Kairos Autonomi
498 W. 8360 S.
Sandy, Utah 84070
801-255-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com

J1939 with Apex and ELM-ST

Introduction

There are two chips that Kairos Autonomi can include in a Pronto4 to perform vehicle diagnostics: the APEX from Harrison R&D and the STN1100 (ELM-ST) from OBD Solutions. Each of these chips is capable of many diagnostics tasks and interfacing with a variety of communication protocols. This document only addresses communicating with each of these chips using the J1939 vehicle bus.

ELM-ST Chipset

The ELM-ST chipset is an extension of the ELM327 chipset. It is based on a PIC24 and is produced by OBD Solutions as the STN1100 chip. It is an OBD to UART interpreter. The STN1100 offers more features and better performance than the original ELM327 chip.

RESET CONFIGURATION

The command **ATZ** will reset all the configurations to their defaults, with the exception of the baud rate. The baud rate is saved in non-volatile memory.

Command	Description
ATZ	Reset configuration

BAUDRATE CONFIGURATION

The ELM-ST chip's baud rate is 9600 by default. Kairos Autonomi changes it to 115200 prior to shipment. The following is for reference.
Change baud rate:

1) Connect to the ELM-ST device at 9600
2) Type **ST SBR 115200**
3) Close the port and change the baud rate to 115200
4) Open the serial port and type **ST WBR**

Command	Description
ST SBR	Set the baud rate
ST WBR	Save the baud rate to non-volatile memory

J1939 CONFIGURATION

The **AT SP A** command should be sent to put the ELM-ST into J1939 mode. The command **AT DP** will return the current selected protocol. For J1939 it will return **SAEJ1939(CAN29/250)**.

Command	Description
AT SP A	Select the J1939 protocol
AT DP	Describe the current selected protocol

MESSAGE OUTPUT FORMAT

The ELM-ST chipset outputs messages received in various formats. In order to easily use this chip along with the APEX, the following set of commands should be sent in order to make the output from the ELM-ST *almost* the same as the APEX. The only difference is that the ELM-ST separates each byte with a space character.

Command	Description
AT CSM 0	CAN silent mode OFF
AT JHF0	J1939 Header Formatting OFF
AT H1	Headers ON

SETTING UP PGN FILTERS

The only limit to the amount of filters that the ELM-ST can use it the available memory of the chip. Each of the add filter commands dynamically allocates a block of RAM to store the filter, and can return OUT OF MEMORY error if there is not enough memory to add the filter. The **ST FAP** command adds a pass filter. The **ST FAB** command adds a block filter. The command **ST FAFC** adds a flow control filter. Each of these commands takes two arguments: *pattern* and *mask*; argument lengths can be from 0 to 5 bytes. The messages are matched MSB first, up to the filter length. Messages

shorter than the filter length will not match the filter. The commands **ST FCP**, **ST FCB**, and **ST FCFC** clear all the pass, block and flow control filters, respectively. The command **ST M** will monitor the J1939 bus using these filters. The command **ST MA** will temporarily suspend the filters and monitor the J1939 bus for all messages.

Command	**Description**
ST FAP [*pattern*], [*mask*]	Add a pass filter
ST FAB [*pattern*], [*mask*]	Add a block filter
ST FAFC [*pattern*], [*mask*]	Add a flow control filter
ST FCP	Clear all pass filters
ST FCB	Clear all block filters
ST FCFC	Clear all flow control filters
ST M	Monitor bus using current filters
ST MA	Monitor all messages on the bus

SENDING REQUESTS ON THE J1939 BUS

J1939 messages are sent by setting the CAN ID of the ELM-ST chip to the Request address and then writing the command to the bus (i.e. actual message for the bus is sent to the chip followed by the command message for the chip to send the previous message to the bus). The command **AT SH xxyyzz** will set the CAN header to xxyyzz. After setting the header, the only thing necessary to write the data on the bus is to send the data to the ELM-ST. For example, to send the message to clear all active trouble codes, issue the following commands:

AT SH EA00F9
00FED3

This sets up the ELM-ST to make a request (0xEA) to the address 0x00 (Engine #1) by the device 0xF9 (the scan tool). After making this change, all further requests will be directed to the engine. The **00FED3** is the PGN to clear the error code. The J1939 standard calls for the byte order to be reversed from what is shown here. The ELM-ST chip handles the reversal for the user so that it can be sent just as it appears in the PGN description.

APEX Chipset

The APEX chipset is produced by Harrison R&D and has been used successfully in the past to communicate with the OBDII port. Harrison R&D developed special firmware for Kairos Autonomi to be able to read and write to the J1939 bus.

RESET CONFIGURATION

Command	Description
D0	Reset to default Resets the APEX to all default values APEX will send 'OBDScan2.1\r\n' if successful

BAUDRATE CONFIGURATION

By default, the APEX chip is set up to communicate at 115200 baud. No special configuration is necessary. The C1 command is listed for reference.

Command	Description
C1	Set Baud Rate Allows the user to change the baud rate of the RS232 communication. Allowable baud settings are: 64 – 19200 31 – 38400 10 – 115200 02 – 428000 APEX will send 'VB\r\n' if successful

J1939 CONFIGURATION

Configure the APEX as follows:

1. Send CR to APEX
2. APEX replies with "OBDScan2.1" followed by CR LF
3. Send "93" followed by CR
4. APEX should respond in about 5 seconds with "P0" CR LF

5. Send “C5” followed by CR
6. APEX responds with “J”
7. Send “71” followed by CR
8. APEX responds with a dump of the J1939 messages on the CAN bus

Command	Description
93	Find diagnostic protocol. Valid response is ‘P0\r\n’
C5	Set J1939 Mode - Reconfigures the CAN Bus to 250kbit data rate, 29 bit IDs and sets all mask and filter registers to accept all messages. Valid response is ‘J’
71	Start monitoring the J1939 bus

MESSAGE OUTPUT FORMAT

There is no configuration to set up the output format of the APEX chip.

SETTING UP PGN FILTERS

The hardware on the APEX chip only supports five filters and two masks. The message acceptance filters and masks are used to determine if a message should be loaded into either of the receive buffers. Once a valid message has been received, the identifier fields of the message are compared to the filter values. If there is a match, that message will be loaded into the appropriate receive buffer. The filter masks are used to determine which bits in the identifiers are examined with the filters. A truth table is shown below that indicates how each bit in the identifier is compared to the masks and filters to determine if the message should be loaded into a receive buffer. The mask bit essentially determines which bits to apply the filter to. If any mask bit is set to a zero, then that bit will automatically be accepted regardless of the filter bit.

Mask Bit n	Filter Bit n	Message Identifier bit	Accept/Reject bit n
0	x	x	Accept
1	0	0	Accept
1	0	1	Reject

1	1	0	Reject
1	1	1	Accept

Filters are set by using the C3 command. The mask is set by using the C4 command.

Command	**Description**
C3	Set CAN Filter Send a 32 bit value to the specified filter number. There are 5 different filters that can be used. When setting 29 bit filters set bit 7 of the filter number. APEX will send 'CF\r\n' if successful
C4	Set CAN Mask Send a 32 bit value to the specified Mask number. There are 2 different mask registers that can be used. When setting 29 bit filters set bit 7 of the filter number. APEX will send 'CM\r\n' if successful

An example of commands to send to the APEX to set up filters to receive a set of PGNs for a minimum set of diagnostic messages: Engine Temperature, RPM, Speed, Battery Voltage, Fuel Level, and Current Gear. The last two commands set each mask wide open.

C38000FEEE00
C38100F00400
C38200FEF100
C38300FEF700
C38400FEFC00
C38500F00500
C48000FFFF00
C48100FFFF00

SENDING REQUESTS ON THE J1939 BUS

J1939 messages are sent by setting the CAN ID of the APEX chip to the Request PGN and then writing the command to the bus. The command **C8wwxxyyzz\r** will set the CAN header to wwxxyyzz. After setting the header, the only thing necessary to do to

write the data on the bus is to send the data to the APEX. For example, to send the message to clear all active trouble codes, issue the following commands:

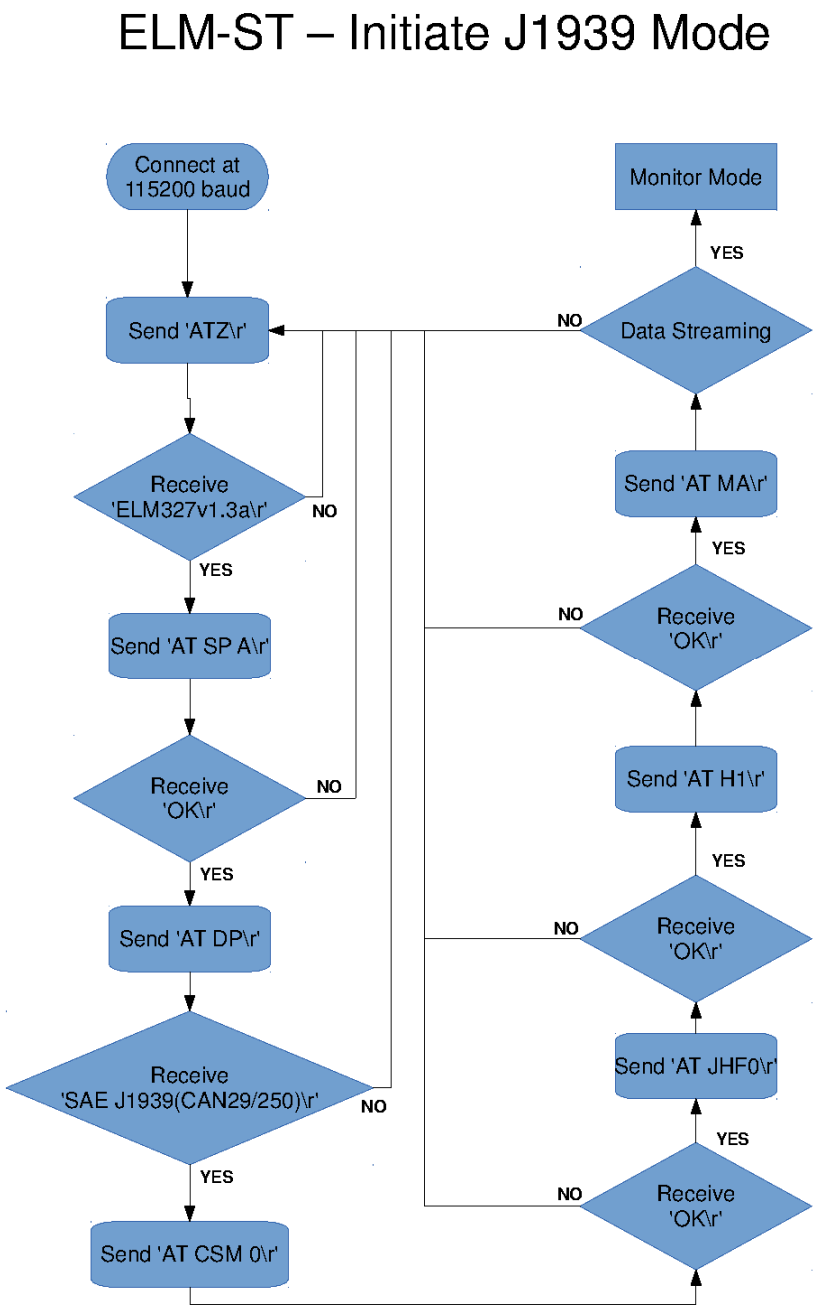
C818EA00F9
CED3FE00000000

The byte order should be reversed, as per the J1939 specification. This example sets up the APEX to make a request (0xEA) to the address 0x00 (Engine #1) by the device 0xF9 (the scan tool). The 00FED3 is the PGN to clear the error code.

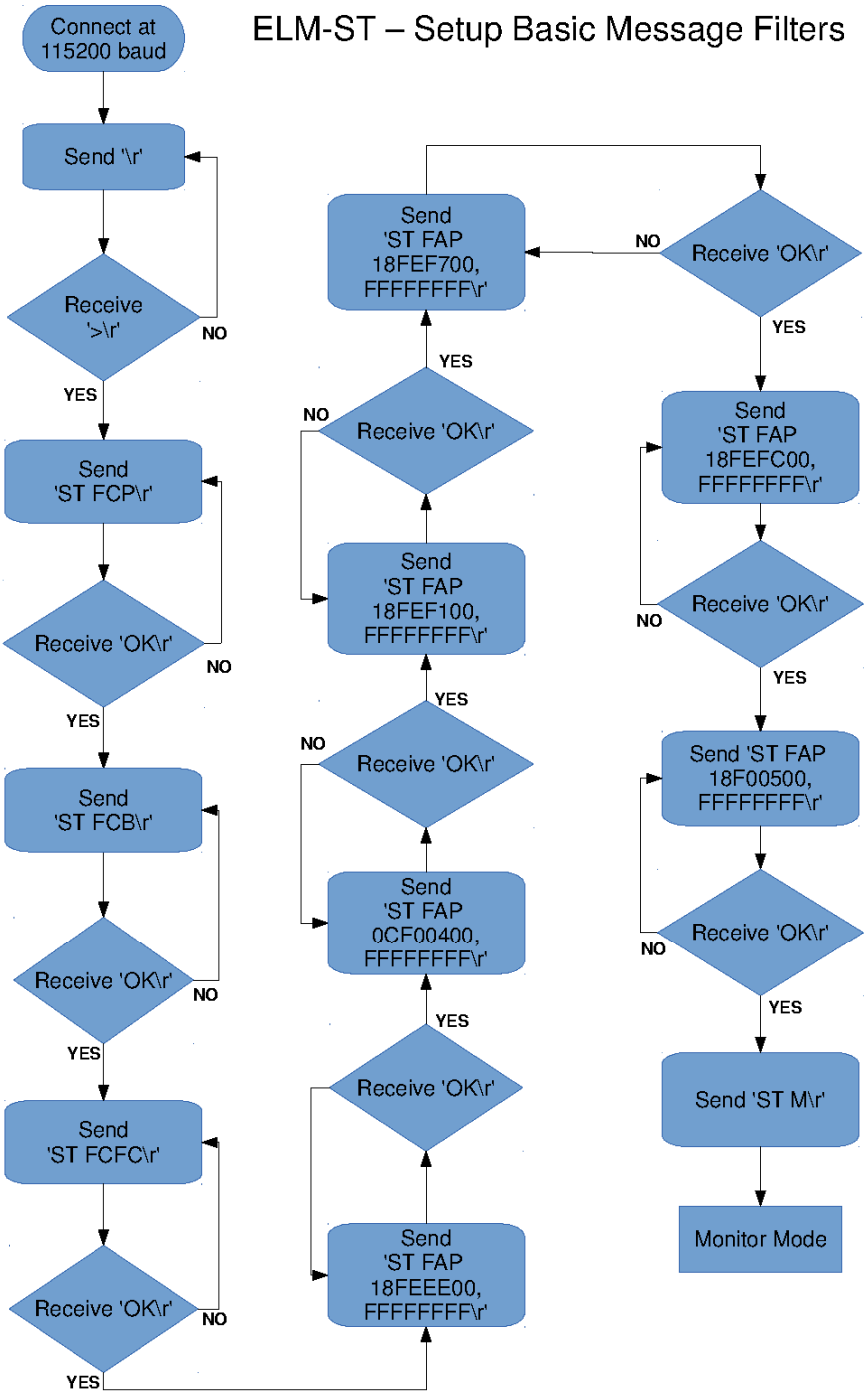
Command	**Description**
C8	Set_CAN ID Allows the 29 or 11 bit CAN ID address of the APEX to be set. This is the ID of messages transmitted on the CAN Bus APEX will send 'VI\r\n' if successful
CE	Pass through 29bit CAN ID message The eight bytes following the CE are transmitted on the CAN bus using the current CAN_ID. This command requires a fixed byte count of 9 bytes, the CE command and 8 bytes following. All CAN messages are 8 bytes and any unused bytes are zero filled.

Kairos Autonomi
498 W. 8360 S.
Sandy, Utah 84070
801-255-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com

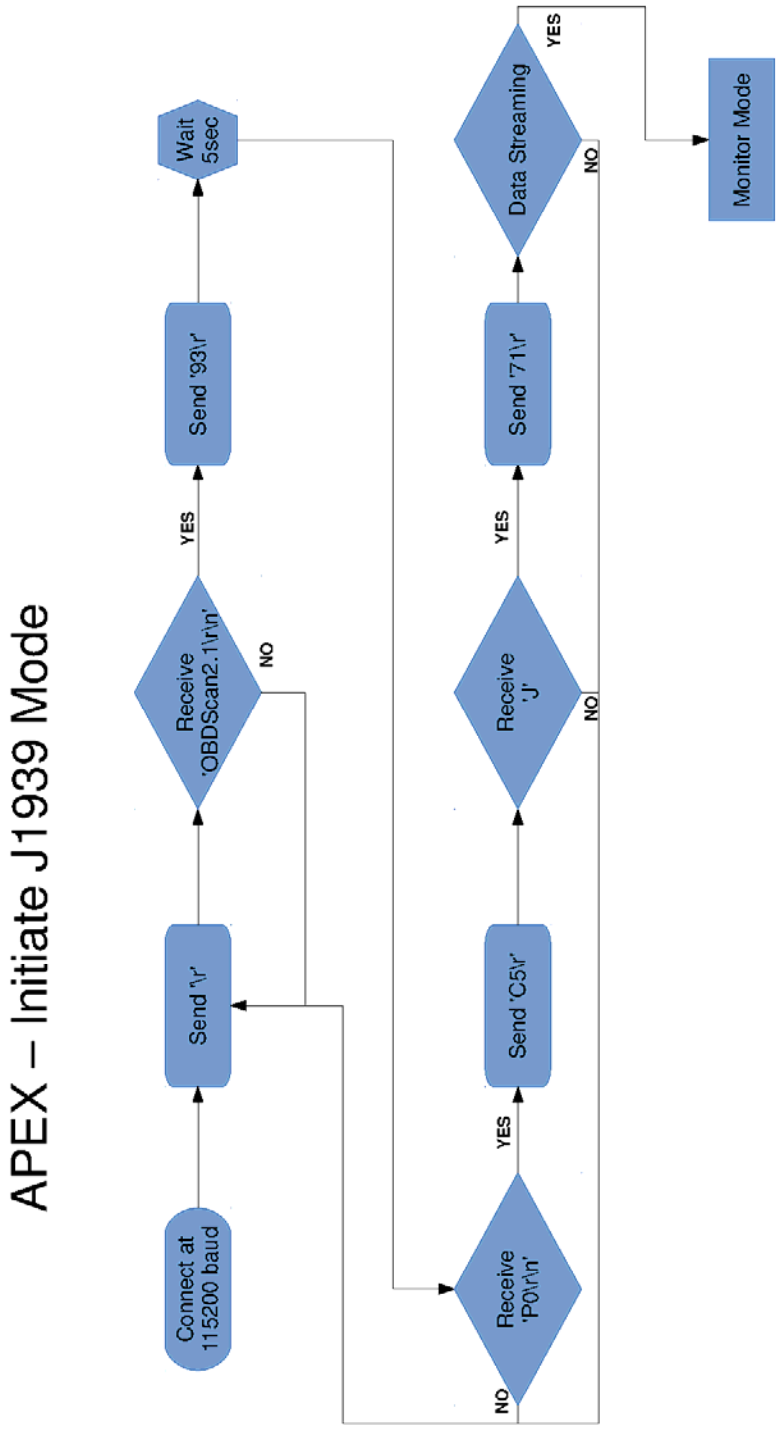
Appendix A – Command Flow



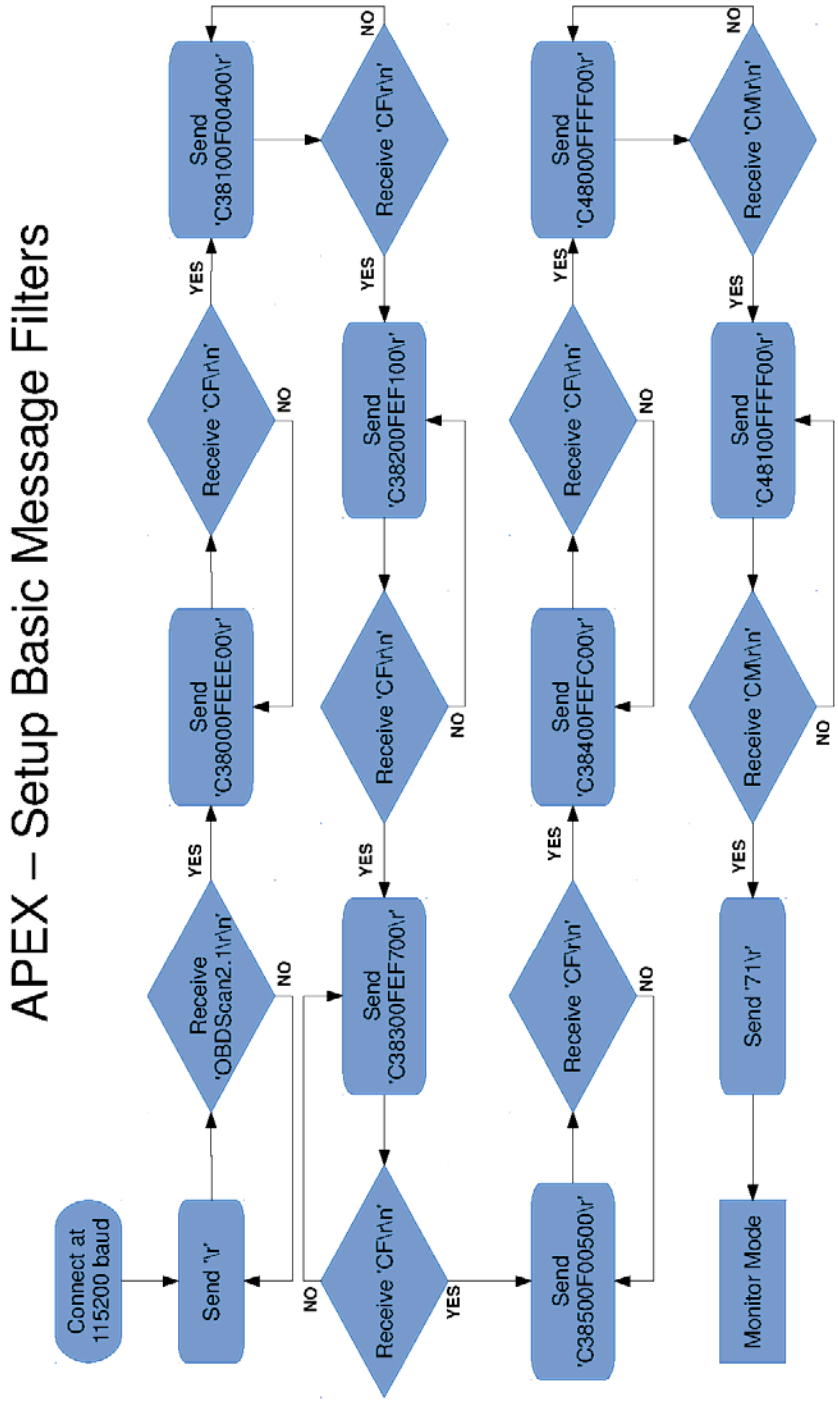
Kairos Autonomi
498 W. 8360 S.
Sandy, Utah 84070
801-255-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com



Kairos Autonomi
498 W. 8360 S.
Sandy, Utah 84070
801-255-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com



Kairos Autonomi
498 W. 8360 S.
Sandy, Utah 84070
801-255-2950 (office)
801-907-7870 (fax)
www.kairosautonomi.com



Appendix B – Sample Code

PROCESS MESSAGES

Python Example

```
    for pgn in PGNs:
        if PGNs[pgn] in msg[2:6]:
            if PGNs[pgn] == 'FEEE':
                unknownMessage = False
                engCoolantTemp = int(msg[8:10],16)-40
                engFuelTemp = int(msg[10:12],16)-40
                data['engCoolantTemp'] = engCoolantTemp
                data['engFuelTemp'] =  engFuelTemp
            elif PGNs[pgn] == 'F004':
                unknownMessage = False
                engSpeedH = int(msg[14:16], 16)
                engSpeedL = int(msg[16:18], 16) << 8
                engSpeed2 = engSpeedH + engSpeedL
                engSpeed = int(msg[14:18],16)
                data['engSpeed'] = round(engSpeed2 * 0.125,0)
            elif PGNs[pgn] == 'F003':
                unknownMessage = False
                acceleratorPedalPos = int(msg[10:12],16)
                data['acceleratorPedalPos'] = acceleratorPedalPos * .4
            elif PGNs[pgn] == 'FEF1':
                unknownMessage = False
                vehSpeed = msg[10:14]
                vehSpeed = int(reverseByGroup(vehSpeed),16)
                data['wheelSpeed'] = vehSpeed/256.0
            elif PGNs[pgn] == 'FEF5':
                unknownMessage = False
                barometricPressure = int(msg[8:10],16)
                cabInteriorTemperature = int(msg[10:14],16)
                ambientAirTemperature = int(msg[14:16],16)
                engineAirInletTemperature = int(msg[18:20],16)
                roadSurfaceTemperature = int(msg[20:24],16)
                data['engAirInletTemp'] = engineAirInletTemperature
                data['ambientAirTemp'] = ambientAirTemperature
            elif PGNs[pgn] == 'FEF6':
                unknownMessage = False
                engineParticulateTrapInletPressure = int(msg[8:10],16)
```

```
        engineIntManiPress = int(msg[10:12],16)
        engineIntakeManifoldTemperature = int(msg[12:14],16)
        engineAirInletPressure = int(msg[14:16],16)
        engineAirFilterDifferentialPressure = int(msg[16:18],16)
        tmpH = int(msg[18:20], 16)
        tmpL = int(msg[20:22], 16) << 8
        engineExhaustGasTemperature = tmpH + tmpL
        engineCoolantFilterDiffPressure = int(msg[22:24],16)
        data['engInManiPress'] = round(engineIntManiPress*.29,1)
    elif PGNs[pgn] == 'FE6C':
        unknownMessage = False
        tmpH = int(msg[20:22], 16)
        tmpL = int(msg[22:24], 16) << 8
        tachVehicleSpeed = tmpH + tmpL
        data['tachVehicleSpeed'] = tachVehicleSpeed/256.0
    elif PGNs[pgn] == 'FEE9':
        unknownMessage = False
        tmp1 = int(msg[16:18],16)
        tmp2 = int(msg[18:20],16) << 8
        tmp3 = int(msg[20:22],16) << 16
        tmp4 = int(msg[22:24],16) << 24
        data['engTotalFuelUsed'] = (tmp1 + tmp2 + tmp3 + tmp4)
    elif PGNs[pgn] == 'FEF7':
        unknownMessage = False
        batteryVoltage = msg[16:20]
        batteryVoltage = int(reverseByGroup(batteryVoltage),16) * 0.05
        data['batteryVoltage'] = batteryVoltage
    elif PGNs[pgn] == 'FEFC':
        unknownMessage = False
        fuelLevel = msg[10:12]
        data['fuelLevel'] = int(fuelLevel, 16) * 0.4
    else:
        unknownMessage = True
```

ELM-ST

J1939 CONFIGURATION

Python Example

```
    ser.write("ATZ\r")
    sleep(2)
```

```
ser.write("AT SP A\r")
sleep(.1)
ser.write("AT DP\r")
sleep(.1)
ser.write("AT CSM 0\r")
sleep(.1)
ser.write("AT JHF0\r")
sleep(.1)
ser.write("AT H1\r")
sleep(.1)
ser.readline(eol='\r')
ser.write("AT MA\r")
```

SETUP FILTERS

Python Example

```
ser.write("\r")
sleep(1)
ser.write("STFCP\r")                          #Clear pass filters
sleep(delay)
ser.write("STFCB\r")                          #Clear block filters
sleep(delay)
ser.write("STFCFC\r")                         #Clear flow control filter
sleep(delay)
ser.write("ST FAP 18FEEE00,FFFFFFFF\r")       #Engine Temp
sleep(delay)
ser.write("ST FAP 0CF00400,FFFFFFFF\r")       #Engine Speed
sleep(delay)
ser.write("ST FAP 18FEF100,FFFFFFFF\r")       #Road Speed
sleep(delay)
ser.write("ST FAP 18FEF700,FFFFFFFF\r")       #Battery Voltage
sleep(delay)
ser.write("ST FAP 18FEFC00,FFFFFFFF\r")       #Fuel Level
sleep(delay)
ser.write("ST FAP 18F00500,FFFFFFFF\r")       #Gear (not commonly used)
sleep(delay)
ser.write("STM\r")                            #Monitor bus w/ filters
```

CLEAR DTC
Python Example

```
dest = "AT SH EA00F9\r"
clr = "00FED3\r"
ser.write("\r")
sleep(0.5)
ser.write(dest)
sleep(0.5)
ser.write(clr)
```

APEX

J1939 CONFIGURATION

Python Example

```
ser.write("\r")
result = ser.readline()
if "OBDScan2.1" in result:
    ser.write("93\r")
    sleep(8)
    result = ser.readline()
    if "P0" in result:
        ser.write("C5\r")
        sleep(1)
        result = ser.read(1)
        if "J" in result:
                ser.write("71\r")
                return True
        else:
                #print "Did not receive a J as a response\r\n"
                return False
    else:
        #print "Did not receive a P0 as a response\r\n"
        return False
else:
    #print "Could not talk to J1939 reader\r\n"
    return False
```

Setup Filters

Python Example

```
    delay = 0.1
    ser.write("\r")
    sleep(1)
    ser.write("C38000FEEE00\r")          #Engine Temp
    sleep(delay)
    ser.write("C38100F00400\r")          #Engine Speed
    sleep(delay)
    ser.write("C38200FEF100\r")          #Road Speed
    sleep(delay)
    ser.write("C38300FEF700\r")          #Battery Voltage
    sleep(delay)
    ser.write("C38400FEFC00\r")          #Fuel Level
    sleep(delay)
    ser.write("C38500F00500\r")          #Current Gear (not commonly used)
    sleep(delay)
    ser.write("C48000FFFF00\r")
    sleep(delay)
    ser.write("C48100FFFF00\r")
    sleep(delay)
    ser.write("71\r")
```

CLEAR DTC

Python Example

```
    ser.write("\r")
    sleep(.5)                            #Wait for OBDScan 2.1
    ser.write("C818EA00F9\r")            #Set up destination address
    sleep(.25)                           #Wait for VI
    ser.write("CED3FE00000000\r")        #Send clear DTC
    ser.write("\r71\r")                  #Resume monitoring
```