READING AND INTERPRETING DIAGNOSTIC DATA FROM VEHICLE OBDII SYSTEM

Peter Dzhelekarski, Dimiter Alexiev

Faculty of Electronic Engineering and Technologies, Technical University of Sofia,
8 Kliment Ohridski Str., 1000 Sofia, Bulgaria, phone: +359 2 965 2622, e-mail: pid@tu-sofia.bg

This paper presents reading and interpreting diagnostic data from vehicle OBDII (On-Board Diagnostics II) system using a PC-based diagnostic tester. The current work represents the data link layer and the application layer of a diagnostic tester project. The paper begins with a brief description of the diagnostic protocols. The ISO 9141-2 and ISO 14230 protocols have been chosen for implementation mainly because they share the same physical interface. The diagnostic software is developed using C++ language utilizing Qt© cross platform toolkit. The connection to physical layer, data link layer and application layer are represented by the DtConnection-, DiagMessage- and DiagService class hierarchies respectively. Various data items from OBDII system are read and interpreted, such as: VIN, DTCs, engine RPM, and so on. The GUI (Graphic User Interface) of the program is a simple window with text field, menus and buttons for user commands. The diagnostic tester is successfully verified in practice and results are provided.

Keywords: OBD, diagnostics, ISO 15031-5, KWP 2000 and ISO 9141-2.

1. INTRODUCTION

This paper represents the data link layer and the application layer of a PC-based diagnostic tester project. It uses the communication link (physical layer) given in [1].

The OBDII (On-Board Diagnostics II) system ensures correct operation of the vehicle's emission control system during its lifetime by monitoring components for deterioration and malfunction. The EOBD (European OBD) system is the European equivalent alternative to the OBDII system. In this paper, from this point forward, the OBD abbreviation refers to OBDII/EOBD. The output of the OBD system is malfunction indicator lamp (MIL), also known as "check engine" lamp. When fault is detected, MIL is illuminated and diagnostic trouble code (DTC) is stored. A freeze frame, containing diagnostic data taken from that moment, is also stored. A diagnostic tester (scan tool) is required to obtain and display the diagnostic information stored via serial diagnostic interface.

The information can be read using one of the following 4 legislated data link layer protocols:

1) SAE J1850 (ISO 11519-4) – class B communication interface in 2 alternatives: 10.4 kbit/s and 41.6 kbit/s. It is used for diagnostics and data sharing purposes. Maximal message length is 12 bytes, including in-frame response. Error detection mechanism: 8-bit CRC (cyclic redundancy check). Application: mainly GM and Ford. [9]

2) ISO 9141-2 – diagnostic protocol, compatible with UART/SCI. Data rate: 10.4 kbit/s; maximal message length: 11 bytes; error detection mechanism: 8-bit checksum. [3]

3) ISO 14230-4 (KWP 2000 – Keyword Protocol) – diagnostic protocol which uses the same physical interface as ISO 9141-2. Maximal message length: 260 bytes; error detection as ISO 9141-2. KWP 2000 and ISO 9141-2 are used mainly for diagnostics by most European and Asian manufacturers, also Chrysler and GM. [6]

4) ISO 15765-4 (CAN Diagnostics) – high speed diagnostic interface (500 kbit/s) using CAN protocol. It is used for diagnostics and data sharing purposes. Maximal message length is 4095 bytes (using segmentation specified in ISO 15765-2). Sophisticated error detection mechanisms are utilized. Application: will be mandatory for all vehicles after model year 2008. [2]

The second and the third options (ISO 9141-2 / KWP 2000) have been chosen for the current project mainly because these are the most common protocols in Europe, sharing common physical interface (K-Line) and also due to similarity of their communication protocols.

The application layer of the diagnostic protocols is specified in ISO 15031-5. It specifies the OBD emissions-related diagnostic services. A service is an information exchange initiated by client (external test equipment) in order to require diagnostic information from server (ECU – electronic control unit). [8]

2. Problem Statement

The main task of present work is to implement the data link layer and application layer of a PC-based OBD tester. The diagnostic tester conforms to the requirements in ISO 15031-4 and uses ISO 9141-2/ISO 14230-4 and ISO 15031-5 diagnostic protocols. The tester should provide the user a simple GUI (graphical user interface) with menu, buttons and text field for output and status information.

Objectives

1) Implementation of connection to physical layer;
2) Implementation of data link layer;
3) Implementation of application layer;
4) Implementation of GUI;
5) Verification in real conditions.

3. Implementation

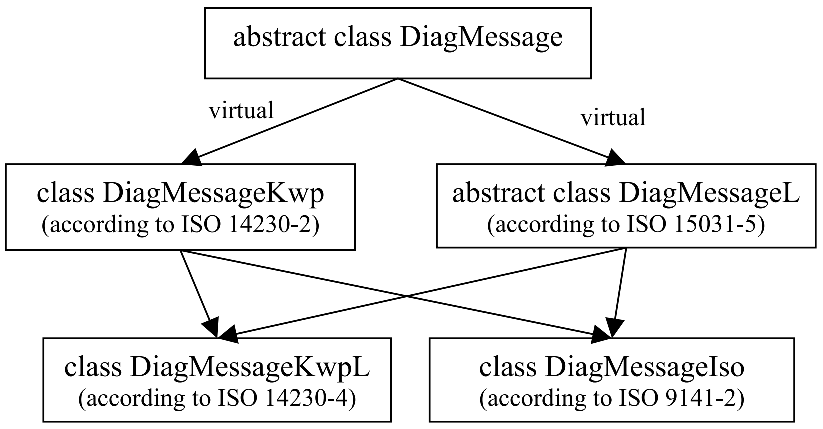
The diagnostic software is developed using C++ object oriented language. It is built using the Trolltech Qt© 3.3 multiplatform GUI toolkit. Some of the advantages of Qt software library are its portability, state-of-the-art GUI, object-oriented library and extensibility.

The interface adapter between OBD system and PC is described in [1]. The physical layer software performs non-overlapped serial port access, providing methods for initialization of communication link and data transfer. It is compiled as a static library (*RS232_WIN.lib*) for Win32 platform which is used by the present diagnostic software.

The connection to physical layer is performed by *class DtConnection*. It initializes diagnostic connection between the client (PC-based tester) and the server (vehicle) and maintains it by periodical sending of request messages to the server. It also provides a high-level interface for sending/receiving (transferring) diagnostic messages. For every data transfer a special program thread (*DtTransfer*) is started, which launches the serial port methods in another thread and waits for its completion. This mechanism is required because the port is used in non-overlapped access and all I/O operations are blocking. Thus the main program is able to respond to user interactions and to perform other operations while data transfer is in progress.

4. DATA LINK LAYER

The data link layer is specific to the diagnostic protocol and is specified in ISO 9141-2 and ISO 14230-2. There are similarities between these two protocols that simplify the implementation. The data link layer is represented by *DiagMessage* class hierarchy (see Fig. 1). Objects are created from *DiagMessageKwpL* and *DiagMessageIso* classes for KWP 2000 and ISO 9141-2 protocols respectively.


abstract class DiagMessage
virtual
virtual
class DiagMessageKwp
(according to ISO 14230-2)
abstract class DiagMessageL
(according to ISO 15031-5)
class DiagMessageKwpL
(according to ISO 14230-4)
class DiagMessageIso
(according to ISO 9141-2)

DiagMessage – base type for all diagnostic messages

DiagMessageL – legislated base type message

DiagMessageKwp – KWP 2000 message (up to 260 bytes, manufacturer specific)

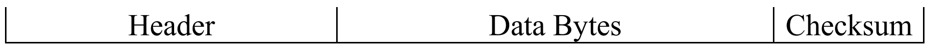
DiagMessageKwp – KWP 2000 legislated (up to 11 bytes)

→ = inherits (as public)

Fig. 1. DiagMessage Class Hierarchy

The data link layer (from the clients side) handles diagnostic message structure and bus errors in 2 modes: request and response modes. In response mode messages are received as a sequence of bytes, error checking is performed and data bytes, which carry diagnostic information, are output to the application layer. In request mode the data bytes are packed into the diagnostic messages to be sent.

Diagnostic messages consist of 3 main parts: header, data bytes and checksum (see Fig. 2). The data bytes length is restricted to 7 bytes for ISO 9141-2 and for legislated KWP 2000 messages. The header of ISO 9141-2 message has fixed form with the following bytes: *Fmt* ($68 for request, $48 for response); *Tgt* ($6A for request, $*6B* for response) and *Src*. The header of KWP 2000 message depends on received keywords. [3, 4, 8]



Fmt = Format byte;
Tgt = Target address;
Src = Source address;
Len = Data bytes length.

Fmt	Tgt*	Src*	Len*	… Data …	CS
1…4 bytes				1…255 bytes (up to 7 for legislated)	1 byte

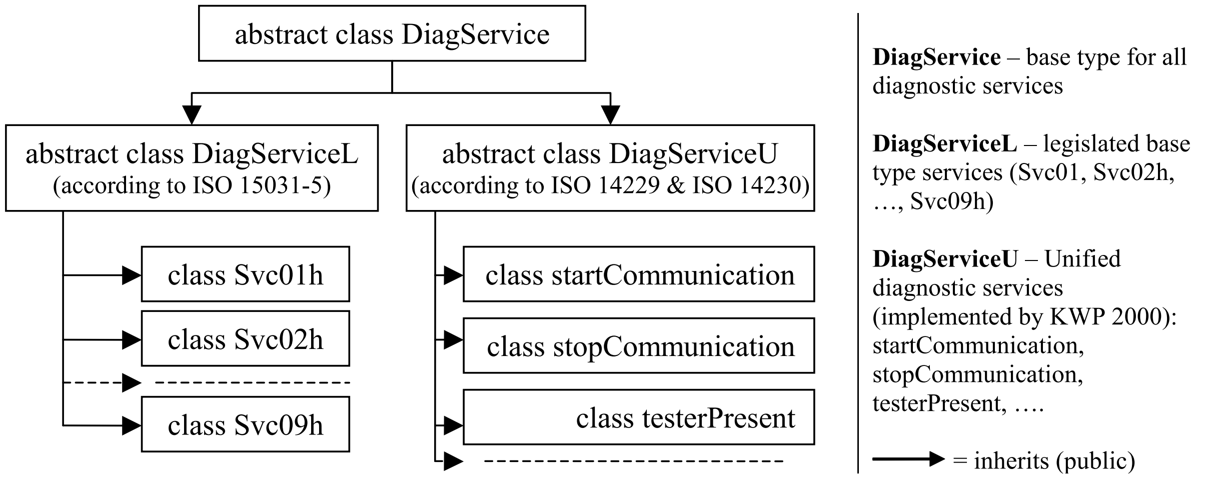
*optional bytes depending on Fmt

Fig. 2. Diagnostic Message Structure

5. Application Layer

The application layer is represented by *DiagService* class hierarchy (see Fig. 3). Its main task is to interpret the diagnostic data received.

The general format of diagnostic services is specified in ISO 14229 (Unified Diagnostic Services). The implementation of unified services in KWP 2000 is specified in ISO 14230-2,3. The legislated (OBD emissions-related) services are specified in ISO 15031-5.


abstract class DiagService
abstract class DiagServiceL
(according to ISO 15031-5)
abstract class DiagServiceU
(according to ISO 14229 & ISO 14230)
class Svc01h
class Svc02h
class Svc09h
class startCommunication
class stopCommunication
class testerPresent

DiagService – base type for all diagnostic services

DiagServiceL – legislated base type services (Svc01, Svc02h, …, Svc09h)

DiagServiceU – Unified diagnostic services (implemented by KWP 2000): startCommunication, stopCommunication, testerPresent, ….

$\longrightarrow$ = inherits (public)

Fig. 3. DiagService Class Hierarchy

The diagnostic services are contained in the data bytes of the messages. Their format depends on the specific diagnostic service implemented. The first byte is always the service identifier (SID). For response messages the sixth bit of SID is always logic “1”. Services with SIDs from $01 to $09 ($41 to $49 for responses) are legislated (OBD emission-related). Some of them have parameter identifier (PID), test identifier (TID) or *InfoType*, following the SID byte. Additional data bytes (up to 6 bytes) can be included depending on the service.

5.1 Implemented Diagnostic Services

The following unified diagnostic services are implemented (for KWP 2000):

- *startCommunication* (SID $81): initializing (starting) communication;
- *stopCommunication* (SID $82): termination of communication;
- *testerPresent* (SID $3E): preventing automatic disconnection.

The following OBD emissions-related diagnostic services are implemented:

- *Service $01*: requesting real-time data values from system inputs and outputs, including number of stored DTCs and MIL status.
- *Service $02*: requesting data values in a freeze frame.
- *Service $03*: obtaining the DTCs stored in vehicle’s ECUs.

- *Service $04*: clearing diagnostic information, including: MIL status, DTCs, freeze frame, monitoring status, oxygen sensor data, etc.
- *Service $07*: obtaining the "pending" DTCs.
- *Service $09*: requesting vehicle identification number (VIN).

5.2 Data Interpretation

Diagnostic data is received as sequence of bytes (from 1 up to 6), contained in diagnostic services. According to the service the data is interpreted and converted into one of the following data types:

1) Floating point / integer data values: physical quantities with minimal and maximal values specified in the standard. Examples for floating point data: oxygen sensor(s) voltage [V], calculated load [%], ignition timing advance [°], air flow rate [g/s], etc. Examples for integer data: engine coolant temperature [°C], fuel gauge pressure [kPa], vehicle speed [km/h], engine RPM [min^{-1}], number of DTCs, etc.

2) Boolean data: ON/OFF devices such as MIL, PTO (power take-off), etc.

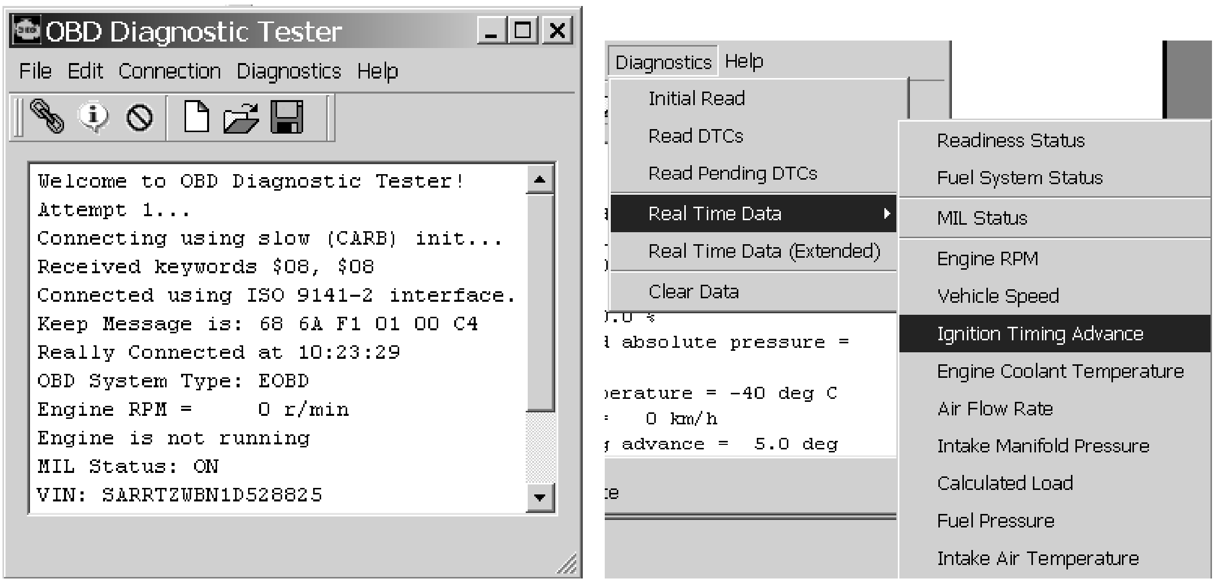
3) String data: VIN (17 ASCII characters), DTC (each code is specified with the letter P, B, C or U followed by 4 numbers, e.g. P0143), etc. For each DTC a description is looked up in and read from a dedicated text file, containing descriptions of all DTCs controlled by ISO/SAE.

4) Bit-encoded data: represents supported PIDs (TIDs, InfoTypes), location of oxygen sensors and OBD monitors readiness status.

The data which is read and interpreted is stored in an object of *DtVehicle* class, which contains data for the entire vehicle OBD system. *DtVehicle* has a list of objects of *DtEcu* classes, which contain the data received from the different ECUs. *DtSession* class controls the sequence of reading and interpreting diagnostic data.

6. GUI

The GUI of the software consists of a main window with a text field for diagnostic reports, menus/buttons for user commands and status bar. Fig. 4 shows the condition of the main window after establishing of connection and some of the available items in menu "Diagnostics". The corresponding data item, requested from user, is printed on the text field. The program can create a log file containing all request / response messages and a list of diagnostic data with text description.



```
Welcome to OBD Diagnostic Tester!
Attempt 1...
Connecting using slow (CARB) init...
Received keywords $08, $08
Connected using ISO 9141-2 interface.
Keep Message is: 68 6A F1 01 00 C4
Really Connected at 10:23:29
OBD System Type: EOBD
Engine RPM =     0 r/min
Engine is not running
MIL Status: ON
VIN: SARRTZWBN1D528825
```

```
l absolute pressure =
perature = -40 deg C
=   0 km/h
g advance =  5.0 deg
```

Fig. 4. Snapshots of OBD Diagnostic Tester Software

7. Results

The OBD diagnostic tester has been successfully verified on ISO 9141-2 protocol, utilizing an ECU from Rover 25-LHD, model 2003. The results from diagnostic session are shown in Table 1, which contains some request and response messages for supported PIDs, OBD system type and engine RPM (see also Fig. 2 and [8]).

Table 1. Some Request and Response Messages for Rover 25-LHD

No.	Requested Data Type	Header			Data Bytes			CS
		Fmt	Tgt	Src	SID	PID	Data …	
1.	Supported PIDs / keeping connection alive:							
	Request	68[1)]	6A	F1[2)]	01	00	–	C4
	Response	48	6B	13[3)]	41	00	BE 3E B8 10 [4)]	CB
2.	OBD system type:							
	Request	68	6A	F1	01	1C	–	E0
	Response	48	6B	13	41	1C	06[5)]	29
3.	Engine RPM:							
	Request	68	6A	F1	01	0C	–	D0
	Response	48	6B	13	41	4C	00 00[6)]	13

[1)] All values in the table are HEX numbers
[2)] Request source address $F1 is the default tester address
[3)] Response address $13 means "Powertrain controllers – Engine controller" [4]
[4)] Supported PIDs $01…$21, bit encoded in 4 bytes (1 = supported / 0 = not supported)
[5)] OBD system type $06 means "EOBD"
[6)] Engine RPM is 0 min^{-1}, calculated as follows: RPM = (byte1 + byte2 * 256) / 4

8. Conclusion

This paper describes the process of reading and interpreting diagnostic data from OBD system using a PC-based tester. The tester supports the ISO 9141-2 and Keyword Protocol 2000 (ISO 14230) which are the most common diagnostic protocols in Europe. The software is written in C++ and provides a graphical user interface for displaying diagnostic data items. The tester has been successfully verified in practice and the results are provided.

9. References

[1] Dzhelekarski, P. and D. Alexiev. *Initializing communication to vehicle OBDII system*. Submitted for publication at Fourteenth Int. Conference ELECTRONICS'05, 2005.

[2] Dzhelekarski, P., V. Zerbe and D. Alexiev. *FPGA implementation of bit timing logic of CAN controller*. IEEE Proceedings, 27th Int'l Spring Seminar on Electronics Technology, 2004.

[3] *ISO 9141-2. Road vehicles – Diagnostic systems – Part 2: CARB requirements for interchange of digital information*. ISO, 1994.

[4] *ISO 14230-2. Road vehicles – Diagnostic systems – Keyword Protocol 2000 – Part 2: Data link layer*. ISO, 1999.

[5] *ISO 14230-3. KWP 2000 – Part 3: Application Layer*. ISO, 1999.

[6] *ISO 14230-4. KWP 2000 – Part 4: Requirements for emission-related systems*. ISO, 2000.

[7] *ISO/DIS 15031-4. Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 4: External Test Equipment*. ISO, 2004.

[8] *ISO/DIS 15031-5. Road vehicles – Communication between vehicle and external equipment for emissions-related diagnostics – Part 5: Emission-related diagnostic services*. ISO, 2004.

[9] Oliver, J. *Implementing the J1850 protocol*. Intel Corporation, 1997.