Coverage for **/home/pi/banyan-bot-blue/banyan_assets/exp_pro_gateway.py** : 71%

180 statements | 133 run | 47 missing | 0 excluded | 11 partial

```
#!/usr/bin/env python3

"""
exp_pro_gateway.py

 Copyright (c) 2017-2019 Alan Yorinks All right reserved.

 Python Banyan is free software; you can redistribute it and/or
 modify it under the terms of the GNU AFFERO GENERAL PUBLIC LICENSE
 Version 3 as published by the Free Software Foundation; either
 or (at your option) any later version.
 This library is distributed in the hope that it will be useful,
 but WITHOUT ANY WARRANTY; without even the implied warranty of
 MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the GNU
 General Public License for more details.

 You should have received a copy of the GNU AFFERO GENERAL PUBLIC LICENSE
 along with this library; if not, write to the Free Software
 Foundation, Inc., 51 Franklin St, Fifth Floor, Boston, MA  02110-1301  USA

"""
import argparse
import signal
import sys
import threading
import time

import explorerhat as eh
from python_banyan.gateway_base import GatewayBase


# noinspection PyMethodMayBeStatic,PyMethodMayBeStatic,SpellCheckingInspection,DuplicatedCode
```

```
class ExpProGateway(GatewayBase, threading.Thread):
    """
    A OneGPIO type gateway for the Pimoroni Explorer Hat Pro
    """

    # noinspection PyDefaultArgument,PyRedundantParentheses
    def __init__(self, *subscriber_list, **kwargs):
        """
        :param subscriber_list: a tuple or list of topics to be subscribed to
        :param kwargs: contains the following parameters:

        see the argparse section at the bottom of this file.
        """

        # initialize the parent
        super(ExpProGateway, self).__init__(
            subscriber_list=subscriber_list,
            back_plane_ip_address=kwargs[
                'back_plane_ip_address'],
            subscriber_port=kwargs[
                'subscriber_port'],
            publisher_port=kwargs[
                'publisher_port'],
            process_name=kwargs[
                'process_name'],
            board_type=kwargs['board_type'],
        )
        # get threshold levels for the analog inputs
        self.threshold = kwargs['threshold']

        # format is different if provided as default values vs.
        # user entered values. format appropriately.
        if isinstance(self.threshold, list):                                65 ↛ 67
            # convert string values to floats
            self.threshold = [float(i) for i in self.threshold]
        else:
            self.threshold = [float(i) for i in self.threshold.split(',')]

        if len(self.threshold) != 4:                                        71 ↛ 72
            raise RuntimeError('You must specify 4 thresholds')

        self.last_analog_value = [0.0, 0.0, 0.0, 0, 0]

        # the explorer analog input code sends data
        # too fast to process properly, so using
        # the lock solves this issue.

        # self.the_lock = threading.RLock()

        # get the report topic passed in
        self.report_topic = (kwargs['report_topic'])

        # A map of gpio pins to input channel numbers
        self.gpio_input_pins = {23: 1, 22: 2, 24: 3, 25: 4}

        # A map of digital output and led object to gpio pins
        self.digital_output_pins = {4: eh.light.blue,
                                    17: eh.light.yellow,
                                    27: eh.light.red,
                                    5: eh.light.green,
                                    6: eh.output.one,
                                    12: eh.output.two,
                                    13: eh.output.three,
                                    16: eh.output.four
                                    }
        # analog input objects
        self.analog_input_objects = [eh.analog.one, eh.analog.two,
                                     eh.analog.three, eh.analog.four]

        # enable all of the digital inputs and assign
        # a callback for when the pin goes high
        eh.input.one.on_high(self.input_callback_high, 30)
        eh.input.two.on_high(self.input_callback_high, 30)
        eh.input.three.on_high(self.input_callback_high, 30)
        eh.input.four.on_high(self.input_callback_high, 30)

        # assign a callback for when a pin goes low
        eh.input.one.on_low(self.input_callback_low, 30)
        eh.input.two.on_low(self.input_callback_low, 30)
        eh.input.three.on_low(self.input_callback_low, 30)
        eh.input.four.on_low(self.input_callback_low, 30)

        # enable touch pins with callback
        eh.touch.pressed(self.touch_pressed)
        eh.touch.released(self.touch_released)

        threading.Thread.__init__(self)
        self.daemon = True

        # start the thread to perform analog input polling
        self.start()

        # start the banyan receive loop
        try:
            self.receive_loop()
        except KeyboardInterrupt:
            self.clean_up()
            sys.exit(0)

    def init_pins_dictionary(self):
        """
        We will not be using this for this gateway, so just pass.
        """
        pass

    def touch_pressed(self, pin, state):
        timestamp = self.get_time_stamp()

        payload = {'report': 'touch', 'pin': pin,
                   'value': 1, 'timestamp': timestamp}
        self.publish_payload(payload, self.report_topic)

    def touch_released(self, pin, state):
        timestamp = self.get_time_stamp()

        payload = {'report': 'touch', 'pin': pin,
                   'value': 0, 'timestamp': timestamp}
        self.publish_payload(payload, self.report_topic)

    def input_callback_high(self, data):
        """
        This method is called by pigpio when it detects a change for
        a digital input pin. A report is published reflecting
        the change of pin state for the pin.
        :param data: callback data
        """

        timestamp = self.get_time_stamp()
        # translate pin number
        if data.pin in self.gpio_input_pins:
            pin = self.gpio_input_pins[data.pin]
            payload = {'report': 'digital_input', 'pin': pin,
                       'value': 1, 'timestamp': timestamp}
            self.publish_payload(payload, self.report_topic)
        else:
            raise RuntimeError('unknown input pin: ', data.pin)

    def input_callback_low(self, data):
        """
        This method is called by pigpio when it detects a change for
        a digital input pin. A report is published reflecting
        the change of pin state for the pin.
        :param data: callback data
        """
        timestamp = self.get_time_stamp()
        # translate pin number
        if data.pin in self.gpio_input_pins:
            pin = self.gpio_input_pins[data.pin]
            payload = {'report': 'digital_input', 'pin': pin,
                       'value': 0, 'timestamp': timestamp}
            self.publish_payload(payload, self.report_topic)
        else:
            raise RuntimeError('unknown input pin: ', data.pin)

    def run(self):
        """
        The input polling thread. Only report changes in input.

        :return:
        """
```

```
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:                                          221 ↛ 224
222                 eh.motor.two.speed(speed)
223             else:
224                 raise RuntimeError('unknown motor number')
225 
226         elif payload['command'] == 'dc_motor_reverse':                           226 ↛ 23
227             speed = payload['speed'] * 100
228             if payload['motor'] == 1:
229                 eh.motor.one.speed(speed)
230             elif payload['motor'] == 2:                                          230 ↛ 23
231                 eh.motor.two.speed(speed)
232             else:
```

```
                raise RuntimeError('unknown motor')

        else:
            raise RuntimeError('Unknown motor command')

    def analog_write(self, topic, payload):
        """

        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def digital_read(self, pin):
        """

        :param pin:
        """
        raise NotImplementedError

    def digital_write(self, topic, payload):
        """
        Set a signal, specified by its pin number in the payload,
        to the value specified in the payload.

        Typical message: to_hardware {'command': 'digital_write', 'value': 0, 'pin': 0}

        :param topic: message topic
        :param payload: message payload
        """
        # we will use the fade function
        pin = payload['pin']
        value = payload['value']
        if 0 <= value <= 100.0:                                                            266 ↛ exi
            if pin in self.digital_output_pins:                                            267 ↛ 271
                output_object = self.digital_output_pins[pin]
                output_object.pwm(eh.PULSE_FREQUENCY, value)
            else:
                raise RuntimeError('illegal digital output pin: ', pin)

    def disable_analog_reporting(self, topic, payload):
        """


        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def disable_digital_reporting(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def enable_analog_reporting(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def enable_digital_reporting(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def i2c_read(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def i2c_write(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def play_tone(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def pwm_write(self, topic, payload):
        """
        Set the specified drive pin to the specified pwm level

        Typical message:
        to_hardware {'pin': 0, 'command': 'pwm_write', 'value': 0.41}

        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def servo_position(self, topic, payload):
        """
        Set servo angle for the specified servo

        Typical message:
        to_hardware {'command': 'servo_position', 'position': 114, 'pin': 1}

        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def set_mode_analog_input(self, topic, payload):
        """
        Set a signal to analog input

        Typical message:
```

```
        to_hardware {'command': 'set_mode_analog_input', 'pin': 5}

        :param topic: message topic
        :param payload: message payload
        """
        pass

    def set_mode_digital_input(self, topic, payload):
        """
        This method sets a pin as digital input.
        :param topic: message topic
        :param payload: {"command": "set_mode_digital_input", "pin": “PIN”, "tag":”TAG” }
        """
        pass

    def set_mode_digital_input_pullup(self, topic, payload):
        pass

    def set_mode_digital_output(self, topic, payload):
        """
        This method sets a pin as a digital output pin.
        :param topic: message topic
        :param payload: {"command": "set_mode_digital_output",
                         "pin": PIN, "tag":”TAG” }
        """
        # self.pi.set_mode(payload['pin'], pigpio.OUTPUT)
        pass

    def set_mode_pwm(self, topic, payload):
        """
         This method sets a GPIO pin capable of PWM for PWM operation.
         :param topic: message topic
         :param payload: {"command": "set_mode_pwm", "pin": “PIN”, "tag":”TAG” }
         """
        raise NotImplementedError

    def set_mode_i2c(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def set_mode_servo(self, topic, payload):
        """
        {'command': 'set_mode_servo', 'pin': 1}

        :param topic: message topic
        :param payload: message payload
        """
        pass

    def set_mode_sonar(self, topic, payload):
        """
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def set_mode_stepper(self, topic, payload):
        """
         - mode does not need to set - the stepper objects
        are used directly.
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def set_mode_tone(self, topic, payload):
        """

        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def stepper_write(self, topic, payload):
        """
         - stepper objects are handled directly
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def get_time_stamp(self):
        """
        Get the time of the pin change occurence
        :return: Time stamp
        """
        t = time.time()
        return time.strftime('%Y-%m-%d %H:%M:%S', time.localtime(t))


def exp_pro_gateway():
    parser = argparse.ArgumentParser()
    parser.add_argument("-a", dest="enable_analog_input", default="false",
                        help="Set to True to enable analog input")
    parser.add_argument("-b", dest="back_plane_ip_address", default="None",
                        help="None or IP address used by Back Plane")
    parser.add_argument("-d", dest="board_type", default="None",
                        help="This parameter identifies the target GPIO "
                             "device")
    parser.add_argument("-l", dest="subscriber_list",
                        default="to_hardware", nargs='+',
                        help="Banyan topics space delimited: topic1 topic2 "
                             "topic3")
    parser.add_argument("-n", dest="process_name", default="ExpProGateway",
                        help="Set process name in banner")
    parser.add_argument("-p", dest="publisher_port", default='43124',
                        help="Publisher IP port")
    parser.add_argument("-r", dest="report_topic", default='report_from_hardware',
                        help="Topic to publish reports from hardware.")
    parser.add_argument("-s", dest="subscriber_port", default='43125',
                        help="Subscriber IP port")
    parser.add_argument("-t", dest="threshold", default="0.3, 0.3, 0.3, 0.3",
                        nargs="+", help="A space delimited list of analog input sensitivities. Must contain 4 values '
                                        "between 0.0 and 5.0")

    args = parser.parse_args()
    if args.back_plane_ip_address == 'None':                                                      472 ↛ 474
```

```
            args.back_plane_ip_address = None
        if args.board_type == 'None':                                                  474 ↛ 476
            args.back_plane_ip_address = None
        args.enable_analog_input = args.enable_analog_input.lower()
        if args.enable_analog_input == 'true':                                         477 ↛ 478
            args.enable_analog_input = True
        else:
            args.enable_analog_input = False
        kw_options = {
            'enable_analog_input': args.enable_analog_input,
            'back_plane_ip_address': args.back_plane_ip_address,
            'publisher_port': args.publisher_port,
            'subscriber_port': args.subscriber_port,
            'process_name': args.process_name,
            # 'loop_time': float(args.loop_time),
            'report_topic': args.report_topic,
            'board_type': args.board_type,
            'threshold': args.threshold}

        try:
            app = ExpProGateway(args.subscriber_list, **kw_options)
        except KeyboardInterrupt:
            sys.exit()

        # noinspection PyUnusedLocal
        def signal_handler(sig, frame):
            print("Control-C detected. See you soon.")
            app.clean_up()
            sys.exit(0)

        # listen for SIGINT
        signal.signal(signal.SIGINT, signal_handler)
        signal.signal(signal.SIGTERM, signal_handler)


    if __name__ == '__main__':                                                         508 ↛ exi
        # replace with name of function you defined above
        exp_pro_gateway()
```

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