

Coverage for /home/pi/Downloads/bots-in-pieces-examples-master/banyan-bot-blue/banyan_assets/bluetooth_gateway.py : 76%

153 statements 121 run 32 missing 0 excluded 12 partial



```

1  #!/usr/bin/env python3
2
3  """
4  blue_tooth_gateway.py
5
6  Copyright (c) 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 json
24 import subprocess
25 import signal
26 import sys
27 import subprocess
28 import threading
29
30 from bluetooth import *
31
32 from boltons.socketutils import BufferedSocket
33
34 from python_banyan.banyan_base import BanyanBase
35
36
37 # noinspection PyMethodMayBeStatic,PyBroadException
38 class BlueToothGateway(BanyanBase, threading.Thread):
39     """
40     This class implements Bluetooth an RFCOMM server or client,
41     configurable from command line options.
42
43     usage: bluetooth_gateway.py [-h] [-a SERVER_BT_ADDRESS]
44                               [-b BACK_PLANE_IP_ADDRESS] [-g GATEWAY_TYPE]
45                               [-j JSON_DATA] [-l PUBLISH_TOPIC]
46                               [-m SUBSCRIBER_LIST [SUBSCRIBER_LIST ...]]
47                               [-n PROCESS_NAME] [-p PUBLISHER_PORT]
48                               [-s SUBSCRIBER_PORT] [-t LOOP_TIME] [-u UUID]
49
50     optional arguments:
51     -h, --help            show this help message and exit
52     -a SERVER_BT_ADDRESS  Bluetooth MAC Address of Bluetooth Gateway
53     -b BACK_PLANE_IP_ADDRESS

```

```

54                                     None or IP address used by Back Plane
55     -g GATEWAY_TYPE                 Type of Gateway : server or client
56     -j JSON_DATA                   Bluetooth packets json encoded True or False
57     -l PUBLISH_TOPIC               Banyan publisher topic
58     -m SUBSCRIBER_LIST [SUBSCRIBER_LIST ...]
59                                     Banyan topics space delimited: topic1 topic2 topic3
60     -n PROCESS_NAME                Set process name in banner
61     -p PUBLISHER_PORT              Publisher IP port
62     -s SUBSCRIBER_PORT             Subscriber IP port
63     -t LOOP_TIME                   Event Loop Timer in seconds
64     -u UUID                         Bluetooth UUID

```

```

65
66     """

```

```

67
68     # gateway types

```

```

69     BTG_SERVER = 0

```

```

70     BTG_CLIENT = 1

```

```

71
72     def __init__(self, back_plane_ip_address=None, subscriber_port='43125',
73                  publisher_port='43124', process_name=None, loop_time=.001,
74                  gateway_type=BTG_SERVER, publish_topic=None,
75                  uuid='e35d6386-1802-414f-b2b9-375c92fa23e0',
76                  server_bt_address=None, subscriber_list=None,
77                  json_data=False):

```

```

78         """

```

```

79         This method initialize the class for operation

```

```

80
81         """

```

```

82         # save input parameters as instance variables

```

```

83         self.back_plane_ip_address = back_plane_ip_address

```

```

84         self.subscriber_port = subscriber_port

```

```

85         self.publisher_port = publisher_port

```

```

86         self.loop_time = loop_time

```

```

87         self.gateway_type = gateway_type

```

```

88
89         # set the name for the banner depending upon client or server

```

```

90         if process_name is None:

```

90 →/96

```

91             if self.gateway_type == self.BTG_CLIENT:

```

91 →/92

```

92                 self.process_name = 'BanyanBluetoothClient'

```

```

93             else:

```

```

94                 self.process_name = 'BanyanBluetoothServer'

```

```

95         else:

```

```

96             self.process_name = process_name

```

```

97
98         self.publish_topic = publish_topic

```

```

99
100        self.uuid = uuid

```

```

101        self.server_bt_address = server_bt_address

```

```

102        self.json_data = json_data

```

```

103
104        # initialize the parent

```

```

105
106        super(BlueToothGateway, self).__init__(
107            back_plane_ip_address=self.back_plane_ip_address,
108            subscriber_port=self.subscriber_port,
109            publisher_port=self.publisher_port,
110            process_name=self.process_name,
111            loop_time=self.loop_time)

```

```

112
113        self.subscriber_list = subscriber_list

```

```

115     for topic in self.subscriber_list:
116         self.set_subscriber_topic(topic)
117         print('Subscribed to: ', topic)
118
119     print('Publish to   : ', self.publish_topic)
120
121     mac = self.find_local_mac_address()
122     if mac:                                     122 →/125
123         print('Local Bluetooth MAC Address: ', mac)
124     else:
125         print('No Bluetooth Interface Found - Exiting')
126         sys.exit(0)
127
128     if self.gateway_type == self.BTG_SERVER:    128 →/150
129         self.server_sock = BluetoothSocket(RFCOMM)
130         self.server_sock.bind(("", PORT_ANY))
131         self.server_sock.listen(1)
132
133         port = self.server_sock.getsockname()[1]
134
135         advertise_service(self.server_sock, "BanyanBlueToothServer",
136                             service_id=uuid,
137                             service_classes=[uuid, SERIAL_PORT_CLASS],
138                             profiles=[SERIAL_PORT_PROFILE],
139                             )
140
141         print("Waiting for connection on RFCOMM channel %d" % port)
142         try:
143             self.client_sock, self.client_info = self.server_sock.accept()
144         except KeyboardInterrupt:
145             self.clean_up()
146             sys.exit(0)
147
148         print("Accepted connection from ", self.client_info)
149     else:
150         service_matches = find_service(uuid=self.uuid,
151                                         address=self.server_bt_address)
152
153         if len(service_matches) == 0:
154             print("Could not find the remote Bluetooth server - exiting")
155             self.clean_up()
156             sys.exit(0)
157
158         first_match = service_matches[0]
159         port = first_match["port"]
160         name = first_match["name"]
161         host = first_match["host"]
162
163         print("connecting to \"%s\" on %s" % (name, host))
164
165         # Create the client socket
166         self.client_sock = BluetoothSocket(RFCOMM)
167         self.client_sock.connect((host, port))
168
169         # wrap the socket for both client and server
170         self.bsock = BufferedSocket(self.client_sock)
171
172         # create a thread to handle receipt of bluetooth data
173         threading.Thread.__init__(self)
174         self.daemon = True
175

```

```

176     # start the thread
177     self.start()
178
179     # this will keep the program running forever
180     try:
181         self.receive_loop()
182     except KeyboardInterrupt:
183         self.clean_up()
184         sys.exit(0)
185
186     def incoming_message_processing(self, topic, payload):
187         """
188         Process the incoming Banyan message to
189         be sent to the Bluetooth network
190         :param topic: topic string
191         :param payload: payload data
192         """
193
194         # if the bluetooth device requires json encoding
195         if self.json_data:
196             data_out = json.dumps(payload)
197             data_out = data_out.encode('utf-8')
198
199             try:
200                 self.bsock.send(data_out)
201             except Exception as e:
202                 self.clean_up()
203                 raise RuntimeError('Write Error')
204         else:
205             # convert the payload to a string
206             data_out = str(payload['report'])
207             data_out = data_out.encode('utf-8')
208             self.client_sock.send(data_out)
209
210     def find_local_mac_address(self):
211         """
212         Get the local bluetooth mac address
213         :return: mac address string or None
214         """
215         proc = subprocess.Popen(['hcitool', 'dev'],
216                                 stdin=subprocess.PIPE, stdout=subprocess.PIPE)
217
218         data = proc.communicate()
219
220         data = data[0].decode()
221
222         data = data.split('\t')
223         if len(data) < 2:
224             return None
225         else:
226             return data[2].strip()
227
228     def run(self):
229         """
230         This is thread that receives packets from the bluetooth interface
231         :return:
232         """
233
234         while True:
235             # if json encoding look for termination character
236             # used for a dictionary

```

223 →/224

```

237         if self.json_data:
238             try:
239                 data = self.bsock.recv_until(b'}',
240                                             timeout=0,
241                                             with_delimiter=True)
242
243             except KeyboardInterrupt:
244                 self.clean_up()
245                 sys.exit(0)
246             except Exception as e:
247                 continue
248
249             data = data.decode()
250             data = json.loads(data)
251
252             self.publish_payload(data, self.publish_topic)
253
254             # data is not json encoded
255             else:
256                 try:
257                     data = (self.client_sock.recv(1)).decode()
258                 except KeyboardInterrupt:
259                     self.clean_up()
260                     sys.exit(0)
261                 payload = {'command': data}
262                 self.publish_payload(payload, self.publish_topic)
263
264     def bluetooth_gateway():
265         parser = argparse.ArgumentParser()
266         parser.add_argument("-a", dest="server_bt_address", default="None",
267                             help="Bluetooth MAC Address of Bluetooth Gateway"),
268         parser.add_argument("-b", dest="back_plane_ip_address", default="None",
269                             help="None or IP address used by Back Plane")
270         parser.add_argument("-g", dest="gateway_type", default="server",
271                             help="Type of Gateway : server or client"),
272         parser.add_argument("-j", dest="json_data", default="False",
273                             help="Bluetooth packets json encoded true or false"),
274         parser.add_argument("-l", dest="publish_topic", default="from_bt_gateway",
275                             help="Banyan publisher topic"),
276         parser.add_argument("-m", dest="subscriber_list",
277                             default=["None"], nargs="+",
278                             help="Banyan topics space delimited: topic1 topic2 "
279                                    "topic3")
280         parser.add_argument("-n", dest="process_name", default="None",
281                             help="Set process name in banner")
282         parser.add_argument("-p", dest="publisher_port", default='43124',
283                             help="Publisher IP port")
284         parser.add_argument("-s", dest="subscriber_port", default='43125',
285                             help="Subscriber IP port")
286         parser.add_argument("-t", dest="loop_time", default=".01",
287                             help="Event Loop Timer in seconds")
288         parser.add_argument("-u", dest="uuid",
289                             default="e35d6386-1802-414f-b2b9-375c92fa23e0",
290                             help="Bluetooth UUID")
291
292         args = parser.parse_args()
293
294         if args.back_plane_ip_address == 'None':
295             args.back_plane_ip_address = None
296
297         if args.server_bt_address == 'None':
298             args.backplane_ip_address = None

```

```

298 | if args.gateway_type == 'server':                                298 →/301
299 |     args.gateway_type = BluetoothGateway.BTG_SERVER
300 | else:
301 |     args.gateway_type = BluetoothGateway.BTG_CLIENT
302 | if args.server_bt_address == 'None':                            302 →/304
303 |     args.server_bt_address = None
304 | if args.process_name == 'None':                                  304 →/306
305 |     args.process_name = None
306 | if args.subscriber_list == ['None']:
307 |     args.subscriber_list = ['to_bt_gateway']
308 | if args.json_data == 'False' or args.json_data == 'false':
309 |     args.json_data = False
310 | else:
311 |     args.json_data = True
312 |
313 | kw_options = {
314 |     'back_plane_ip_address': args.back_plane_ip_address,
315 |     'publisher_port': args.publisher_port,
316 |     'subscriber_port': args.subscriber_port,
317 |     'process_name': args.process_name,
318 |     'json_data': args.json_data,
319 |     'loop_time': float(args.loop_time),
320 |     'publish_topic': args.publish_topic,
321 |     'gateway_type': args.gateway_type,
322 |     'uuid': args.uuid,
323 |     'server_bt_address': args.server_bt_address,
324 |     'subscriber_list': args.subscriber_list
325 | }
326 |
327 | BluetoothGateway(**kw_options)
328 |
329 |
330 | # signal handler function called when Control-C occurs
331 | # noinspection PyShadowingNames,PyUnusedLocal,PyUnusedLocal
332 | def signal_handler(sig, frame):
333 |     print('Exiting Through Signal Handler')
334 |     raise KeyboardInterrupt
335 |
336 |
337 | # listen for SIGINT
338 | signal.signal(signal.SIGINT, signal_handler)
339 | signal.signal(signal.SIGTERM, signal_handler)
340 |
341 |
342 | if __name__ == '__main__':                                        342 →/exit
343 |     bluetooth_gateway()

```