MobileView

MobileView J1939 Interface Guide

UM-TR-0420147

MobileView J1939 Interface Guide

This document applies to MobileView 7000 Series NVR

http://www.mobileviewvideo.com

Contact Technical Support:
Monday–Friday 6 am – 7 pm EST
Phone: 1.844.899.7366
Email: service@seon.com

Table of Contents

CHAPTER 1 GENERAL REQUIREMENTS 4

CHAPTER 2 GENERAL COMMUNICATION FOR SET-UP AND OPERATION 6

INITIALIZATION 6

NAME 6

NAME RESPONSE 7

CHAPTER 3 FAILURE DIAGNOSTIC INFORMATION FROM THE DVR 8

EXPLANATION OF DIAGNOSTICS PARAMETERS 9

CHAPTER 4 SIGNALS FROM J1939 TO THE DVR 10

MANUFACTURER-ASSIGNED DATA FROM J1939 NETWORK TO NVR 11

CHAPTER 5 INFORMATION QUERIES 13

QUERY RESPONSES BY DVR 13

DVR TIME AND DATE REQUESTS 13

GPS REQUESTS 14

SOFTWARE VERSION 14

Software Version Data 14

MAKE, MODEL, AND SERIAL NUMBER 15

Make Data 15

Model Data 15

Serial Number Data 15

QUERY RESPONSE MESSAGE DATA 16

CHAPTER 6 COM1939 LIBRARY 17

COM1939.H – HEADER FILE 18

FUNCTIONS 18

VARIABLES 18

COMMUNICATION BETWEEN NVR AND J1939 LIBRARY 20

EXAMPLE FILE MAIN.CPP 21

THE DIAGNOSTICS, MESSAGES, AND QUERIES TABLES 29

DIAGNOSTICS TABLE 29

MESSAGES TABLE 30

QUERIES TABLE 30

GLOSSARY 32

Chapter 1
General Requirements

A Can Bus using the J1939 protocol must use a conversion hardware known as the converter to interface and communicate. The converter translates information from the DVR or NVR via RS-232 and communicates electrically and logically to meet the requirements of the J1939 network. Functionality requirements for the DVR and converter are specified in this document.

NOTE: In this document, DVR refers to both NVR and ECU.

The following guide details the communication between the Converter and the DVR with the J1939 Network. The converter is used to translate the following information between the DVR to and from the J1939 network:

- General communication information for set up and operation
- Failure diagnostic information from the DVR
- Messages and information from other systems on the J1939 network to the DVR
- Queries from the J1939 network

A separate library or application will reside on the DVR and interface with the DVR firmware to communicate the J1939 information. This library or application is known as COM1939.

The DVR can process the MobileView binary file to coordinate the loading of J1939 codes and information. The name of the binary file will vary per application or end user, but it will always have an .mvb file extension. Reading and storing this file will allow for field configuration of new and unexpected items while allowing for fast transfer of information to a fleet of devices.

The binary file is a compilation of three different comma separated variables files (.csv files), which are used by the Windows J1939 test utility.

The assigned file names are:

- diagnostics.csv
- messages.csv
- queries.csv

The information extracted from the binary file is stored in a memory area shared between the DVR and the COM1939 library. A programming interface document (header file COM1939.h) will be provided with the library which assigns the shared memory areas and details where the information is stored.

A graphical interface to the DVR will display, control, and facilitate modification of the information uploaded from the binary file to allow for enabling and disabling of the different features.

Chapter 2
General Communication for Set-Up and Operation

Initialization

When the system boots up, the converter will establish a two way link with the DVR over RS-232 and retrieve information needed to establish a NAME. The converter will perform the operations needed to use this NAME to connect to the vehicle network. If the converter cannot establish communication with the network, the converter will continue trying to establish communication with the network and report communications failures to the DVR.

NAME

Each DVR connected to the vehicle network will have one NAME to be uniquely identified by the DVR. A DVR NAME is used in arbitration when claiming an ECU address and when multiple DVR attempts claim the same address.

The Web UI or MobileView Navigator is used to modify the parameters allowing the converter to establish a unique address for the DVR required by SAE-J1939. These parameters are the DVR Instance and the Vehicle System Instance. The DVR Instance and the Vehicle System Instance are configurable and memorized by the DVR during installation and commissioning of the device. The Vehicle System Instance and the ECU Instance default values will be set to 0. The DVR Instance is a 3-bit value starting at 0 and allows for 8 instances. The Vehicle System Instance is a 4-bit value starting at 0 and allows for 16 instances. If only one DVR is installed per vehicle, the default address will be used. If more than one DVR is to be installed per vehicle, as on a train, a unique number assigned using the unique DVR Instance and Vehicle System Instance numbers set up by the installer. These two numbers are used by the converter to establish a unique name for the DVR/converter pair with the J1939 network.

For example, a train may contain multiple cars with multiple DVRs in each but all on the same network. The DVR Instance represents the number of cars on the train and the Vehicle System Instance represents the order of the DVRs from front to back on each car (i.e. 1st car 2nd DVR or 3rd Car 1st DVR).

An identifier can be supplied as a serial number assigned by UTC.

NAME Response

The NVR will provide the following data:

- 3-bit ECU Instance= nNVRInstance in the COM1939.h file
- 4-bit Vehicle Instance= nVehicleInstance in the COM1939.h file
- 21-bit Identifier = ldentifier in the COM1939.h file

The user can set the DVR Instance and Vehicle System Instance from the DVR/NVR Web UI or the MobileView Navigator, and these values can be substituted into their respective variables.

Chapter 3
Failure Diagnostic Information from the DVR

The DVR provides self-diagnostic information to the converter. The converter provides Diagnostic Messages in accordance with SAE J1939 back to the network.
Below is an example of a spreadsheet presented as a table of information where the binary file is generated. These messages are shared with the converter upon the respective event occurrence.

The converter sends diagnostics code to the J1939 network. Sending the diagnostics code, whether a fault exists or not, is accomplished by the library using a one-second time interval. The converter uses the information as defined in the binary file.

The following is an example listing of diagnostic trouble codes from the DVR.

Table 1: Diagnostic Trouble Codes from DVR

Enable	Description	Sub Index	FMI (Decimal)
1	DVR General Fault	523002	31
1	DVR Temperature Below Operating Range	523003	1
1	DVR Temperature Above Operating Range	523004	0
1	DVR HDD Temperature Above Operation Range	523006	0
1	DVR PCB(s) Temperature Above Normal Operational Range	523007	0
0	DVR HDD Caddy Fault or Missing	523008	12
1	DVR Network 1 Erratic, Intermittent or Incorrect	523009	2
1	DVR Wireless, Erratic, Intermittent or Incorrect	523010	2
1	Video Channel 1 Erratic, Intermittent or Incorrect	523011	2
1	Video Channel 2 Erratic, Intermittent or Incorrect	523012	2
1	Video Channel 3 Erratic, Intermittent or Incorrect	523013	2
1	Video Channel 4 Erratic, Intermittent or Incorrect	523014	2
1	Video Channel 5 Erratic, Intermittent or Incorrect	523015	2
1	Video Channel 6 Erratic, Intermittent or Incorrect	523016	2
1	Video Channel 7 Erratic, Intermittent or Incorrect	523017	2
1	Video Channel 8 Erratic, Intermittent or Incorrect	523018	2
1	Video Channel 9 Erratic, Intermittent or Incorrect	523019	2
1	Video Channel 10 Erratic, Intermittent or Incorrect	523020	2
1	Video Channel 11 Erratic, Intermittent or Incorrect	523021	2
1	Video Channel 12 Erratic, Intermittent or Incorrect	523022	2
1	Video Channel 13 Erratic, Intermittent or Incorrect	523023	2
1	Video Channel 14 Erratic, Intermittent or Incorrect	523024	2
1	Video Channel 15 Erratic, Intermittent or Incorrect	523025	2
1	Video Channel 16 Erratic, Intermittent or Incorrect	523026	2
1	DVR Input Voltage Above Operation Range	523027	0
1	DVR Input Voltage Below Operation Range	523028	1
1	Device Fuse Fault or Missing	523029	12
1	Device Expansion Module Fault or Missing	523030	12

1	POE Module Erratic, Intermittent or Incorrect	523031	2
1	HDD Caddy Fan Erratic, Intermittent or Incorrect	523032	2
1	Accelerometer Erratic, Intermittent or Incorrect	523033	2
1	Device 5VDC Output Below Operation Range	523034	1
1	Device 12VDC Output Below Operation Range	523035	1
1	Event Saved	520210	31

Explanation of Diagnostics Parameters

File: diagnostics.csv

In the COM1939.h header file refer to structure pDiagnostics.

The NVR must set the variable nEnable in the corresponding row to true to report an error.

Column	**Enable**
Type	int
Description	Activates/deactivates current row
Value	0: Inactive, 1: Active

Column	**Description**
Type	char[50]
Description	Description as ASCII string
Value	Manufacturer -Assigned, max. 50 characters
Note	The description is only used for readability of the assigned parameters and not needed for the DVR.

Column	**Sub Index**
Type	long
Description	Assigned Sub Index
Value	These values are specified from 523002 to 523502 (0x7FAF9 to 0x7FB2A) but can span from 520192 to 524287.

Column	**FMI**
Type	int
Description	Failure Mode Identifier
Value	See Table 1: Diagnostic Trouble Codes from DVR

Chapter 4
Signals from J1939 to the DVR

Information from the J1939 network shall be accepted and stored as metadata by the DVR. This table is an example of how this information is formatted for communication to the DVR firmware through the COM1939 Library. This is the second sheet of the information in the binary file. This list can have up to 1000 entries allowing for expansion for multiple transit manufacturers implementation without having to change this file.

Table 2: J1939 Network to DVR Information

Enable	Index	Sub-Index	Description	Byte	Bit	Bit Length
0	65521	65415	Stop Request: Passenger	1	7	1
0	65521	65413	Kneeling in Process	1	5	1
1	65521	65411	Door Open: Rear	1	3	1
1	65521	65410	Door Open: Front	1	1	1
0	65521	65454	Door Emergency Switch: Front	2	7	1
0	65521	65453	Door Master Switch	2	5	1
0	65521	65416	Stop Request: Wheelchair	2	1	1
0	65521	65412	Wheelchair Ramp Deployed	4	5	1
0	65521	65437	Stop Pressure Switch: Rear 4 PSI	5	7	1
0	65521	65436	Stop Pressure Switch: Front 1.5 PSI	5	5	1
0	65521	65438	Stop Pressure Switch: 7 PSI	6	7	1
0	65521	65420	Turn Signal Lamps: Right Hand	7	5	1
0	65521	65421	Turn Signal Lamps: Left Hand	7	3	1
0	65521	65422	Hazard Lamps	7	1	1
0	65521	65423	Stop Lamps	8	7	1
0	65521	65429	Silent Alarm	8	5	1
1	65521	65430	Event Marker	8	3	1
1	65521	65477	Accelerometer Event	8	1	1
0	65522	65425	MRS Position: Night Run	1	7	1
0	65522	65522	MRS Position: Day Run	1	5	1
0	65522	65431	Bike Rack Deployed	2	7	1
0	65522	65426	MRS Position: Park	2	1	1
1	65523	50100	Door Open: Right Side Doors	1	7	1
0	65523	50101	Door Open: Left Side Doors	1	5	1
1	65523	50102	Door Open: Right Center	1	3	1
0	65523	50103	Door Open: Left Center	1	1	1
1	65523	50104	Door Open: Right Rear	2	7	1
0	65523	50105	Door Open: Left Rear	2	5	1

Enable	Index	Sub-Index	Description	Byte	Bit	Bit Length
0	65523	50111	Ramp Deployed: Right Center	3	1	1
0	65523	50112	Ramp Deployed: Left Center	4	7	1
0	65523	50113	Ramp Deployed: Right Rear	4	5	1
0	65523	50114	Ramp Deployed: Left Rear	4	3	1
0	65524	50143	Mobility Aid Ramp Deployed	4	5	1
1	65524	50155	Vehicle Ignition	7	5	1
1	65524	50140	Brake 35 PSI Activated	3	3	1
1	65524	50133	Driving Direction: Reverse	1	1	1
1	65524	50134	Driving Direction: Forward	2	7	1
1	65531	50176	Brake Depression Percent Signal	6	1	8
1	61443	91	Accelerator Pedal Position 1	2	1	8
1	65274	116	Brake Application Pressure	1	1	8

Manufacturer-Assigned Data from J1939 network to NVR

This data is also part of the binary file. In the COM1939.h header file, refer to structure pMessages.
The NVR displays the information on the corresponding screen. The following displays the format of the messages:

Column	**Enable**
Type	int
Description	Activates/deactivates Index and Sub Index in this row
Value	0: Inactive, 1: Active

Column	**Index**
Type	long
Description	Manufacturer-Assigned
Value	Manufacturer-Assigned

Column	**Sub Index**
Type	long
Description	Manufacturer-Assigned
Value	Manufacturer-Assigned

Column	**Byte**
Type	int
Description	Byte position in Index data field
Value	1 to 8

Column	**Bit**
Type	int
Description	Bit position in current byte
Value	1 to 8

Column	**Bit Length**
Type	int
Description	Data length in bits starting at bit position
Value	1 to 64

Column	Description
Type	char[50]
Description	PGN Description as ASCII string
Value	Manufacturer-Assigned, max. 50 characters

Column	Unit
Type	char[15]
Description	Unit assigned to data, e.g. % or in-lb
Value	ASCII string, max. 15 characters. Leave empty for counters or other data that without a unit

Column	Address
Type	int
Description	Address of device requesting the data
Value	1 to 252, application-specific

Chapter 5
Information Queries

The J1939 network can request metadata from the DVR such as Time and Date, GPS data, I/O status and other information. Equally, in a future version, the same data can be requested by the DVR from the J1939 network if it is needed by the DVR.

The following table displays 4 of the possible 100 queries. The remaining items on the list require further interface definition.

> **Note:** Further items will be added in the future, and each individual addition will require program changes in the DVR firmware as well as the COM1939 library. The DVR firmware must be extended to provide the requested information.

Table 3: Queries from J1939 Network

Enable	Index	Description
1	65242	Device Firmware ID by Request
1	65259	Device Make, Model, and Serial Number Request
1	65254	Device Date-Time by Request
1	65267	Device GPS Data by Request

Query Responses by DVR

The DVR communicates the query response to the COM1939 library according to the previous table. The requested data is described in greater detail in this document. Some of the data (e.g. software ID) need only be stored at program start up. Others including GPS, data and time, must be stored as soon as they change.

DVR Time and Date Requests

The J1939 network will request the time and date information when it is needed, no more than once every 30 minutes to validate internal data and reset or calibrate date and time information. The requests for time and date information will be sent to the converter. The converter will read the time and date information from the area of memory shared with the DVR.

In the COM1939.h header file, refer to variable pTimeAndDate, which represents an array of eight bytes. The DVR will load the information as follows every time a change occurs:

Byte 1: 0x00 to 0xEF - Seconds 0 to 59.75 in .25 sec increments
Byte 2: 0x00 to 0x2F - Minutes in 1 min per bit
Byte 3: 0x00 to 0x17 - Hours in 1 hour increment with a range of 0 to 23 hours
Byte 4: 0x01 to 0x0C - Month in 1 month increment with a range of 1 to 12

Byte 5: 0x01 to 0x7F - Day is .25 days/bit; valid range is .25 to 31.75
Byte 6: 0x00 to 0xFF - Year in 1 year/bit starting with 1985 to 2236
Byte 7: 0x42 to 0xB8 - Local Minute Offset - 1min/bit with -125 minute offset. -59 to +59 minutes (0x00 = -125 and 0xFA = +125)
Byte 8: 0x66 to 0x94 - Local Hour Offset - 1 Hour/bit with -125 hour offset. 23 to +23 hours (-125 = 0x00 and +125 = 0xFA)

GPS Requests

The converter is used to assess GPS information from the DVR using the following format:

Byte 1: Byte 2: Byte 3: Byte 4:	Bytes 1 through 4: 10^{-7} degrees per bit with -210 degree offset 0x00000000 = -210 South to 0xFAFEE27A = +211.108122 North
Byte 5: Byte 6: Byte 7: Byte 8:	Bytes 5 through 8: 10^{-7} degrees per bit with -210 degree offset 0x00000000 = -210 West to 0xFAFEE27A = +211.108122 East

In the COM1939.h header file refer to variable pGPS, which represents an array of eight bytes. The DVR will fill the information every time a change occurs.

Response Data Frame example for 211.108122 Degrees North and -210 degrees

East/West				**North/South**			
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x00	0x00	0x00	0x00	0xFA	0xFE	0xE2	0x7A

Example: For the Museum of Science and Industry in Chicago IL located 41.790566 (North) and -87.583059(West): = ((210+41.790566)*1000000) - North by ((210-87.583059)*10000000) – East. These numbers convert in hex to the following:

East/West				**North/South**			
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x48	0xF7	0x57	0xC2	0x0F	0x5F	0x53	0xFC

This information is relayed in its existing format split into one-byte segments lowest to highest. For instance, the response for GPS coordinates of 41.790566 North and -87.583059 West, back from the Converter yields the following input:

East/West				**North/South**			
Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8
0x48	0xF7	0x57	0xC2	0x0F	0x5F	0x53	0xFC

Software Version

Upon query from the converter, the DVR provides the software revision in the format as shown below.

Software Version Data

The data is represented by hexadecimal values for the ASCII numbers and letters representing the revision code followed by the hexadecimal ASCII code for Asterisk (*) for the end of the field. The software version shall be conveyed in a field of 8 Bytes. A version of M.23.45 shall be sent as follows:

Byte 1 “M”	Byte 2 “.”	Byte 3 “2”	Byte 4 “3”	Byte 5 “.”	Byte 6 “4”	Byte 7 “5”	Byte 8 “*”
0x4D	0x2E	0x32	0x33	0x2E	0x34	0x35	0x2A

Where the software version is less than the allotted 7 Byte field, “0” hex shall be sent to fill out the unused bytes. Please see example below for a revision code of 1.14:

Byte 1 “0”	Byte 2 “0”	Byte 3 “0”	Byte 4 “1”	Byte 5 “.”	Byte 6 “1”	Byte 7 “4”	Byte 8 “*”
0x30	0x30	0x30	0x31	0x2E	0x31	0x34	0x2A

In the COM1939.h header file refer to variable sSoftwareVersion, which represents an ASCII string of 9 characters.

Make, Model, and Serial Number

Upon query from the converter, the DVR provides the Make, Model, and Serial Number in the format as shown below starting with the ID header in its 4 individual byte format.

The Make, Model, and Serial Number are sent as a concatenated stream consisting of all three items with the “*” delimiter between as shown below.

The Make conforms to a 5-byte frame where the Model and Serial Numbers can use up to 14 Bytes each. Each item is separated by the Asterisk (“*”) to identify end of field.

Make Data

The Make shall be transmitted to the converter as below. The 5 bytes are followed by the hexadecimal ASCII value for “*” representing the end of the field in the 6th byte.

Byte 1 “M”	Byte 2 “V”	Byte 3 “I”	Byte 4 “E”	Byte 5 “W”	Byte 6 “*”
0x40	0x56	0x49	0x45	0x57	0x2A

Model Data

Following the Make information, the model number is appended as shown below for MVH3004:

Byte 1 “M”	Byte 2 “V”	Byte 3 “H”	Byte 4 “3”	Byte 5 “0”	Byte 6 “0”	Byte 7 “4”	Byte 8 “*”
0x4D	0x56	0x50	0x33	0x30	0x30	0x34	0x2A

Serial Number Data

The serial number data shall be 14 bytes long and follows the “*” delimiter of the model (see above). The Serial Number shall be represented by the ACSII codes of the numbers or letters that make up the serial number. If the actual serial number is less that 14 bytes, the most significant unfilled bytes shall be filled with the ASCII value for “0” = 0x30. The field shall end with the last two bytes filled with the ASCII value for the “*” delimiter.

Example of serial number “MV6624538439 shall be represented as shown directly below:

Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8

“0”	“0”	“M”	“V”	“6”	“6”	“2”	“4”
0x30	0x30	0x4D	0x56	0x36	0x36	0x32	0x34

Byte 9 “5”	Byte10 “3”	Byte 11 “8”	Byte 12 “4”	Byte 13 “3”	Byte 14 “9”	Byte 15 “*”	Byte 16 “*”
0x35	0x33	0x38	0x34	0x33	0x39	0x2A	0x2A

In the COM1939.h header refer to variable sMakeModelSerialNumber, which represents an ASCII string of 40 characters.

Query Response Message Data

In the COM1939.h header file refer to structure pQueries.

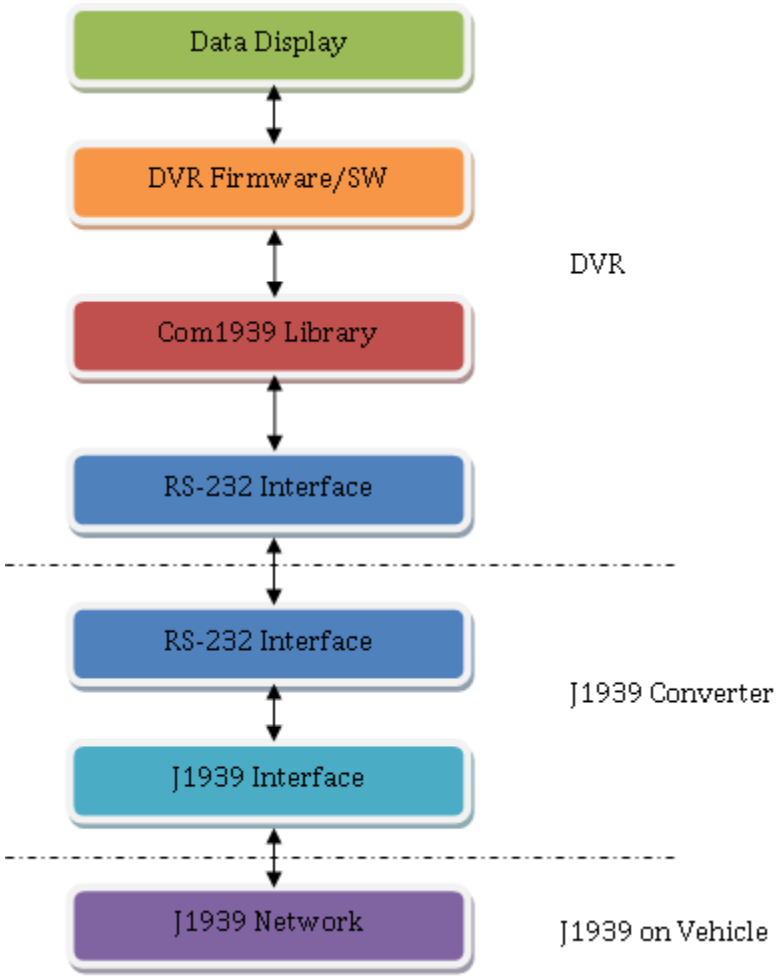
Column	**Index**
Type	long
Description	Requested Index

Column	**Description**
Type	char[50]
Description	Description as ASCII string
Value	Manufacturer/J1939-Assigned, max. 50 characters
Note	The description is only used for readability of the assigned parameters and is not needed for the NVR.

Chapter 6
COM1939 Library

The J1939 Converter connects the Linux Computer (NVR) to the vehicle using a RS-232 serial connection. The communication between the module and the NVR is managed through a protocol allowing the NVR to retrieve information from the vehicle network.

The image below displays the connection between the vehicle network and the NVR including the COM1939 Library.



The COM1939 library is designed to scan the vehicle network roughly every 100 milliseconds, assuring proper polling of the data from the vehicle.
The serial connection is assumed as COM1. The DVR operating system needs to assume control and have the access rights of the RS-232 port. For Linux, this is represented by the Linux file "/dev/ttyS0".

COM1939.h – Header File

The header file contains the available function information and variables stored in the COM1939 class.

Functions

- Initialize(): Initializes the RS232 port plus internal settings. This function must be called once after start of the NVR.
- Operate(): This function must be called every 100 milliseconds to assure proper timing and message reception.
- Terminate(): This function must be called when the NVR program terminates.

Variables

The following variables are only modified during the Initialize() function:

- bool bCommunicationError: When true indicates a global error with the RS232 COM port.
- int nErrNum: Error number related to the RS232 COM port as reported by the Linux OS.
- bool bErrorOpenCOMPort: When true indicates an error during COM port initialization.
- bool bErrorCOMPortSettings: When true indicates an error after attempting to apply the COM port settings.

```
#ifndef COM1939_H
#define COM1939_H

#define byte unsigned char

// Constants ------------------------------------------------------------------
// ----------------------------------------------------------------------------
const int SYSTEM_LOOP_TIME = 100000;         // microseconds
const int DIAGNOSTICS = 100;
const int MESSAGES = 100;
const int QUERIES = 20;
const int TBLEND = 255;

// System Class ---------------------------------------------------------------
// ----------------------------------------------------------------------------
class COM1939
{
    public:
        COM1939();
        virtual ~COM1939();

        // COM1939 Version Number
        char sCOM1939Version[8];
        //
        // V 1.01.00 03-16-2015
        // ------------------------------------------------------------------
        // Version number recording
        // Heartbeat and version numbers from jCOM1939/RS232 converter
        // Enable / Disable without reboot
        // Active / Listen-Only mode without reboot
        // Updating J1939 converter when enabled messages change
        //
        // V 1.02.00 04-05-2015
        // ------------------------------------------------------------------
        // Added FMI code to diagnostics
        // Memory allocation optimization
        //
        // V 1.03.00 04-16-2015
        // ------------------------------------------------------------------
        // Removed usleep() function due to performance tests with DVR/NVR
        // This required changes to the RS232 COM port access functionality
        //
        // V 1.03.01 06-06-2015
        // ------------------------------------------------------------------
        // Added "Do not access" flags to avoind read/write collisions between threads
        // Modified function com1939.CheckMessage (called in main.cpp) to print actual data on console
        // Refinement of address claim procedure
        //
```

```
        // Thread
        bool bAccessJCOM1939;               // true = jCOM1939 is reading/writing data.
        bool bAccessNVR;                    // true = NVR/DVR is reading/writing data.
                                            // (set to false when calling JCOM1939.Initialize function)

        // jCOM1939/RS232
        bool bjCOM1939Heartbeat;            // Will toggle every second
        char sjCOM1939HWVersion[8];         // Will be overwritten by heartbeat message
        char sjCOM1939SWVersion[8];         // Will be overwritten by heartbeat message

        // COM Port
        int fileCOMPort;                    // COM port file handle
        char sPort[20];                     // String representation of used port

        // Error Management
        bool bCommunicationError;           // Global error flag
        int nErrNum;

        bool bErrorOpenCOMPort;             // True = Error opening COM port
        bool bErrorCOMPortSettings;         // True = Error applying COM port settings

        // Protocol Functions
        int Initialize(char* sCOMPort, long lSystemLoopTime, bool bListenOnly);     // For instance
"/dev/ttyS0"
                                                                                    // System loop time in
microseconds
        void Operate(void);
        void Terminate(void);
        void Disable(void);
        void Enable(void);
        void ActiveMode(void);
        void ListenOnlyMode(void);
        void ReportMessageChange(void);

        // Only used for test & simulation purposes during development
        bool CheckMessage(char*);
        void SetDiagnostics(int);
        void CopyDataToTable(long, byte*);
        void SetProtocolParameters(void);

        // Initialization data filled by NVR
        int nNVRInstance;                   // 0 to 7 - To identify multiple NVRs; default = 0
        int nVehicleInstance;               // 0 to 15 - To identify multiple vehicle systems; default = 0
        long lIdentifier;                   // 21 bit Identifier (according to info from UTC)

       // Runtime data from NVR (updated when necessary)
        unsigned char pTimeAndDate[8];      // Byte 1: 0x00 to 0xEF - Seconds 0 to 59.75 in .25 sec
increments
                                            // Byte 2: 0x00 to 0x2F - Minutes in 1 min per bit
                                            // Byte 3: 0x00 to 0x17 - Hours in 1 hour increment with a
range of 0 to 23 hours
                                            // Byte 4: 0x01 to 0x0C - Month in 1 month increment with a
range of 1 to 12
                                            // Byte 5: 0x01 to 0x7F - Day is .25 days/bit; valid range is
.25 to 31.75
                                            // Byte 6: 0x00 to 0xFF - Year in 1 year/bit starting with 1985
to 2236
                                            // Byte 7: 0x42 to 0xB8 - Local Minute Offset - 1min/bit with -
125 minute offset. -59 to
                                            //                      +59 minutes (0x00 = -125 and 0xFA =
+125)
                                            // Byte 8: 0x66 to 0x94 - Local Hour Offset - 1 Hour/bit with -
125 hour offset. 23 to +23
                                            //                      hours (-125 = 0x00 and +125 = 0xFA)

        unsigned char pGPS[8];              // According to information in the UTC document

        // Response by NVR as reaction to previous error events
        bool bEventSaved;

        // Information received from AVM System
        bool bTimeDateReceived;             // true = AVM's time & date received; READ-ONLY! DO NOT MODIFY!
        unsigned char pAVMTimeAndDate[8];   // Byte 1: 0x00 to 0xEF - Seconds 0 to 59.75 in .25 sec
increments
                                            // Byte 2: 0x00 to 0x2F - Minutes in 1 min per bit
                                            // Byte 3: 0x00 to 0x17 - Hours in 1 hour increment with a
range of 0 to 23 hours
                                            // Byte 4: 0x01 to 0x0C - Month in 1 month increment with a
range of 1 to 12
```

```
                                        // Byte 5: 0x01 to 0x7F - Day is .25 days/bit; valid range is
.25 to 31.75
                                        // Byte 6: 0x00 to 0xFF - Year in 1 year/bit starting with 1985
to 2236
                                        // Byte 7: 0x42 to 0xB8 - Local Minute Offset - 1min/bit with -
125 minute offset. -59 to
                                        //                        +59 minutes (0x00 = -125 and 0xFA =
+125)
                                        // Byte 8: 0x66 to 0x94 - Local Hour Offset - 1 Hour/bit with -
125 hour offset. 23 to +23
                                        //                        hours (-125 = 0x00 and +125 = 0xFA)

        // Tables according to csv files read by NVR
        // ------------------------------------------------------------------------

        // Project Description as derived from the binary file
        char sProject[41];  // Max. 40 characters plus termination character "\0"

        // Diagnostics: Messages from MVR to J1939 vehicle bus
        struct com1939Diagnostics
        {
            byte nEnable;
            char sDescription[61];
            long lSubIndex;
            int nFMI;

            bool bErrorActive;              // To be set by NVR (true/false)
        };
        com1939Diagnostics p1939Diagnostics[DIAGNOSTICS];

        // ------------------------------------------------------------------------
        // Messages: Messages from J1939 vehicle bus to MVR
        struct com1939Messages
        {
            byte nEnable;
            long lIndex;
            long lSubIndex;
            char sDescription[61];
            byte nByte;
            byte nBit;
            byte nBitLength;

            bool bDataReceived;                 // Set to true by COM1939; must be reset by NVR upon
reading
                                                // Note: The flag does NOT indicate a change in data,
meaning the
                                                //       same data may be received with everry update
cycle.
            byte nData;                         // Data received from J1939 vehicle bus
        };
        com1939Messages p1939Messages[MESSAGES];

        struct com1939Queries
        {
            byte nEnable;
            long lIndex;
            char sDescription[61];

            byte nResponseLen;                  // Number of bytes filled by NVR
            byte sResponse[50];                 // Responses to be filled by NVR; they can be much longer
than 8 bytes
        };
        com1939Queries p1939Queries[QUERIES];

   protected:

   private:
        void ScanAndSetMessageFilters(bool);

};

#endif // COM1939_H
```

Communication between NVR and J1939 Library

The COM1939 class header file (COM1939.h) contains of three sections:

1. Initialization data from NVR to COM1939

These data are NVR Instance, Vehicle Instance, Identifier, Software Version, Make, Model, and Serial Number as explained in the previous chapters.

2. Runtime data from NVR to COM1939
   These data are Time and Date, and the GPS information as explained in the previous chapters.
3. Status messages from COM1939 to NVR
   This data includes Turn Signal Lamps Right Hand, Turn Signal Lamps Left Hand, Stop Lamps, Silent Alarm, Event Marker, and Accelerometer Event.

Example file main.cpp

The main.cpp file represents a demo example of how to incorporate the COM1939 library functions. Main.cpp represents a Linux console application that is used to display the data on a terminal screen.

```
#include <iostream>
#include <stdio.h>
#include <errno.h>
#include <termios.h>
#include <unistd.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

#include "include/COM1939.h"

using namespace std;

//
// V 1.01.00 03-16-2015
// ---------------------------------------------------------------------
// Version number recording
// Heartbeat and version numbers from jCOM1939/RS232 converter
// Enable / Disable without reboot
// Active / Listen-Only mode without reboot
// Updating J1939 converter when enabled messages change
//
//

int kbhit(char* key)
{
    struct termios oldt;
    struct termios newt;
    int ch;
    int oldf;

    tcgetattr(STDIN_FILENO, &oldt);
    newt = oldt;
    newt.c_lflag &= ~(ICANON | ECHO);
    tcsetattr(STDIN_FILENO, TCSANOW, &newt);
    oldf = fcntl(STDIN_FILENO, F_GETFL, 0);
    fcntl(STDIN_FILENO, F_SETFL, oldf | O_NONBLOCK);

    ch = getchar();
    *key = (char)ch;

    tcsetattr(STDIN_FILENO, TCSANOW, &oldt);
    fcntl(STDIN_FILENO, F_SETFL, oldf);

    if(ch != EOF)
    {
        //ungetc(ch, stdin);
        return 1;
    }

    return 0;

}

// Reference to the COM1939 module
static COM1939 com1939;
```

```
//-NOTE----------------------------------------------------------------------
// INCORPORATE THE FOLLOWING CODE INTO YOUR PROJECT
// THE CODE READS THE MOBILEVIEW BINARY FILE
// ---------------------------------------------------------------------------

#define TESTPRINT   0
#define CHECKSUM    0

char pScramble[] = {
        'a', 'z',
        'b', 'y',
        'c', 'x',
        'd', 'w',
        'e', 'v',
        'f', 'u',
        'g', 't',
        'h', 's',
        'i', 'r',
        'j', 'q',
        'k', 'p',
        'l', 'o',
        'm', 'n',
        'n', 'm',
        'o', 'l',
        'p', 'k',
        'q', 'j',
        'r', 'i',
        's', 'h',
        't', 'g',
        'u', 'f',
        'v', 'e',
        'w', 'd',
        'x', 'c',
        'y', 'b',
        'z', 'a',
        '0', '9',
        '1', '8',
        '2', '7',
        '3', '6',
        '4', '5',
        '5', '4',
        '6', '3',
        '7', '2',
        '8', '1',
        '9', '0' };

const int nScrambleCount = 36;

//-FUNCTION----------------------------------------------------------------------
// Routine     : UnscrambleString
// Description : Uncrambles a string
// Returncode  : -
// ---------------------------------------------------------------------------
void UnscrambleString(char* sString)
{
    for (int nIndex = 0; nIndex < (int)strlen(sString); nIndex++)
    {
        for (int nIndex1 = 0; nIndex1 < nScrambleCount * 2; nIndex1 += 2)
        {
            if (sString[nIndex] == pScramble[nIndex1])
            {
                sString[nIndex] = pScramble[nIndex1 + 1];
                break;

            }// end if

        } //end for

    }// end for

}// end UnscrambleString

//-FUNCTION----------------------------------------------------------------------
// Routine     :
// Description : Computes the checksum of a message and returns the
//               2 character ASCII equivalents
// Returncode  : true = Checksum okay, false = Checksum failed
// ---------------------------------------------------------------------------
bool ComputeCheckSumString(char* sMsg, int nLen)
```

```
{
    // Declarations
    bool bRetCode;
    char sNumber[5];
    char sCheckSum[5];
    byte nCheckSum = 0;

    // Default settings
    bRetCode = true;

    // Create the checksum
    for (int nIndex = 0; nIndex < nLen; nIndex++)
        nCheckSum += (byte)sMsg[nIndex];

    nCheckSum = (byte)((~(int)nCheckSum) + 1);
    sprintf(sNumber, "%X", (int)nCheckSum);

    // Check if leading zero is needed
    if (strlen(sNumber) == 1)
    {
        strcpy(sCheckSum, "0");
        strcat(sCheckSum, sNumber);

    }// end if
    else
        strcpy(sCheckSum, sNumber);

    // Compare the checksum
    if(sMsg[nLen] != sCheckSum[0])
        bRetCode = false;
    else if(sMsg[nLen+1] != sCheckSum[1])
        bRetCode = false;

    // Return the result
    return bRetCode;

}// end ComputeCheckSumString

const int IDX_ProjectMarker = 1;
const int IDX_Enable = 1;
const int IDX_Index = 2;
const int IDX_FMI = 8;
const int IDX_SubIndex = 8;

const int IDX_ProjectDescription = 1;
const int IDX_DiagnosticsDescription = 10;
const int IDX_MessagesDescription = 14;
const int IDX_QueriesDescription = 8;

const int IDX_ProjectChecksum = 41;
const int IDX_DiagnosticsChecksum = 68;
const int IDX_MessagesChecksum = 77;
const int IDX_QueriesChecksum = 68;

const int IDX_MessagesByte = 74;
const int IDX_MessageBit = 75;
const int IDX_MessageBitLength = 76;

const int LEN_ProjectDescription = 40;
const int LEN_Number = 6;
const int LEN_Description = 60;

const int MSGLEN_Diagnostics = 69;
const int MSGLEN_Messages = 76;
const int MSGLEN_Queries = 67;

// ----------------------------------------------------------------
// Function:    ReadMobileViewBinaryFile
//              Reads the binary file and stores the data into the
//              corresponding structures
// Returncode : 0 - Successful
//              1 - Cannot open file
//              2 - File is corrupted (Checksum error)
// ----------------------------------------------------------------
int ReadMobileViewBinaryFile(char* sFilePath, char* sFileName)
{
    // Declarations
    int nRetCode;

    char sLine[100];
```

```
    //char sCheckSum[5];

    int nDiagnosticsRecords = 0;
    int nMessagesRecords = 0;
    int nQueriesRecords = 0;

    FILE* pBinaryFile;

    // Default settings
    nRetCode = 1;

    // Open the file
    strcat(sFilePath, "/");
    pBinaryFile = fopen(strcat(sFilePath, sFileName), "r");
    if(pBinaryFile != NULL)
    {
        // File was opened sucessfully
        nRetCode = 0;

        // Get the next line from the file
        while(fgets(sLine, 100, pBinaryFile) != NULL)
        {
            // Unscramble the line
            UnscrambleString(sLine);

            // Store data according to the line identifier
            switch(sLine[0])
            {

                case 'P':   // Project Description

                    strncpy(com1939.sProject, &sLine[IDX_ProjectDescription], LEN_ProjectDescription);
#if TESTPRINT == 1
                    printf("Project Description: %s\n\r", com1939.sProject);
#endif

#if CHECKSUM == 1
                    if(ComputeCheckSumString(&sLine[IDX_ProjectMarker], LEN_ProjectDescription) == false)
                        nRetCode = 2;
#endif
                    break;

                case 'D':   // Diagnostics

                    // Enable
                    if(sLine[IDX_Enable] == '0')
                        com1939.p1939Diagnostics[nDiagnosticsRecords].nEnable = 0;
                    else
                        com1939.p1939Diagnostics[nDiagnosticsRecords].nEnable = 1;

                    // SubIndex
                    // Note: In the following we use IDX_Index, because that is the format in this case
                    com1939.p1939Diagnostics[nDiagnosticsRecords].lSubIndex =
                        ((sLine[IDX_Index] - '0') * 100000) +
                        ((sLine[IDX_Index+1] - '0') * 10000) +
                        ((sLine[IDX_Index+2] - '0') * 1000) +
                        ((sLine[IDX_Index+3] - '0') * 100) +
                        ((sLine[IDX_Index+4] - '0') * 10) +
                        (sLine[IDX_Index+5] - '0');

                    // FMI
                    com1939.p1939Diagnostics[nDiagnosticsRecords].nFMI =
                        ((sLine[IDX_FMI] - '0') * 10) +
                        (sLine[IDX_FMI+1] - '0');

                    // Description
                    strncpy(com1939.p1939Diagnostics[nDiagnosticsRecords].sDescription,
&sLine[IDX_DiagnosticsDescription], LEN_Description);

#if TESTPRINT == 1
                    printf("Diagnostics: %d %lu %d %s\r",
                            com1939.p1939Diagnostics[nDiagnosticsRecords].nEnable,
                            com1939.p1939Diagnostics[nDiagnosticsRecords].lSubIndex,
                            com1939.p1939Diagnostics[nDiagnosticsRecords].nFMI,
                            com1939.p1939Diagnostics[nDiagnosticsRecords].sDescription
                          );
#endif
                    // Increase the record counter
                    nDiagnosticsRecords++;
#if CHECKSUM == 1
```

```
                    if(ComputeCheckSumString(&sLine[IDX_Enable], MSGLEN_Diagnostics) == false)
                        nRetCode = 2;
#endif
                    break;

                case 'M':   // Messages

                    // Enable
                    if(sLine[IDX_Enable] == '0')
                        com1939.p1939Messages[nMessagesRecords].nEnable = 0;
                    else
                        com1939.p1939Messages[nMessagesRecords].nEnable = 1;

                    // Index
                    com1939.p1939Messages[nMessagesRecords].lIndex =
                        ((sLine[IDX_Index] - '0') * 100000) +
                        ((sLine[IDX_Index+1] - '0') * 10000) +
                        ((sLine[IDX_Index+2] - '0') * 1000) +
                        ((sLine[IDX_Index+3] - '0') * 100) +
                        ((sLine[IDX_Index+4] - '0') * 10) +
                        (sLine[IDX_Index+5] - '0');

                    // SubIndex
                    com1939.p1939Messages[nMessagesRecords].lSubIndex =
                        ((sLine[IDX_SubIndex] - '0') * 100000) +
                        ((sLine[IDX_SubIndex+1] - '0') * 10000) +
                        ((sLine[IDX_SubIndex+2] - '0') * 1000) +
                        ((sLine[IDX_SubIndex+3] - '0') * 100) +
                        ((sLine[IDX_SubIndex+4] - '0') * 10) +
                        (sLine[IDX_SubIndex+5] - '0');

                    // Description
                    strncpy(com1939.p1939Messages[nMessagesRecords].sDescription,
&sLine[IDX_MessagesDescription], LEN_Description);

                    // Byte
                    com1939.p1939Messages[nMessagesRecords].nByte = sLine[IDX_MessagesByte] - '0';

                    // Bit
                    com1939.p1939Messages[nMessagesRecords].nBit = sLine[IDX_MessageBit] - '0';

                    // Bit Length
                    com1939.p1939Messages[nMessagesRecords].nBitLength = sLine[IDX_MessageBitLength] - '0';

#if TESTPRINT == 1
                    printf("Messages: %d %d %lu %lu %d %d %d %s\r",
                            nMessagesRecords,
                            com1939.p1939Messages[nMessagesRecords].nEnable,
                            com1939.p1939Messages[nMessagesRecords].lIndex,
                            com1939.p1939Messages[nMessagesRecords].lSubIndex,
                            com1939.p1939Messages[nMessagesRecords].nByte,
                            com1939.p1939Messages[nMessagesRecords].nBit,
                            com1939.p1939Messages[nMessagesRecords].nBitLength,
                            com1939.p1939Messages[nMessagesRecords].sDescription
                          );
#endif
                    // Increase the record counter
                    nMessagesRecords++;

#if CHECKSUM == 1
                    if(ComputeCheckSumString(&sLine[IDX_Enable], MSGLEN_Messages) == false)
                        nRetCode = 2;
#endif

                    break;

                case 'Q':   // Queries

                    // Enable
                    if(sLine[IDX_Enable] == '0')
                        com1939.p1939Queries[nQueriesRecords].nEnable = 0;
                    else
                        com1939.p1939Queries[nQueriesRecords].nEnable = 1;

                    // Index
                    com1939.p1939Queries[nQueriesRecords].lIndex =
                        ((sLine[IDX_Index] - '0') * 100000) +
                        ((sLine[IDX_Index+1] - '0') * 10000) +
                        ((sLine[IDX_Index+2] - '0') * 1000) +
                        ((sLine[IDX_Index+3] - '0') * 100) +
```

```
                        ((sLine[IDX_Index+4] - '0') * 10) +
                        (sLine[IDX_Index+5] - '0');

                    // Description
                    strncpy(com1939.p1939Queries[nQueriesRecords].sDescription,
&sLine[IDX_QueriesDescription], LEN_Description);

#if TESTPRINT == 1
                    printf("Queries: %d %lu %s\r\n",
                            com1939.p1939Queries[nQueriesRecords].nEnable,
                            com1939.p1939Queries[nQueriesRecords].lIndex,
                            com1939.p1939Queries[nQueriesRecords].sDescription
                          );
#endif
                    // Increase the record counter
                    nQueriesRecords++;

#if CHECKSUM == 1
                    if(ComputeCheckSumString(&sLine[IDX_Enable], MSGLEN_Messages) == false)
                        nRetCode = 2;
#endif
                    break;

            }// end switch

            // End program in case of checksum error
            if(nRetCode == 2)
                break;

        }// end while

        // Terminate all tables
        com1939.p1939Diagnostics[nDiagnosticsRecords].nEnable = TBLEND;
        com1939.p1939Messages[nMessagesRecords].nEnable = TBLEND;
        com1939.p1939Queries[nQueriesRecords].nEnable = TBLEND;

        // Close the file
        fclose(pBinaryFile);

    }// end if

    return nRetCode;

}// end ReadMobileViewBinaryFile

//-NOTE-------------------------------------------------------------------------
// END OF CODE
// ------------------------------------------------------------------------------

int main()
{
    // Declarations
    bool bAllowHeartbeat = false;
    bool bHeartbeat = false;

    char sCOMPort[20];
    char sData[100];

    long lCount0 = 0;
    long lCount1 = 0;

    // Initialize the port and protocol settings
    strcpy(sCOMPort, "/dev/ttyS0");
    //strcpy(sCOMPort, "/dev/ttyUSB0");
    com1939.Initialize(&sCOMPort[0], SYSTEM_LOOP_TIME, false);

    // Simulate some data
    strcpy(sData, "FIRMWARE");
    for(int nIndex = 0; nIndex < 8; nIndex++)
        com1939.p1939Queries[0].sResponse[nIndex] = sData[nIndex];
    com1939.p1939Queries[0].nResponseLen = 8;

    strcpy(sData, "MVIEW*0MN70*011427002**");
    for(unsigned int nIndex = 0; nIndex < strlen(sData); nIndex++)
        com1939.p1939Queries[1].sResponse[nIndex] = sData[nIndex];
    com1939.p1939Queries[1].nResponseLen = 23;

    strcpy(sData, "DATETIME");
    for(int nIndex = 0; nIndex < 8; nIndex++)
        com1939.p1939Queries[2].sResponse[nIndex] = sData[nIndex];
```

```
    com1939.p1939Queries[2].nResponseLen = 8;

    strcpy(sData, "GPS_DATA");
    for(int nIndex = 0; nIndex < 8; nIndex++)
        com1939.p1939Queries[3].sResponse[nIndex] = sData[nIndex];
    com1939.p1939Queries[3].nResponseLen = 8;

    if(com1939.bCommunicationError == false)
        printf("COM Port Init OK. %s\n\r", com1939.sPort);
    else if(com1939.bErrorOpenCOMPort == true)
        printf("Unable to open COM port: %s - ErrNo: %d\n\r", com1939.sPort, com1939.nErrNum);
    else if(com1939.bErrorCOMPortSettings == true)
        printf("Unable to apply COM port settings: %s - ErrNo: %d\n\r", com1939.sPort, com1939.nErrNum);

    // Read the MobileView binary file and store it into the corresponding structures
    char sCurrentPath[120];
    getcwd(sCurrentPath, 120);
    char sFile[] = {"Test.mvb"};
    int nRetCode = ReadMobileViewBinaryFile(sCurrentPath, sFile);
    if(nRetCode == 0)
    {
        // Main loop
        bool bExit = false;
        while(1)
        {
            // Wait for assigned system loop time period
            usleep(SYSTEM_LOOP_TIME);

            // Call the COM1939 protocol
            com1939.Operate();

            // Check for messages from the J1939 vehicle network
            char sStr[250];
            if(com1939.CheckMessage(sStr) == true)
                printf("%s", sStr);

            // Diagnostic message simulation
            char key;
            if(kbhit(&key))
            {
                switch(key)
                {
                    case 'x':       // Exit the program
                        com1939.Terminate();
                        bExit = true;
                        break;

                    case '0':       // Cleart all error messages
                        com1939.SetDiagnostics(-1);
                        break;

                    case '1':       // Toggle: DVR General Fault
                        com1939.SetDiagnostics(0);
                        break;

                    case '2':       // Toggle: DVR Temperature Below Operational Range
                        com1939.SetDiagnostics(1);
                        break;

                    case '3':       // Toggle: DVR Temperature Above Operational Range
                        com1939.SetDiagnostics(2);
                        break;

                    case '4':       // Toggle: DVR HDD Temperature Above Operation Range
                        com1939.SetDiagnostics(3);
                        break;

                    case '5':       // Set the Event Saved flag
                        com1939.bEventSaved = true;
                        break;

                    case 'v':       // Print the version numbers
                        printf("\n\rJ1939 Lib: %s\n\rjCOM1939-HW: %s\n\rjCOM1939-SW: %s\n\r",
                               com1939.sCOM1939Version, com1939.sjCOM1939HWVersion,
com1939.sjCOM1939SWVersion);
                        break;

                    case 'h':       // Enable/disable heartbeat display
                        bAllowHeartbeat = !bAllowHeartbeat;
```

```
                        if(bAllowHeartbeat == true)
                            printf("=ON\n\r");
                        else
                            printf("=OFF\n\r");

                        break;

                    case 'd':
                        com1939.Disable();
                        printf(" - Disabled.\n\r");
                        break;

                    case 'e':
                        com1939.Enable();
                        printf(" - Enabled.\n\r");
                        break;

                    case 'a':
                        com1939.ActiveMode();
                        printf(" - Active Mode.\n\r");
                        break;

                    case 'l':
                        com1939.ListenOnlyMode();
                        printf(" - Listen-Only Mode.\n\r");
                        break;

                    case 'r':
                        com1939.ReportMessageChange();

                        // Toggle one entry in the messages' table - entry = 65274 Brake Application
Pressure
                        if(com1939.p1939Messages[39].nEnable == 0)
                            com1939.p1939Messages[39].nEnable = 1;
                        else
                            com1939.p1939Messages[39].nEnable = 0;

                        break;


                }// end switch

            }// end if

            if(bHeartbeat != com1939.bjCOM1939Heartbeat)
            {
                bHeartbeat = com1939.bjCOM1939Heartbeat;

                if(com1939.bjCOM1939Heartbeat == true)
                    lCount0++;
                else
                    lCount1++;

                // Display the heartbeat
                if(bAllowHeartbeat == true)
                    printf("True: %ld False: %ld\n\r", lCount0, lCount1);

                    /*
                    if(bHeartbeat != com1939.bjCOM1939Heartbeat)
                    {
                        bHeartbeat = com1939.bjCOM1939Heartbeat;
                        printf("Tock.\n\r");
                    }// end if
                    */

            }// end if

            if(bExit == true)
                break;

        }// end while

    }// end if
    else if(nRetCode == 1)
        printf("Error opening binary file: %s\n\r", sCurrentPath);
    else if(nRetCode == 2)
        printf("Error in binary file (Checksum): %s\n\r", sCurrentPath);

    return 0;
```

```
}// end main
```

Note: The main.cpp portion delivered for the project will be extended to support testing the communication between the DVR and the J1939 vehicle network. The above code serves as an example.

To explain the crucial portions of the program:

```
// Declarations
static COM1939 com1939;
```

It is mandatory that the COM1939 class is static to assure proper function.

```
// Initialize the port and protocol settings
com1939.Initialize();
```

This call should be made only once during system initialization. After calling Initialize() make sure to check the COM port error information and react accordingly.

```
// Wait for assigned system loop time period
usleep(SYSTEM_LOOP_TIME);
```

In this case, the usleep() function is only used for test purposes. Under no circumstances should usleep() be used in the DVR program, since it does not provide a reliable time basis. Any other method of calling the library roughly every 100 milliseconds will work.

```
// Call the COM1939 protocol
com1939.Operate();
```

The Operate() function manages the communication with the J1939 converter and stores data and possible error messages.

The Diagnostics, Messages, and Queries Tables

The COM1939.h header file includes three tables, reflecting the .csv files as explained earlier. They are extended by parameters needed for program control.

Diagnostics Table

```
// Tables according to csv files read by NVR
struct com1939Diagnostics
{
    int nEnable;
    char sDescription[50];
    long lSubIndex;
    int nFMI;

     bool bErrorActive;              // To be set by NVR (true/false)
};
com1939Diagnostics pDiagnostics[DIAGNOSTICS];
```

This table represents the diagnostics.csv file, extended by the bErrorActive field.

The DVR program checks the error conditions as described in the corresponding .csv file, and sets the flag bErrorActive to true or false.

The COM1939 library will read the table, check the nEnable and the bErrorActive fields, and send out the diagnostics message to the J1939 vehicle network. When the bErrorActive field is true, the library will reset it to false.

Note: The COM1939 library supports up to 100 diagnostics entries. If the diagnostics table contains less than 100 entries, the table end must be marked with nEnable = 255.

Messages Table

This table represents the messages.csv file, extended by two fields, bDataReceived and nData.

```
struct com1939Messages
{
    int nEnable;
    long lIndex;
    long lSubIndex;
    int nByte;
    int nBit;
    int nBitLength;
    char sDescription[50];
    int nType;
    char sStat0[30];
    char sStat1[30];
    float fLowerRange;
    float fUpperRange;
    float fOffset;
    float fResolution;
    char sUnit[15];
    int nAddress;

    bool bDataReceived;     // Set to true by COM1939; must be reset by VNA upon reading
    char nData[50];         // Data to be processed by NVR according to variable nType
};
com1939Messages pMessages[MESSAGES];
```

This table represents the messages.csv file, extended by two fields, bDataReceived and nData.

The COM1939 library receives messages from the J1939 vehicle network, and enters the corresponding data into the nData field. At the same time, it will set the bDataReceived field to true. The DVR must reset the field to false upon reading the data.

The data is received as a string of 8-bit fields and its length is variable. The fields nByte, nBit, and nBitLength provide the information of the individual message data, where nBitLength may span over several bytes.

Note: Should the data information span over several bytes, the format is always LSB first, MSB last. See also the description of the corresponding .csv file for more information.

The COM1939 library supports up to 1000 message entries. If the messages table contains less than 1000 entries, the table end must be marked with nEnable = 255.

Queries Table

```
struct com1939Queries

{

  int nEnable; long
  lIndex;
  char sDescription[50];

  int nResponseLen;

  unsigned char sResponse[50];

};

com1939Queries pQueries[QUERIES];
```

The table represents the queries.csv file, extended by two fields, nResponseLen and sResponse.

The sResponse field is filled by the DVR according to the description of the .csv file. For example, Time and Date, GPS, etc. Since the response can be variable, the COM1939 must know the length of the response.

> **Note:** The COM1939 library supports up to 50 query entries. If the query table contains less than 50 entries, the table end must be marked with nEnable = 255.

Glossary

- **0x:** Prefix for signifying hexadecimal
- **DTC:** Diagnostic Trouble Code
- **DVR:** Digital Video Recorder.
- **ECU:** Electronic Control Unit also referred to as Controller Application in SAE J1939 (In this document same as DVR or NVR)
- **J1939:** Refers to the SAE J1939 Can Bus Network
- **OC:** Occurrence Count
- **NVR:** Network video recorder.
- **SA:** Source Address
- **Web UI:** Web UI