GPSD socket connection and decoding JSON into Python dictionaries

<https://codereview.stackexchange.com/questions/120951/gpsd-socket-connection-and-decoding-json-into-python-dictionaries>

[GPS3](https://github.com/wadda/gps3) is a python 2.7-3.5 interface to [GPSD](http://www.catb.org/gpsd).

I've stripped back everything to two classes.

#!/usr/bin/env python3

# coding=utf-8

"""

GPS3 (gps3.py) is a Python 2.7-3.5 GPSD interface (http://www.catb.org/gpsd)

Defaults host='127.0.0.1', port=2947, gpsd\_protocol='json'

GPS3 has two classes.

1) 'GPSDSocket' to create a socket connection and retreive the output from GPSD.

2) 'Fix' unpacks the streamed gpsd data into python dictionaries.

These dictionaries are populated from the JSON data packet sent from the GPSD.

Import import gps3

Instantiate gps\_connection = gps3.GPSDSocket()

gps\_fix = gps3.Fix()

Use print('Altitude = 'gps\_fix.TPV['alt'])

print('Latitude = 'gps\_fix.TPV['lat'])

Consult Lines 150-ff for Attribute/Key possibilities.

or http://www.catb.org/gpsd/gpsd\_json.html

Run human.py; python[X] human.py [arguments] for a human experience.

"""

from \_\_future\_\_ import print\_function

import json

import select

import socket

import sys

\_\_author\_\_ = 'Moe'

\_\_copyright\_\_ = "Copyright 2015-2016 Moe"

\_\_license\_\_ = "MIT"

\_\_version\_\_ = "0.11a"

HOST = "127.0.0.1" # gpsd defaults

GPSD\_PORT = 2947 # "

PROTOCOL = 'json' # "

class GPSDSocket(object):

"""Establish a socket with gpsd, by which to send commands and receive data.

"""

def \_\_init\_\_(self, host=HOST, port=GPSD\_PORT, gpsd\_protocol=PROTOCOL, devicepath=None):

self.devicepath\_alternate = devicepath

self.response = None

self.protocol = gpsd\_protocol

self.streamSock = None

if host:

self.connect(host, port)

def connect(self, host, port):

"""Connect to a host on a given port.

:param port:

:param host:

"""

for alotta\_stuff in socket.getaddrinfo(host, port, 0, socket.SOCK\_STREAM):

family, socktype, proto, \_canonname, host\_port = alotta\_stuff

try:

self.streamSock = socket.socket(family, socktype, proto)

self.streamSock.connect(host\_port)

self.streamSock.setblocking(False)

except OSError as error:

sys.stderr.write('\nGPSDSocket.connect OSError is-->', error)

sys.stderr.write('\nAttempt to connect to a gpsd at {0} on port \'{1}\' failed:\n'.format(host, port))

sys.stderr.write('Please, check your number and dial again.\n')

self.close()

sys.exit(1) # TODO: gpsd existence check and start

finally:

self.watch(gpsd\_protocol=self.protocol)

def watch(self, enable=True, gpsd\_protocol='json', devicepath=None):

"""watch gpsd in various gpsd\_protocols or devices.

Arguments:

self:

enable: (bool) stream data to socket

gpsd\_protocol: (str) 'json', 'nmea', 'rare', 'raw', 'scaled', 'split24', or 'pps'

devicepath: option for non-default device path

Returns:

command: (str) e.g., '?WATCH={{"enable":true,"json":true}}'

"""

# TODO: 'timing' requires special attention, as it is undocumented and lives with dragons

command = '?WATCH={{"enable":true,"{0}":true}}'.format(gpsd\_protocol)

if gpsd\_protocol == 'rare': # 1 for a channel, gpsd reports the unprocessed NMEA or AIVDM data stream

command = command.replace('"rare":true', '"raw":1')

if gpsd\_protocol == 'raw': # 2 channel that processes binary data, received data verbatim without hex-dumping.

command = command.replace('"raw":true', '"raw",2')

if not enable:

command = command.replace('true', 'false') # sets -all- command values false .

if devicepath:

command = command.replace('}', ',"device":"') + devicepath + '"}'

return self.send(command)

def send(self, commands):

"""Ship commands to the daemon

:param commands:

"""

# session.send("?POLL;") # TODO: Figure a way to work this in.

# The POLL command requests data from the last-seen fixes on all active GPS devices.

# Devices must previously have been activated by ?WATCH to be pollable.

if sys.version\_info[0] < 3: # Not less than 3, but 'broken hearted' because

self.streamSock.send(commands) # 2.7 chokes on 'bytes' and 'encoding='

else:

self.streamSock.send(bytes(commands, encoding='utf-8')) # It craps out here when there is no gpsd running

# TODO: Add recovery, check gpsd existence, re/start, etc..

def \_\_iter\_\_(self):

"""banana""" # <------- for scale

return self

def next(self, timeout=0):

"""Return empty unless new data is ready for the client. Will sit and wait for timeout seconds

:param timeout:

"""

try:

(waitin, \_waitout, \_waiterror) = select.select((self.streamSock,), (), (), timeout)

if not waitin:

return

else:

gpsd\_response = self.streamSock.makefile() # was '.makefile(buffering=4096)' In strictly Python3

self.response = gpsd\_response.readline()

return self.response

except OSError as error:

sys.stderr.write('The readline OSError in GPSDSocket.next is this: ', error)

return

\_\_next\_\_ = next # Workaround for changes in iterating between Python 2.7 and 3.5

def close(self):

"""turn off stream and close socket"""

if self.streamSock:

self.watch(enable=False)

self.streamSock.close()

self.streamSock = None

return

class Fix(object):

"""Retrieve JSON Object(s) from GPSDSocket and unpack it into respective

gpsd 'class' dictionaries, TPV, SKY, etc. yielding hours of fun and entertainment.

"""

def \_\_init\_\_(self):

"""Sets of potential data packages from a device through gpsd, as a generator of class attribute dictionaries"""

version = {"release", "proto\_major", "proto\_minor", "remote", "rev"}

tpv = {"alt", "climb", "device", "epc", "epd", "eps", "ept", "epv", "epx", "epy", "lat", "lon", "mode", "speed", "tag", "time", "track"}

sky = {"satellites", "gdop", "hdop", "pdop", "tdop", "vdop", "xdop", "ydop"}

gst = {"alt", "device", "lat", "lon", "major", "minor", "orient", "rms", "time"}

att = {"acc\_len", "acc\_x", "acc\_y", "acc\_z", "depth", "device", "dip", "gyro\_x", "gyro\_y", "heading", "mag\_len", "mag\_st", "mag\_x", "mag\_y", "mag\_z",

"pitch", "pitch\_st", "roll", "roll\_st", "temperature", "time", "yaw", "yaw\_st"} # TODO: Check Device flags

pps = {"device", "clock\_sec", "clock\_nsec", "real\_sec", "real\_nsec"}

device = {"activated", "bps", "cycle", "mincycle", "driver", "flags", "native", "parity", "path", "stopbits", "subtype"} # TODO: Check Device flags

poll = {"active", "fixes", "skyviews", "time"}

devices = {"devices", "remote"}

# ais = {} # see: http://catb.org/gpsd/AIVDM.html

error = {"message"}

# 'repository' of dictionaries possible, and possibly 'not applicable'

packages = {"VERSION": version,

"TPV": tpv,

"SKY": sky, "GST": gst, "ATT": att, "PPS": pps,

"DEVICE": device, "POLL": poll,

"DEVICES": devices,

"ERROR": error} # etc.

# TODO: Create the full suite of possible JSON objects and a better way for deal with subsets

for package\_name, datalist in packages.items():

\_emptydict = {key: 'n/a' for (key) in datalist} # There is a case for using None instead of 'n/a'

setattr(self, package\_name, \_emptydict)

self.SKY['satellites'] = [{'PRN': 'n/a', 'ss': 'n/a', 'el': 'n/a', 'az': 'n/a', 'used': 'n/a'}]

self.DEVICES['devices'] = [{"class": 'n/a', "path": 'n/a', "activated": 'n/a', "flags": 'n/a', "driver": 'n/a',

"native": 'n/a', "bps": 'n/a', "parity": 'n/a', "stopbits": 'n/a', "cycle": 'n/a'}]

def refresh(self, gpsd\_data\_package):

"""Sets new socket data as Fix attributes

Arguments:

self (class):

gpsd\_data\_package (json object):

Returns:

self attribute dictionaries, e.g., self.TPV['lat']

Raises:

AttributeError: 'str' object has no attribute 'keys' when the device falls out of the system

ValueError, KeyError: stray data, should not happen

"""

try:

fresh\_data = json.loads(gpsd\_data\_package) # The reserved word 'class' is popped from JSON object class

package\_name = fresh\_data.pop('class', 'ERROR') # gpsd data package errors are also 'ERROR'.

package = getattr(self, package\_name, package\_name) # packages are named for JSON object class

for key in package.keys(): # TODO: Rollover and retry. It fails here when device disappears

package[key] = fresh\_data.get(key, 'n/a') # Updates and restores 'n/a' if key is absent in the socket

# response, present --> "key: 'n/a'" instead.'

except AttributeError: # 'str' object has no attribute 'keys' TODO: if returning 'None' is a good idea

print("No Data")

return None

except (ValueError, KeyError) as error:

sys.stderr.write(str(error)) # Look for extra data in stream

return None

if \_\_name\_\_ == '\_\_main\_\_':

print('\n', \_\_doc\_\_)

#

# Someday a cleaner Python interface will live here

#

# End

While 'refreshing' the data from the GPSD socket read, the JSON object is loaded into a JSON decoder module.

This fresh data output has 'class' popped and it's value becomes an attribute of the instance.

Remaining data goes into a dictionary with the new values, such as gps\_fix.TPV['lat'] = -33.123456789.

If data is missing from the socket, key or value, persistently or sporadically, the key has its value replaced with 'n/a', the initialised value.

In general looks good and well documented.

* In the close method the return statement is unnecessary.
* Quotes are inconsistently used.
* Most of the :param annotations in the docstrings are unused. If you're not going to document them, just leave them out. The watch method is also not using the syntax at all, where it would make a lot of sense to use it.
* The finally block in connect seems weird. If I'm not mistaken it *will* be executed even if sys.exit is called (since that's implemented using a SystemExit exception) - is that intentional? I'd put a comment on it if so.
* Also, is the watch method intended to be called from outside the class? If not, then the default arguments are moot. Possibly also prefix it to avoid calling it from outside the class.
* In next the else block can be put inline as the if already returns from the method. Again, the return in the except handler is not necessary.
* Also, return None is the same as return, but I imagine that's done for clarity.
* In the \_emptydict creation, the parens around key aren't needed:

\_emptydict = {key: 'n/a' for key in datalist}

If possible I'd use the same construction for SKY and DEVICES btw.

* The documentation for refresh is wrong, there's nothing returned from that method (well None, but that doesn't count).