_delay)
353         else:
354             for steps in range(int(number_of_steps)):
355                 stepper_motor.onestep(direction=stepper.BACKWARD, style=the_style)
356                 time.sleep(inter_step_delay)
357 
358     def neo_pixel_control(self, number_of_pixels, pixel_position, red, green, blue):
359         """
360         This is the neopixel handler
361 
362         Typical command:
363         from_crickit_gui {'number_of_pixels': 8, 'command': 'set_pixel', 'green': 128,
364                           'red': 121, 'pixel_position': 4, 'blue': 137}
365         :param number_of_pixels: pixels on ring or strip
366         :param pixel_position: pixel number to control - zero is the first
367         :param red: color value
368         :param green: color value
369         :param blue: color value
370         """
371         crickit.init_neopixel(number_of_pixels)
372 
373         crickit.neopixel.fill(0)
374 
375         # Assign to a variable to get a short name and to save time.
376         np = crickit.neopixel
377 
378         np[pixel_position] = (red, green, blue)
379 
380     def analog_write(self, topic, payload):
381         """
382         Not used for the crickit
383         :param topic: message topic
384         :param payload: message payload
385         """
386         raise NotImplementedError
387 
388     def digital_read(self, pin):
389         """
390         Not used for the crickit
391         :param pin:
392         """
393         raise NotImplementedError
394 
395     def digital_write(self, topic, payload):
396         """
397         Set a signal, specified by its pin number in the payload,
398         to the value specified in the payload.
399 
400         Typical message: from_crickit_gui {'command': 'digital_write', 'value': 0, 'pin': 0
401 
402         :param topic: message topic
403         :param payload: message payload
404         """
```

```
405         pin = payload['pin']
406         the_object = self.pins_dictionary[pin]['crickit_object']
407 
408         value = payload['value']
409         self.ss.digital_write(the_object, value)
410 
411     def disable_analog_reporting(self, topic, payload):
412         """
413         Not used for the crickit
414 
415         :param topic: message topic
416         :param payload: message payload
417         """
418         raise NotImplementedError
419 
420     def disable_digital_reporting(self, topic, payload):
421         """
422         Not used for the crickit
423 
424         :param topic: message topic
425         :param payload: message payload
426         """
427         raise NotImplementedError
428 
429     def enable_analog_reporting(self, topic, payload):
430         """
431         Not used for the crickit
432 
433         :param topic: message topic
434         :param payload: message payload
435         """
436         raise NotImplementedError
437 
438     def enable_digital_reporting(self, topic, payload):
439         """
440         Not used for the crickit
441 
442         :param topic: message topic
443         :param payload: message payload
444         """
445         raise NotImplementedError
446 
447     def i2c_read(self, topic, payload):
448         """
449         Not used for the crickit
450 
451         :param topic: message topic
452         :param payload: message payload
453         """
454         raise NotImplementedError
455 
456     def i2c_write(self, topic, payload):
457         """
458         Not used for the crickit
459 
460         :param topic: message topic
461         :param payload: message payload
462         """
463         raise NotImplementedError
```

```
464
465     def play_tone(self, topic, payload):
466         """
467         Not used for the crickit
468
469         :param topic: message topic
470         :param payload: message payload
471         """
472         raise NotImplementedError
473
474     def pwm_write(self, topic, payload):
475         """
476         Set the specified drive pin to the specified pwm level
477
478         Typical message:
479         from_crickit_gui {'pin': 0, 'command': 'pwm_write', 'value': 0.41}
480
481         :param topic: message topic
482         :param payload: message payload
483         """
484         pin = payload['pin'] + self.DRIVE_BASE
485         the_object = self.pins_dictionary[pin]['crickit_object']
486
487         the_value = payload['value']
488         the_object.fraction = the_value
489
490     def servo_position(self, topic, payload):
491         """
492         Set servo angle for the specified servo
493
494         Typical message:
495         from_crickit_gui {'command': 'servo_position', 'position': 114, 'pin': 1}
496
497         :param topic: message topic
498         :param payload: message payload
499         """
500         pin = payload['pin'] + self.SERVO_BASE
501         the_object = self.pins_dictionary[pin]['crickit_object']
502
503         the_angle = payload['position']
504         the_object.angle = the_angle
505
506     def set_mode_analog_input(self, topic, payload):
507         """
508         Set a signal to analog input
509
510         Typical message:
511         from_crickit_gui {'command': 'set_mode_analog_input', 'pin': 5}
512
513         :param topic: message topic
514         :param payload: message payload
515         """
516         pin = payload['pin']
517         if self.pins_dictionary[pin]['current_mode'] is not None:          517 ↛ 51
518             self.mode_previously_set_warning(pin, self.pins_dictionary[pin]['current_mode'])
519             return
520
521         self.pins_dictionary[pin]['current_mode'] = self.ANALOG_INPUT_MODE
522         self.pins_dictionary[pin]['enabled'] = True
```

```
523
524     def set_mode_digital_input(self, topic, payload):
525         """
526         Set a signal to digital input
527
528         Typical message: from_crickit_gui {'command': 'set_mode_digital_input', 'pin': 5}
529
530         :param topic: message topic
531         :param payload: message payload
532         """
533         pin = payload['pin']
534         if self.pins_dictionary[pin]['current_mode'] is not None:                    534 ↛ 53
535             self.mode_previously_set_warning(pin, self.pins_dictionary[pin]['current_mode']
536             return
537
538         self.pins_dictionary[pin]['enabled'] = True
539         self.pins_dictionary[pin]['last_value'] = 0
540         self.pins_dictionary[pin]['current_mode'] = self.DIGITAL_INPUT_MODE
541
542         # handle signals
543         if 0 <= pin <= 7:
544             the_object = self.pins_dictionary[pin]['crickit_object']
545             self.ss.pin_mode(the_object, self.ss.INPUT)
546
547         # handle the touch pins
548         if 8 <= pin <= 11:
549             self.pins_dictionary[pin]['enabled'] = True
550             self.pins_dictionary[pin]['last_value'] = 0
551
552     def set_mode_digital_input_pullup(self, topic, payload):
553         """
554         Set a signal to digital input pullup
555
556         Typical message:
557         from_crickit_gui {'command': 'set_mode_digital_input_pullup', 'pin': 5}
558         :param topic: message topic
559         :param payload: message payload
560         """
561
562         pin = payload['pin']
563         if self.pins_dictionary[pin]['current_mode'] is not None:                    563 ↛ 56
564             self.mode_previously_set_warning(pin, self.pins_dictionary[pin]['current_mode']
565             return
566
567         self.pins_dictionary[pin]['enabled'] = True
568         self.pins_dictionary[pin]['last_value'] = 0
569         self.pins_dictionary[pin]['current_mode'] = self.DIGITAL_INPUT_PULLUP_MODE
570
571         the_object = self.pins_dictionary[pin]['crickit_object']
572         self.ss.pin_mode(the_object, self.ss.INPUT_PULLUP)
573
574     def set_mode_digital_output(self, topic, payload):
575         """
576         Set a signal for digital output
577         Typical message: from_crickit_gui {'command': 'set_mode_digital_output', 'pin': 0}
578         :param topic: message topic
579         :param payload: message payload
580         """
581
```

```
582         pin = payload['pin']
583
584         if self.pins_dictionary[pin]['current_mode'] is not None:                  584 ↛ 59
585             if self.pins_dictionary[pin]['current_mode'] != self.DIGITAL_OUTPUT_MODE:  585 ↛ 59
586                 self.mode_previously_set_warning(pin,
587                                                  self.pins_dictionary[pin]['current_mode'])
588                 return
589
590         the_object = self.pins_dictionary[pin]['crickit_object']
591
592         self.ss.pin_mode(the_object, self.ss.OUTPUT)
593
594     def set_mode_i2c(self, topic, payload):
595         """
596         Not used for the crickit
597
598         :param topic: message topic
599         :param payload: message payload
600         """
601         raise NotImplementedError
602
603     def set_mode_pwm(self, topic, payload):
604         """
605         Set the frequency for a drive pin.
606
607         Typical message: from_crickit_gui {'pin': 0, 'command': 'set_mode_pwm'}
608
609         :param topic: message topic
610         :param payload: message payload
611         """
612         pin = payload['pin'] + self.DRIVE_BASE
613         if self.pins_dictionary[pin]['current_mode'] is not None:                  613 ↛ 61
614             if self.pins_dictionary[pin]['current_mode'] != self.PWM_OUTPUT_MODE:
615                 self.mode_previously_set_warning(pin,
616                                                  self.pins_dictionary[pin]['current_mode'])
617                 return
618
619         the_object = self.pins_dictionary[pin]['crickit_object']
620
621         the_object.frequency = 1000
622
623     def set_mode_servo(self, topic, payload):
624         """
625         Not used for crickit, but gui sends the following message:
626         from_crickit_gui {'command': 'set_mode_servo', 'pin': 1}
627
628         :param topic: message topic
629         :param payload: message payload
630         """
631         pass
632
633     def set_mode_sonar(self, topic, payload):
634         """
635         Not used for crickit
636         :param topic: message topic
637         :param payload: message payload
638         """
639         raise NotImplementedError
640
```

```
641     def set_mode_stepper(self, topic, payload):
642         """
643         Not used for crickit - mode does not need to set - the stepper objects
644         are used directly.
645         :param topic: message topic
646         :param payload: message payload
647         """
648         raise NotImplementedError
649 
650     def set_mode_tone(self, topic, payload):
651         """
652         Not used for crickit
653         :param topic: message topic
654         :param payload: message payload
655         """
656         raise NotImplementedError
657 
658     def stepper_write(self, topic, payload):
659         """
660         Not used for crickit - stepper objects are handled directly
661         :param topic: message topic
662         :param payload: message payload
663         """
664         raise NotImplementedError
665 
666     def dc_motor_move(self, motor, speed):
667         """
668         Set the specified motor to the specified speed.
669         Typical message: from_crickit_gui {'command': 'digital_write', 'value': 0, 'pin': 0
670 
671         :param motor: 1 or 2
672         :param speed: motor speed
673         """
674         motor_object = self.pins_dictionary[motor + self.MOTOR_BASE]['crickit_object']
675         motor_object.throttle = speed
676 
677     def run(self):
678         """
679         The input polling thread
680         :return:
681         """
682         topic = "to_crickit_gui"
683         while True:
684             # check the signal inputs
685             for pin in range(0, 8):
686                 the_object = self.pins_dictionary[pin]['crickit_object']
687                 if self.pins_dictionary[pin]['enabled']:
688                     if self.pins_dictionary[pin][
689                         'current_mode'] == self.DIGITAL_INPUT_MODE or \
690                             self.pins_dictionary[pin]['current_mode'] \
691                             == self.DIGITAL_INPUT_PULLUP_MODE:
692                         the_input = self.ss.digital_read(the_object)
693 
694                         if the_input != self.pins_dictionary[pin]['last_value']:
695                             self.pins_dictionary[pin]['last_value'] = the_input
696                             timestamp = self.get_time_stamp()
697                             payload = {'report': 'digital_input', 'pin': pin,
698                                        'value':
699                                            the_input, 'timestamp': timestamp}
```

```
700                             self.publish_payload(payload, topic)
701
702                     elif self.pins_dictionary[pin]['current_mode'] \                 702 ↛ 68
703                             == self.ANALOG_INPUT_MODE:
704                         the_input = self.ss.analog_read(the_object)
705                         if the_input != self.pins_dictionary[pin]['last_value']:
706                             self.pins_dictionary[pin]['last_value'] = the_input
707                             timestamp = self.get_time_stamp()
708                             payload = {'report': 'analog_input', 'pin': pin,
709                                        'value':
710                                            the_input, 'timestamp': timestamp}
711                             self.publish_payload(payload, topic)
712
713             # check the touch pins
714             for pin in range(8, 12):
715                 the_object = self.pins_dictionary[pin]['crickit_object']
716
717                 if self.pins_dictionary[pin]['enabled']:
718                     touch_value = the_object.value
719
720                     if touch_value != self.pins_dictionary[pin]['last_value']:
721                         self.pins_dictionary[pin]['last_value'] = touch_value
722                         timestamp = self.get_time_stamp()
723                         payload = {'report': 'digital_input', 'pin': pin,
724                                    'value':
725                                        touch_value, 'timestamp': timestamp}
726                         self.publish_payload(payload, topic)
727
728             time.sleep(.1)
729
730     def get_time_stamp(self):
731         t = time.time()
732         return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(t))
733
734     def mode_previously_set_warning(self, pin, mode):
735         print('Warning: Mode Not Set For Pin: ', pin)
736         if mode == self.DIGITAL_INPUT_MODE:                                         736 ↛ 73
737             print('Current Mode is Digital Input')
738         elif mode == self.DIGITAL_OUTPUT_MODE:                                      738 ↛ 73
739             print('Current Mode is Digital Input')
740         elif mode == self.ANALOG_INPUT_MODE:                                        740 ↛ 74
741             print('Current Mode is Analog Input')
742
743
744 def crickit_gateway():
745     parser = argparse.ArgumentParser()
746     parser.add_argument("-b", dest="back_plane_ip_address", default="None",
747                         help="None or IP address used by Back Plane")
748     parser.add_argument("-d", dest="board_type", default="None",
749                         help="This parameter identifies the target GPIO "
750                              "device")
751     parser.add_argument("-l", dest="subscriber_list",
752                         default="from_crickit_gui", nargs='+',
753                         help="Banyan topics space delimited: topic1 topic2 "
754                              "topic3")
755     parser.add_argument("-n", dest="process_name", default="CrickitGateway",
756                         help="Set process name in banner")
757     parser.add_argument("-p", dest="publisher_port", default='43124',
758                         help="Publisher IP port")
```

```
759     parser.add_argument("-s", dest="subscriber_port", default='43125',
760                         help="Subscriber IP port")
761     parser.add_argument("-t", dest="loop_time", default=".1",
762                         help="Event Loop Timer in seconds")
763 
764     args = parser.parse_args()
765     if args.back_plane_ip_address == 'None':                          765 ↛ 76
766         args.back_plane_ip_address = None
767     if args.board_type == 'None':                                     767 ↛ 76
768         args.back_plane_ip_address = None
769     kw_options = {
770         'back_plane_ip_address': args.back_plane_ip_address,
771         'publisher_port': args.publisher_port,
772         'subscriber_port': args.subscriber_port,
773         'process_name': args.process_name,
774         'loop_time': float(args.loop_time),
775         'board_type': args.board_type}
776 
777     try:
778         app = CrickitGateway(args.subscriber_list, **kw_options)
779     except KeyboardInterrupt:
780         sys.exit()
781 
782     # noinspection PyUnusedLocal
783     def signal_handler(sig, frame):
784         print("Control-C detected. See you soon.")
785         app.clean_up()
786         sys.exit(0)
787 
788     # listen for SIGINT
789     signal.signal(signal.SIGINT, signal_handler)
790     signal.signal(signal.SIGTERM, signal_handler)
791 
792 
793 if __name__ == '__main__':                                            793 ↛ ex
794     # replace with name of function you defined above
795     crickit_gateway()
```

« index coverage.py v4.5.3, created at 2019-08-15 16:13