```
    1: import sys
    2: import subprocess
    3: import atexit
    4: from PyQt5 import QtWidgets
    5: from PyQt5.QtQml import QQmlApplicationEngine
    6: from PyQt5.QtCore import QObject, QThread
    7: from PyQt5.QtGui import QTextObject
    8: from PyQt5 import QtCore
    9: import random
   10: from functools import partial
   11: import re
   12: from datetime import datetime
   13: import obd
   14: import json
   15: from context import Main_Context
   16:
   17:
   18:
   19: # Main_Context moved to context.py
   20:
   21: mc = Main_Context()
   22:
   23: class Obd_Thread(QThread):
   24:
   25:     def __init__(self, mc):
   26:         QThread.__init__(self, mc)
   27:         self.mc = mc
   28:
   29:     def __del__(self):
   30:         self.wait()
   31:
   32:     def set_rpm(self, r):
   33:         if(r.value):
   34:             # mc.rpmValue = re.findall('\d{3}', r.value)
   35:             mc.handler = {'rpm': r.value.magnitude}
   36:             # print(r.value.magnitude)
   37:
   38:     def set_speed(self, r):
   39:         loc = mc.getConfig()['locality']['current']
   40:         if(r.value):
   41:             val = r.value.to("mph").magnitude if loc == 'en-US' else r.value.magni
tude
   42:             mc.handler = {'speed': val}
   43:             # mc.speedValue = r.value.to("mph").magnitude
   44:             # print(r.value.magnitude)
   45:
   46:     def set_temp(self, r):
   47:         loc = mc.getConfig()['locality']['current']
   48:         if(r.value):
   49:             # mc.tempValue = r.value.to('degF').magnitude
   50:             mc.handler = {'engine_temp': r.value.to('degF').magnitude if loc == 'e
n-US' else r.value.magnitude}
   51:
   52:     def set_code(self, r):
   53:         if(r.value and r.value != '[]'):
   54:             code_list = []
   55:             for entry in r.value:
   56:                 code_list.append(entry[0] + '|' + entry[1])
   57:             mc.handler = {'code-exists': True, 'code': code_list}
   58:     # def set_val(self, r, r.name):
   59:     #     if(r.value)
   60:     #         mc.diagnostics = {r.name, r.value}
   61:
   62:
   63:
   64:
   65:     def run(self):
   66:
```

```
    67:         # OBD lib setup
    68:         # port = obd.scan_serial()
    69:
    70:         connection = obd.Async(fast=True, timeout=0.5)
    71:
    72:         connection.watch(obd.commands.RPM, callback=self.set_rpm)
    73:         connection.watch(obd.commands.SPEED, callback=self.set_speed)
    74:         connection.watch(obd.commands.COOLANT_TEMP, callback=self.set_temp)
    75:         connection.watch(obd.commands.GET_DTC, callback=self.set_code)
    76:
    77:         command_list = connection.supported_commands
    78:
    79:         # ignore_list = [obd.commands.GET_DTC]
    80:
    81:         connection_sync = obd.OBD()
    82:
    83:         # ret = connection.query(obd.commands.GET_DTC)
    84:         # print("Command: " + ret.command + ", Value: " + ret.value)
    85:         # print(ret.value)
    86:
    87:         # mc.diagnostics = {'connection-established': connection_sync.status()}
    88:         mc.diagnostics = {'connection-established': connection_sync.status()}
    89:
    90:         for command in command_list:
    91:                 if(command is not None and command not in ignore_list):
    92:                     # connection.query(obd.commands.RPM)
    93:                     ret = connection_sync.query(command)
    94:                     if not ret.is_null():
    95:                         # add_diagnostic()
    96:                         # print(ret.__str__())
    97:                         print("Command: " + command.name + ", Value: " + str(ret._
_str__()));
    98:                         mc.diagnostics = {command.name : ret.__str__()}
    99:
   100:
   101:
   102:                 # ret = connection.query(command)
   103:                 # print(ret)
   104:             # mc.handler = {command.name: ret.value.magnitude}
   105:
   106:         # for i in range(len(command_list)):
   107:         #     print(list(command_list)[i])
   108:
   109:         connection.start()
   110:         print(connection.status())
   111:
   112:
   113:
   114:
   115:
   116: def click_handle():
   117:     print("button clicked\n")
   118:
   119:
   120: def getFromFile():
   121:     f = open("output.txt", "r")
   122:     if f.mode == 'r':
   123:         contents = f.readlines()
   124:
   125:     int_arr = []
   126:
   127:     for str in contents:
   128:         num = re.findall('\d{3}', str)
   129:         if num:
   130:             int_arr.append(int(num[0]))
   131:
   132:     return int_arr
   133:
```

```
134: def readConfig():
135:     with open('./config.json') as config_file:
136:         config_data = json.load(config_file)
137:     print(config_data)
138:     for key in config_data:
139:         mc.config = {key: config_data[key]}
140:
141: def writeConfig():
142:     with open('./config.json', 'w') as config_file:
143:         json.dump(mc.getConfig(), config_file, indent=2)
144:     print("\n\n_____Writing config to file..._____\n\n")
145:     print(json.dumps(mc.getConfig(), indent=2))
146:
147: def writeLog():
148:     with open('./log/%s' % datetime.date(datetime.now()), 'a+') as log_file:
149:         log_file.write("\nTime: %s\n" % datetime.time(datetime.now()))
150:         # log_file.write("\n___Config___\n")
151:         # json.dump(mc.getConfig(), log_file, indent=2)
152:         log_file.write("\n___Diagnostics___\n")
153:         json.dump(mc.getDiagnostics(), log_file, indent=2)
154:
155:
156:     # print(data)
157:
158: def onRestart():
159:     print("exit_status: restart")
160:     mc.close()
161:
162: def onExit():
163:     print("exit_status: exit")
164:     mc.close()
165:
166: def main():
167:
168:     app = QtWidgets.QApplication(sys.argv)
169:     ex = QQmlApplicationEngine()
170:
171:     # Begin OBD Reading
172:     m_obdThread = Obd_Thread(mc)
173:     m_obdThread.start()
174:
175:     ctx = ex.rootContext()
176:     # mc = Main_Context()
177:     ctx.setContextProperty("main", mc)
178:
179:     # int_arr = getFromFile()
180:
181:     # Pull config data
182:     readConfig()
183:
184:     mc.handler = {'rpm': 0}
185:     mc.handler = {'speed': 0}
186:     mc.handler = {'engine_temp': 240}
187:
188:     # DEV MODE
189:
190:     if(len(sys.argv) > 1 and sys.argv[1].lower() == 'dev'):
191:         mc.handler = {'dev': True}
192:
193:     if mc.getHandler().get('dev'):
194:         mc.handler = {'rpm': 700}
195:         mc.handler = {'speed': 20}
196:         mc.diagnostics = {'temp1': 200}
197:         mc.diagnostics = {'temp2': "All Circuits Busy"}
198:         mc.diagnostics = {'temp3': 17123948}
199:         mc.diagnostics = {"temp mode this is a long temp this is a long temp\
200:                           this is a long temp this is a long temp": "Hello World
- this is a test"}
```

```
  201:         mc.diagnostics = {'temp5': "temp mode this is a long temp this is a long t
emp\
  202:                                    this is a long temp this is a long temp"}
  203:         mc.diagnostics = {'temp6': -14.2}
  204:
  205:         code_tuple = ('P1234', 'Testcode description... Lorem Ipsum\
  206:                                                 Lorem Ipsum Lorem IpsumLor
em IpsumLorem IpsumLorem IpsumLorem Ipsum')
  207:         mc.handler = {'code-exists': True, 'code': [code_tuple[0] + '|' + code_tup
le[1]]}
  208:         mc.diagnostics = {'connection-established': False}
  209:
  210:     else:
  211:         mc.diagnostics = {'connection-established': True}
  212:
  213:     ex.load('run.qml')
  214:
  215:     win = ex.rootObjects()[0]
  216:
  217:     if mc.getConfig()['fullscreen']['current']:
  218:         win.setWindowState(QtCore.Qt.WindowFullScreen)
  219:
  220:
  221:
  222:     # Timer for time
  223:     timer = QtCore.QTimer()
  224:     timer.setInterval(1000)
  225:     timer.timeout.connect(mc.updateTime)
  226:     timer.start()
  227:
  228:     # Timer for log
  229:     timer_log = QtCore.QTimer()
  230:     timer_log.setInterval(5000)
  231:     timer_log.timeout.connect(writeLog)
  232:     timer_log.start()
  233:
  234:     # for val in int_arr:
  235:     #     mc.rpmValue = val
  236:     #     app.processEvents()
  237:     #     time.sleep(.5)
  238:
  239:
  240:     win.findChild(QObject, "stack").sig_exit.connect(onExit)
  241:     win.findChild(QObject, "stack").sig_restart.connect(onRestart)
  242:
  243:
  244:     win.show()
  245:     atexit.register(writeConfig)
  246:     atexit.register(writeLog)
  247:     sys.exit(app.exec_())
  248:
  249:
  250:
  251:
  252: if __name__ == '__main__':
  253:     main()
```