

Coverage for /home/pi/banyan-bot-blue/banyan_assets/bluetooth_gateway.py : 69%

141 statements 103 run 38 missing 0 excluded 15 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
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:
91             if self.gateway_type == self.BTG_CLIENT:
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
114

```

90 →/96

91 →/92

```
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 →/147
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 |
143 |         self.client_sock, self.client_info = self.server_sock.accept()
144 |
145 |         print("Accepted connection from ", self.client_info)
146 |     else:
147 |         service_matches = find_service(uuid=self.uuid,
148 |                                         address=self.server_bt_address)
149 |
150 |         if len(service_matches) == 0:
151 |             print("Could not find the remote Bluetooth server - exiting")
152 |             sys.exit(0)
153 |
154 |         first_match = service_matches[0]
155 |         port = first_match["port"]
156 |         name = first_match["name"]
157 |         host = first_match["host"]
158 |
159 |         print("connecting to \"%s\" on %s" % (name, host))
160 |
161 |         # Create the client socket
162 |         self.client_sock = BluetoothSocket(RFCOMM)
163 |         self.client_sock.connect((host, port))
164 |
165 |         # wrap the socket for both client and server
166 |         self.bsock = BufferedSocket(self.client_sock)
167 |
168 |         # create a thread to handle receipt of bluetooth data
169 |         threading.Thread.__init__(self)
170 |         self.daemon = True
171 |
172 |         # start the thread
173 |         self.start()
174 |
175 |         # this will keep the program running forever
```

```

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

```

187 →/188

215 →/216

229 →/230

```

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

```

```
298 |     kw_options = {
299 |         'back_plane_ip_address': args.back_plane_ip_address,
300 |         'publisher_port': args.publisher_port,
301 |         'subscriber_port': args.subscriber_port,
302 |         'process_name': args.process_name,
303 |         'json_data': args.json_data,
304 |         'loop_time': float(args.loop_time),
305 |         'publish_topic': args.publish_topic,
306 |         'gateway_type': args.gateway_type,
307 |         'uuid': args.uuid,
308 |         'server_bt_address': args.server_bt_address,
309 |         'subscriber_list': args.subscriber_list
310 |     }
311 |
312 |     try:
313 |         app = BlueToothGateway(**kw_options)
314 |     except KeyboardInterrupt:
315 |         sys.exit()
316 |
317 |     # noinspection PyUnusedLocal
318 |     def signal_handler(sig, frame):
319 |         print("Control-C detected. See you soon.")
320 |         app.clean_up()
321 |         sys.exit(0)
322 |
323 |     # listen for SIGINT
324 |     signal.signal(signal.SIGINT, signal_handler)
325 |     signal.signal(signal.SIGTERM, signal_handler)
326 |
327 |
328 | if __name__ == '__main__':
329 |     bluetooth_gateway()
```

328 →/exit