```
#
# File: run.py
# Description: Main class. Lasts the runtime of the program. Instantiates
#               python-obd library, backend for the PyQt application. It all
#                 starts here!
# Project: Carberry Pi
# Author: Ryan McHugh
# Year: 2020
#
import sys
import subprocess
import atexit
from PyQt5 import QtWidgets
from PyQt5.QtQml import QQmlApplicationEngine
from PyQt5.QtCore import QObject, QThread
from PyQt5.QtGui import QTextObject
from PyQt5 import QtCore
import random
from functools import partial
import re
from datetime import datetime
import obd
import json
from context import Main_Context



# Main_Context moved to context.py

mc = Main_Context()

class Obd_Thread(QThread):

    def __init__(self, mc):
        QThread.__init__(self, mc)
        self.mc = mc

    def __del__(self):
        self.wait()

    def set_rpm(self, r):
        if(r.value):
            # mc.rpmValue = re.findall(’\d{3}’, r.value)
            mc.handler = {’rpm’: r.value.magnitude}
            # print(r.value.magnitude)

    def set_speed(self, r):
        loc = mc.getConfig()[’locality’][’current’]
        if(r.value):
            val = r.value.to("mph").magnitude if loc == ’en-US’ else r.value.magnitude
            mc.handler = {’speed’: val}
            # mc.speedValue = r.value.to("mph").magnitude
            # print(r.value.magnitude)

    def set_temp(self, r):
        loc = mc.getConfig()[’locality’][’current’]
        if(r.value):
            # mc.tempValue = r.value.to(’degF’).magnitude
            mc.handler = {’engine_temp’: r.value.to(’degF’).magnitude if loc == ’en-US’ else r.value.magnitude}

    def set_code(self, r):
        if(r.value and r.value != ’[]’):
            code_list = []
            for entry in r.value:
                code_list.append(entry[0] + ’|’ + entry[1])
            mc.handler = {’code-exists’: True, ’code’: code_list}
    # def set_val(self, r, r.name):
    #     if(r.value)
    #         mc.diagnostics = {r.name, r.value}




    def run(self):

        # OBD lib setup
        # port = obd.scan_serial()

        connection = obd.Async(fast=True, timeout=0.5)

        connection.watch(obd.commands.RPM, callback=self.set_rpm)
        connection.watch(obd.commands.SPEED, callback=self.set_speed)
        connection.watch(obd.commands.COOLANT_TEMP, callback=self.set_temp)
        connection.watch(obd.commands.GET_DTC, callback=self.set_code)

        command_list = connection.supported_commands

        ignore_list = []

        connection_sync = obd.OBD()

        # ret = connection.query(obd.commands.GET_DTC)
        # print("Command: " + ret.command + ", Value: " + ret.value)
        # print(ret.value)

        # mc.diagnostics = {’connection-established’: connection_sync.status()}
        mc.diagnostics = {’connection-established’: connection_sync.status()}

        for command in command_list:
                if(command is not None and command not in ignore_list):
                    # connection.query(obd.commands.RPM)
                    ret = connection_sync.query(command)
                    if not ret.is_null():
                        # add_diagnostic()
                        # print(ret.__str__())
                        print("Command: " + command.name + ", Value: " + str(ret.__str__()));
                        mc.diagnostics = {command.name : ret.__str__()}



                # ret = connection.query(command)
                # print(ret)
            # mc.handler = {command.name: ret.value.magnitude}

        # for i in range(len(command_list)):
        #     print(list(command_list)[i])

        connection.start()
        print(connection.status())






def click_handle():
    print("button clicked\n")


def getFromFile():
    f = open("output.txt", "r")
    if f.mode == ’r’:
        contents = f.readlines()

    int_arr = []

    for str in contents:
        num = re.findall('\d{3}', str)
        if num:
            int_arr.append(int(num[0]))

    return int_arr

def readConfig():
    with open('./config.json') as config_file:
        config_data = json.load(config_file)
    print(config_data)
    for key in config_data:
        mc.config = {key: config_data[key]}

def writeConfig():
    with open('./config.json', 'w') as config_file:
        json.dump(mc.getConfig(), config_file, indent=2)
    print("\n\n_____Writing config to file..._____\n\n")
    print(json.dumps(mc.getConfig(), indent=2))

def writeLog():
    with open('./log/%s' % datetime.date(datetime.now()), 'a+') as log_file:
        log_file.write("\nTime: %s\n" % datetime.time(datetime.now()))
        # log_file.write("\n___Config___\n")
        # json.dump(mc.getConfig(), log_file, indent=2)
        log_file.write("\n___Diagnostics___\n")
        json.dump(mc.getDiagnostics(), log_file, indent=2)


    # print(data)

def onRestart():
    print("exit_status: restart")
    mc.close()

def onExit():
    print("exit_status: exit")
    mc.close()

def main():

    app = QtWidgets.QApplication(sys.argv)
    ex = QQmlApplicationEngine()

    # Begin OBD Reading
    m_obdThread = Obd_Thread(mc)
    m_obdThread.start()

    ctx = ex.rootContext()
    # mc = Main_Context()
    ctx.setContextProperty("main", mc)

    # int_arr = getFromFile()

    # Pull config data
    readConfig()

    mc.handler = {'rpm': 0}
    mc.handler = {'speed': 0}
    mc.handler = {'engine_temp': 240}

    # DEV MODE

    if(len(sys.argv) > 1 and sys.argv[1].lower() == 'dev'):
        mc.handler = {'dev': True}

```

```python
    if mc.getHandler().get('dev'):
        mc.handler = {'rpm': 700}
        mc.handler = {'speed': 20}
        mc.diagnostics = {'temp1': 200}
        mc.diagnostics = {'temp2': "All Circuits Busy"}
        mc.diagnostics = {'temp3': 17123948}
        mc.diagnostics = {"temp mode this is a long temp this is a long temp\
                            this is a long temp this is a long temp": "Hello World - this is a test"}
        mc.diagnostics = {'temp5': "temp mode this is a long temp this is a long temp\
                                        this is a long temp this is a long temp"}
        mc.diagnostics = {'temp6': -14.2}

        code_tuple = ('P1234', 'Testcode description... Lorem Ipsum\
                                                        Lorem Ipsum Lorem IpsumLorem IpsumLorem IpsumLorem IpsumLorem Ipsum')
        mc.handler = {'code-exists': True, 'code': [code_tuple[0] + '|' + code_tuple[1]]}
        mc.diagnostics = {'connection-established': False}

    else:
        mc.diagnostics = {'connection-established': True}

    ex.load('run.qml')

    win = ex.rootObjects()[0]

    if mc.getConfig()['fullscreen']['current']:
        win.setWindowState(QtCore.Qt.WindowFullScreen)



    # Timer for time
    timer = QtCore.QTimer()
    timer.setInterval(1000)
    timer.timeout.connect(mc.updateTime)
    timer.start()

    # Timer for log
    timer_log = QtCore.QTimer()
    timer_log.setInterval(5000)
    timer_log.timeout.connect(writeLog)
    timer_log.start()

    # for val in int_arr:
    #     mc.rpmValue = val
    #     app.processEvents()
    #     time.sleep(.5)


    win.findChild(QObject, "stack").sig_exit.connect(onExit)
    win.findChild(QObject, "stack").sig_restart.connect(onRestart)


    win.show()
    atexit.register(writeConfig)
    atexit.register(writeLog)
    sys.exit(app.exec_())




if __name__ == '__main__':
    main()
```