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

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

"""
crickit_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

from adafruit_crickit import crickit
from adafruit_motor import stepper
from python_banyan.gateway_base import GatewayBase


# noinspection PyMethodMayBeStatic,PyMethodMayBeStatic,SpellCheckingInspection
class CrickitGateway(GatewayBase, threading.Thread):
    """
    A OneGPIO type gateway for the Adafruit Crickit Hat for the Raspberry Pi
    """

    # 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.
        """

        # virtual pin numbers for the cricket control objects
        # this will serve as an index into the pins_dictionary

        # pins  0 -  7: signals
        self.SIGNAL_BASE = 0
        self.SIGNAL_MAX = 7

        self.TOUCH_BASE = 8
        self.TOUCH_MAX = 11

        # pins 12 - 15
        self.DRIVE_BASE = 12
        self.DRIVE_MAX = 15

        # pins 16-17 servo
        self.SERVO_BASE = 16
        self.SERVO_MAX = 19

        # pins 18 - 19 dc motor
        self.MOTOR_BASE = 20
        self.MOTOR_MAX = 21

        # pins 20 - 21 stepper
        self.STEPPER_BASE = 22
        self.STEPPER_MAX = 23

        # pin 22 neopixel
        self.NEOPIXEL_BASE = 24

        # initialize the parent
        super(CrickitGateway, 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 a seesaw object
        self.ss = crickit.seesaw

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

        # start the thread to perform 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):
        """
        Initialize the pins data structure. For this interface, it
        is an array of dictionaries. To find the entry, use the pin
        number as an index.

        Initialize the pins data structure. For this interface, it
        is an array of dictionaries. To find the entry, use the pin
        number as an index.

        pins  0 -  7: signals
        pins  8 - 11: touch
        pins 12 - 15: drive
        pins 16 - 19: servo
        pins 20 - 21: dc motor
        pins 22 - 23 stepper
        :return:

        """

        self.pins_dictionary = [
            # SIGNALS - 0
            {'crickit_object': crickit.SIGNAL1,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
             },

            {'crickit_object': crickit.SIGNAL2,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
             },

            {'crickit_object': crickit.SIGNAL3,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
             },

            {'crickit_object': crickit.SIGNAL4,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
             },

            {'crickit_object': crickit.SIGNAL5,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
             },

            {'crickit_object': crickit.SIGNAL6,
             'modes': ['input', 'input_pullup', 'analog_input',
                       'digital_output'],
             'current_mode': None, 'enabled': False,
             'last_value': None, 'callback': None
```

```
                 },

                {'crickit_object': crickit.SIGNAL7,
                 'modes': ['input', 'input_pullup', 'analog_input',
                           'digital_output'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.SIGNAL8,
                 'modes': ['input', 'input_pullup', 'analog_input',
                           'digital_output'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                # TOUCH PADS - 8
                {'crickit_object': crickit.touch_1,
                 'modes': ['input'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.touch_2,
                 'modes': ['input'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.touch_3,
                 'modes': ['input'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.touch_4,
                 'modes': ['input'],
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                # DRIVES - 12

                {'crickit_object': crickit.drive_1,
                 'modes': ['pwm'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.drive_2,
                 'modes': ['pwm'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'last_value': None, 'callback': None
                 },

                {'crickit_object': crickit.drive_3,
                 'modes': ['pwm'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 },

                {'crickit_object': crickit.drive_4,
                 'modes': ['pwm'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 },

                # SERVOS - 16
                {'crickit_object': crickit.servo_1,
                 'modes': ['servo'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'min_pulse': 500, 'max_pulse': 2500
                 },

                {'crickit_object': crickit.servo_2,
                 'modes': ['servo'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'min_pulse': 500, 'max_pulse': 2500
                 },

                {'crickit_object': crickit.servo_3,
                 'modes': ['servo'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'min_pulse': 500, 'max_pulse': 2500
                 },

                {'crickit_object': crickit.servo_4,
                 'modes': ['servo'], 'frequency': 1000,
                 'current_mode': None, 'enabled': False,
                 'min_pulse': 500, 'max_pulse': 2500
                 },

                # DC MOTORS - 20
                {'crickit_object': crickit.dc_motor_1,
                 'modes': ['dc_motor'],
                 'current_mode': None, 'enabled': False,
                 },

                {'crickit_object': crickit.dc_motor_2,
                 'modes': ['dc_motor'],
                 'current_mode': None, 'enabled': False,
                 },

                # STEPPERS 23
                {'crickit_object': crickit.stepper_motor,
                 'modes': ['stepper'],
                 'current_mode': None, 'enabled': False,
                 },

                {'crickit_object': crickit.drive_stepper_motor,
                 'modes': ['drive_stepper'],
                 },
            ]

            # This is a workaround for an adafruit library anomaly -
            # without these 2 lines, if a dc motor is connected,
            # it will start spinning by itself.
            stepper_motor = crickit.stepper_motor
            stepper_motor.release()

```

```
    def additional_banyan_messages(self, topic, payload):
        """
        This method will pass any messages not handled by this class to the
        specific gateway class. Must be overwritten by the hardware gateway
        class.
        :param topic: message topic
        :param payload: message payload
        """

        # dc motor commands
        if payload['command'] == 'dc_motor_forward' or payload['command'] == \
                'dc_motor_reverse':
            self.dc_motor_move(payload['motor'] - 1, payload['speed'])

        # stepper commands
        elif payload['command'] == 'stepper_drive_forward':
            self.stepper_drive('drive', stepper.FORWARD, payload['steps'],
                               payload['style'], payload['speed'])
        elif payload['command'] == 'stepper_drive_reverse':
            self.stepper_drive('drive', stepper.BACKWARD, payload['steps'],
                               payload['style'], payload['speed'])
        elif payload['command'] == 'stepper_forward':
            self.stepper_drive('motor', stepper.FORWARD, payload['steps'],
                               payload['style'], payload['speed'])
        elif payload['command'] == 'stepper_reverse':
            self.stepper_drive('motor', stepper.BACKWARD, payload['steps'],
                               payload['style'], payload['speed'])
        # pixel commands
        elif payload['command'] == 'set_pixel':                              315 ↛ 31
            self.neo_pixel_control(payload['number_of_pixels'], payload['pixel_position'],
                                   payload['red'], payload['green'], payload['blue'])
        else:
            raise RuntimeError('Unknown command: ', payload['command'])

    def stepper_drive(self, port, direction, number_of_steps, the_style, inter_step_delay):
        """
        This method control both drive and motor port steppers

        Typical command:
        from_crickit_gui {'steps': '100', 'command': 'stepper_reverse',
                          'speed': 0.0294, 'style': 'Double'}
        from_crickit_gui {'steps': '100', 'command': 'stepper_drive_forward',
                          'speed': 0.0, 'style': 'Single'}

        :param port: drive or motor port
        :param direction: direction to move
        :param number_of_steps: steps to move
        :param the_style: Single, Double or Interleave
        :param inter_step_delay: time between steps
        """
        if port == 'drive':
            stepper_motor = crickit.drive_stepper_motor
        else:
            stepper_motor = crickit.stepper_motor

        if the_style == 'Double':
            the_style = stepper.DOUBLE
        elif the_style == 'Interleave':
            the_style = stepper.INTERLEAVE
        else:
            the_style = stepper.SINGLE

        if direction == stepper.FORWARD:
            for steps in range(int(number_of_steps)):
                stepper_motor.onestep(direction=stepper.FORWARD, style=the_style)
                time.sleep(inter_step_delay)
        else:
            for steps in range(int(number_of_steps)):
                stepper_motor.onestep(direction=stepper.BACKWARD, style=the_style)
                time.sleep(inter_step_delay)

    def neo_pixel_control(self, number_of_pixels, pixel_position, red, green, blue):
        """
        This is the neopixel handler

        Typical command:
        from_crickit_gui {'number_of_pixels': 8, 'command': 'set_pixel', 'green': 128,
                          'red': 121, 'pixel_position': 4, 'blue': 137}
        :param number_of_pixels: pixels on ring or strip
        :param pixel_position: pixel number to control - zero is the first
        :param red: color value
        :param green: color value
        :param blue: color value
        """
        crickit.init_neopixel(number_of_pixels)

        crickit.neopixel.fill(0)

        # Assign to a variable to get a short name and to save time.
        np = crickit.neopixel

        np[pixel_position] = (red, green, blue)

    def analog_write(self, topic, payload):
        """
        Not used for the crickit
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def digital_read(self, pin):
        """
        Not used for the crickit
        :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: from_crickit_gui {'command': 'digital_write', 'value': 0, 'pin': 0

        :param topic: message topic
        :param payload: message payload
        """
        pin = payload['pin']
        the_object = self.pins_dictionary[pin]['crickit_object']

        value = payload['value']
        self.ss.digital_write(the_object, value)

    def disable_analog_reporting(self, topic, payload):
        """
        Not used for the crickit

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

    def disable_digital_reporting(self, topic, payload):
        """
        Not used for the crickit

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

    def enable_analog_reporting(self, topic, payload):
        """
        Not used for the crickit

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

    def enable_digital_reporting(self, topic, payload):
        """
        Not used for the crickit

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

    def i2c_read(self, topic, payload):
        """
        Not used for the crickit

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

    def i2c_write(self, topic, payload):
        """
        Not used for the crickit

        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError
    def play_tone(self, topic, payload):
        """
        Not used for the crickit

        :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:
        from_crickit_gui {'pin': 0, 'command': 'pwm_write', 'value': 0.41}

        :param topic: message topic
        :param payload: message payload
        """
        pin = payload['pin'] + self.DRIVE_BASE
        the_object = self.pins_dictionary[pin]['crickit_object']

        the_value = payload['value']
        the_object.fraction = the_value

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

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

        :param topic: message topic
        :param payload: message payload
        """
        pin = payload['pin'] + self.SERVO_BASE
        the_object = self.pins_dictionary[pin]['crickit_object']

        the_angle = payload['position']
        the_object.angle = the_angle

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

        Typical message:
        from_crickit_gui {'command': 'set_mode_analog_input', 'pin': 5}

        :param topic: message topic
        :param payload: message payload
        """
        pin = payload['pin']
        if self.pins_dictionary[pin]['current_mode'] is not None:                517 ↛ 51
            self.mode_previously_set_warning(pin, self.pins_dictionary[pin]['current_mode']
            return

        self.pins_dictionary[pin]['current_mode'] = self.ANALOG_INPUT_MODE
        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
```

```
        pin = payload['pin']

        if self.pins_dictionary[pin]['current_mode'] is not None:                584 ↛ 59
            if self.pins_dictionary[pin]['current_mode'] != self.DIGITAL_OUTPUT_MODE585 ↛ 59
                self.mode_previously_set_warning(pin,
                                                 self.pins_dictionary[pin]['current_mode'])
                return

        the_object = self.pins_dictionary[pin]['crickit_object']

        self.ss.pin_mode(the_object, self.ss.OUTPUT)

    def set_mode_i2c(self, topic, payload):
        """
        Not used for the crickit

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

    def set_mode_pwm(self, topic, payload):
        """
        Set the frequency for a drive pin.

        Typical message: from_crickit_gui {'pin': 0, 'command': 'set_mode_pwm'}

        :param topic: message topic
        :param payload: message payload
        """
        pin = payload['pin'] + self.DRIVE_BASE
        if self.pins_dictionary[pin]['current_mode'] is not None:                613 ↛ 61
            if self.pins_dictionary[pin]['current_mode'] != self.PWM_OUTPUT_MODE:
                self.mode_previously_set_warning(pin,
                                                 self.pins_dictionary[pin]['current_mode'])
                return

        the_object = self.pins_dictionary[pin]['crickit_object']

        the_object.frequency = 1000

    def set_mode_servo(self, topic, payload):
        """
        Not used for crickit, but gui sends the following message:
        from_crickit_gui {'command': 'set_mode_servo', 'pin': 1}

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

    def set_mode_sonar(self, topic, payload):
        """
        Not used for crickit
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def set_mode_stepper(self, topic, payload):
        """
        Not used for crickit - 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):
        """
        Not used for crickit
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def stepper_write(self, topic, payload):
        """
        Not used for crickit - stepper objects are handled directly
        :param topic: message topic
        :param payload: message payload
        """
        raise NotImplementedError

    def dc_motor_move(self, motor, speed):
        """
        Set the specified motor to the specified speed.
        Typical message: from_crickit_gui {'command': 'digital_write', 'value': 0, 'pin': 0

        :param motor: 1 or 2
        :param speed: motor speed
        """
        motor_object = self.pins_dictionary[motor + self.MOTOR_BASE]['crickit_object']
        motor_object.throttle = speed

    def run(self):
        """
        The input polling thread
        :return:
        """
        topic = "to_crickit_gui"
        while True:
            # check the signal inputs
            for pin in range(0, 8):
                the_object = self.pins_dictionary[pin]['crickit_object']
                if self.pins_dictionary[pin]['enabled']:
                    if self.pins_dictionary[pin][
                        'current_mode'] == self.DIGITAL_INPUT_MODE or \
                            self.pins_dictionary[pin]['current_mode'] \
                            == self.DIGITAL_INPUT_PULLUP_MODE:
                        the_input = self.ss.digital_read(the_object)

                        if the_input != self.pins_dictionary[pin]['last_value']:
                            self.pins_dictionary[pin]['last_value'] = the_input
                            timestamp = self.get_time_stamp()
                            payload = {'report': 'digital_input', 'pin': pin,
                                       'value':
                                           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")
```

```
    parser.add_argument("-s", dest="subscriber_port", default='43125',
                        help="Subscriber IP port")
    parser.add_argument("-t", dest="loop_time", default=".1",
                        help="Event Loop Timer in seconds")

    args = parser.parse_args()
    if args.back_plane_ip_address == 'None':                                          765 ↛ 76
        args.back_plane_ip_address = None
    if args.board_type == 'None':                                                     767 ↛ 76
        args.back_plane_ip_address = None
    kw_options = {
        '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),
        'board_type': args.board_type}

    try:
        app = CrickitGateway(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__':                                                            793 ↛ ex
    # replace with name of function you defined above
    crickit_gateway()
```

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