The OpenOtto Project

$Date: 2004/07/08 02:34:14 $

Abstract

An open implementation of automotive communication and diagnostic protocols.

The goal of the OpenOtto Project is to provide complete free and open access to the networked electronic devices in an automobile. The interface to vehicle devices will be primarily through the standard diagnostic connector, though communication will be supported through all busses, not only diagnostic busses. Access to vehicle devices will include monitoring and diagnostics as well as reprogramming and enhanced control of their operation.[1]

Contents

[1] It is important to be aware that there are applications that may interfere with the operation of certain safety related devices in an automobile, e.g., braking systems and airbags.

List of Tables

Copyright

`$Id: architecture.lyx,v 1.12 2004/07/08 02:34:14 alpha Exp $`

Foreword

- todo
  - other things in forward?
    - history
    - acknowledgements

1 Hardware

Interface between the computer and the automotive physical layer. Certain devices are restricted to certain data link layers and/or certain subsets of protocols, messages, and addresses.

Hardware data link devices and software data link devices are similar; the major difference is that in hardware data link devices the software that decodes the data link is embedded in the device and generally referred to as firmware. In all devices except for the Programmable Logic based devices, the data link decode is performed in software on a microprocessor.

1.1 Physical Layer Requirements

Provide interface between CMOS/TTL level signals and bus level signals. Where applicable based on the bus type, provide termination, differential signalling, and half duplex interface.

1.1.1 Pulse Width Modulation (PWM)

The Pulse Width Modulation (PWM) bus type is specified in [SAE J1850].

1.1.2 Variable Pulse Width Modulation (VPW)

The Pulse Width Modulation (PWM) bus type is specified in [SAE J1850].

Bit Encoding	pulse width modulation
Drive Type	differential voltage
Data rate	41.6 kbps
Minimum Pulse Width	$6\mu s$
Media	dual wire
Output Low Voltage	min 0 V, max 1.2 V
Output High Voltage	min 3.8 V, max 5.25 V

Table 1: Pulse Width Modulation Physical Layer

Bit Encoding	variable pulse width modulation
Drive Type	voltage
Data Rate	10.4 kbps
Minimum Pulse Width	$34\mu s$
Media	single wire
Output Low Voltage	min 0 V, max 1.5 V
Output High Voltage	min 6.25 V, max 8 V

Table 2: Variable Pulse Width Modulation Physical Layer

1.1.3 ISO

The ISO bus type is specified in [ISO 9141-1]. This physical layer is the same as that specified by [ISO 14230-1] for vehicles with a 12 V electrical system.

The ISO bus type employs a similar signalling scheme to a serial port as specified in [TIA-232]. With appropriate voltage level conversion, a serial UART may be used to communicate on an ISO bus.

1.1.4 Controller Area Network (CAN)

The Controller Area Network (CAN) bus type is specified in [SAE J2284].

Bit Encoding	voltage level
Drive Type	voltage
Data Rate	10.4 kbps
Minimum Pulse Width	$67\mu s$
Media	single wire
Output Low Voltage	min 0 V, max 2.4 V
Output High Voltage	min 9.6 V, max 12 V

Table 3: ISO Physical Layer

Bit Encoding	voltage level
Drive Type	differential voltage
Data Rate	500 kbps
Minimum Pulse Width	1990 ns
Media	dual wire
Output Low Voltage	1.5 V
Output High Voltage	3.5 V

Table 4: Controller Area Network Physical Layer

1.2 Software Data Link Devices

The software data link devices provide little to no additional circuitry beyond logic conversion. All data link decode is done in software. In these devices, the physical layer interface is optimized so that signal conversion is directly between the automotive bus and the computer bus, instead of converting to and from CMOS/TTL signalling.

The requirements for the computer interface of a software data link device are one bit of input, one bit of output, and a double edge sensitive interrupt on the input. Processor clock speed, interrupt latency, and I/O latency may affect the maximum possible data rates, and may make some or all of the network types impossible to implement with a particular interface. For slower busses, polling with a level sensitive or single edge sensitive interrupt or no interrupt may be adequate on faster processors.

1.2.1 Serial

This device interfaces with a serial port as specified by [TIA-232]. Only the ISO bus type is compatible with the UART of the serial port. For the VPW, PWM, and CAN bus types, pulses are measured and generated via software. Since the PWM and CAN buses communicate at faster data rates, some hardware may not be able to provide the real time response necessary to support either or both higher speed busses.

todo: CAN issues: is CAN feasible at all? CAN Tx vs. ISO L-line Tx, which is more useful?

todo: Each bus connects

1.2.2 Parallel

This device interfaces with a parallel port as specified by [IEEE 1284]. This device supports the VPW, PWM, and ISO bus types. Since only one pin on the parallel port triggers an interrupt, and then only on the rising edge, additional hardware as well as software overhead is necessary to accommodate interfaces on this port.

DB-9 pin	serial function	direction	IRQ	function
1	DCD	in	delta	CAN Rx
2	RxD	in	no	ISO K-line Rx
3	TxD	out	no	ISO K-line Tx
4	DTR	out	no	ISO L-line Tx
5	GND		no	signal ground
6	DSR	in	delta	ISO L-line Rx
7	RTS	out	no	VPW/PWM Tx
8	CTS	in	delta	VPW/PWM Rx
9	RI	in	falling edge	

Table 5: Serial Hardware Interface Pin Assignments

DB-9 pin	serial function	direction	IRQ	function	VPW/PWM	ISO	CAN
1	DCD	in	delta				
2	RxD	in	no	UART Rx		K-line Rx	
3	TxD	out	no	UART Tx		K-line Tx	
4	DTR	out	no	software Tx	Tx	L-line Tx	Tx
5	GND		no	signal ground			
6	DSR	in	delta	software Rx	Rx	L-line Rx	Rx
7	RTS	out	no				
8	CTS	in	delta				
9	RI	in	falling edge				

Table 6: Serial Hardware Interface Pin Assignments

DB-25 pin	parallel port function	direction	IRQ	function
1	/STROBE	out		
2	D0	out		ISO K-line Tx
3	D1	out		ISO L-line Tx
4	D2	out		VPW/PWM Tx
5	D3	out		CAN Tx
6	D4	out		
7	D5	out		
8	D6	out		
9	D7	out		
10	/ACK	in	positive edge	Rx wired OR
11	BUSY	in		ISO K-line Rx
12	PE	in		ISO L-line Rx
13	SELIN	in		VPW/PWM Rx
14	/AUTOFD	out		
15	/ERROR	in		CAN Rx
16	/INIT	out		
17	/SEL	out		
18-25	GND			signal ground

Table 7: Parallel Hardware Interface Pin Assignments

1.2.3 GPIO

The GPIO data link device uses general purpose IO (GPIO) pins on a microprocessor or peripheral controller. A different kernel driver will be required for each device due to variations in the specific hardware used.

1.3 Hardware Data Link Devices

These devices utilize more hardware to decode the data link protocol than software data link devices. These devices have more complex hardware, are generally more expensive, but require less host system resources to operate.

1.3.1 Microcontroller

General Purpose Microcontrollers These devices use a general purpose microcontroller to perform the real time functions of data link decode. Any general purpose microcontroller would be suitable, such as the 68HC11, 8051, or more modern variants such as the PIC or AVR. In addition to the microcontroller, additional circuitry is necessary to interface the bus physical layer.

Special Purpose Microcontrollers There are a few microcontrollers designed specifically for automotive diagnostic interface applications. These device

are generally restricted to network protocols necessary for implementing a scantool. Devices may contain some or all of the physical layer interface circuitry. These devices include the Elm Electronics ELM3xx and the Oezen Elektronik Mobydic.

Repurposed Consumer Hardware This device leverages the programmable nature of an off-the-shelf USB to serial converter. As with the general purpose microcontroller designs, additional circuitry is necessary to interface to the bus physical layer. Custom firmware is loaded from the host machine to the USB adapter to perform decode of the data link protocol.

1.3.2 Programmable Hardware

The interface based on programmable hardware, such as an FPGA or PLD, performs data link decode with programmable logic. Additional circuitry needed is the physical interface to the host and the automobile. A programmable hardware device can potentially yield better performance, which matter most when supporting more than one bus in a single device, and when supporting high data rate busses.

1.3.3 Development Board

This interface is based on an embedded PC. This device is much like a software data link device, in that a general purpose computer performs data link decode. However, that computer does not perform other functions such as managing a user interface, but instead communicates over a network to another computer. The firmware for such devices may be the same as a software data link device, i.e., a Linux kernel device driver, or it may run a more specialized driver under a simpler operating system such as eCos or DOS.

2 Software

2.1 Device Protocols

2.1.1 Software Data Link Device Kernel Interface

The kernel device driver presents a character device with raw access to the capabilities of the device. Data is sent to and from the device file and control information is passed via ioctls.

Data The data is composed of 32 bit packets. Each packet specifies one pulse. The format of each data packet is described in table 8.

Bus Configuration The following ioctls are used to specify the busses to enable:

bit	function
31-28	bus number
25-27	unused, reserved for future expansion
24	pulse polarity
23-0	pulse length in nanoseconds ($1ns - 16.8ms \pm 1ns$)

Table 8: Data Packet Format

Name	Value	Description
IFS	0	Deasserted bus timeout, also known as interframe separation.
BREAK	1	Asserted bus timeout, also known as break.

Table 9: Pulse Configuration IDs

- set_busenable(int busmask)
- get_busenable(int *busmask)

The bus interfaces to enable is specified as a bitmask, a set bit enables the specified bus. Busses that are not enabled will not receive or send data.

Pulse Configuration The following ioctls are used to configuration certain pulses:

- set_pulse(int id, int pulse)
- get_pulse(int id, int *pulse)

The interframe separation (IFS) time is the minimum pulse required between packets. Any pulse longer than this is assumed to be equivalent to, and is reported as, this length. The break signal is the maximum asserted pulse width, used to interrupt a transfer. The interframe separation (IFS) time is the minimum pulse required between two packets, also called the bus idle time. The odd IDs specify pulses for an asserted bus and the even IDs specify pulses for a deasserted bus. Therefore, the logical polarity of the pulse is specified by the lowest bit. Table 9 lists the defined pulse IDs.

The time parameter is specified in the same format as the data packets. The pulse polarity specifies the physical polarity of the signal, to allow for electrically inverted busses and/or interfaces. The bus number field of the pulse is ignored.

2.1.2 OBD Over Stream Protocol

The OBD over stream protocol is used to encapsulate packetized OBD data over a byte stream. It is used to tunnel automotive busses over serial lines and sockets.

byte sequence	meaning
END	End of packet.
ESC ESC_END	END data in packet.
ESC ESC_ESC	ESC data in packet.

Table 10: OBD Over Stream Control Characters

Table 10 summarizes byte sequences in the stream that have special meaning. Where applicable, these byte sequences and their numeric values are shared with similar implementations (e.g., SLIP and KISS).

For busses that specify an initialization procedure that does not utilize the usual bus symbols, this procedure will be performed automatically the first time the socket is opened. If the initialization procedure can determine whether or not a bus is present, this information is available through the control socket. Any finalization procedure is performed after all applications close the socket.

An additional socket exists for performing configuration and out of band control of bus interfaces. Control functions include the following:

- Query bus interface parameters,
- Change bus interface parameters,
- Transmit bus commands (such as initialization or finalization), and
- Retrieve results of previous commands.

The following parameters are available for each bus interface:

- Bus type (vpw, pwm, iso, can, or auto),
- Bus speed in bits per second or zero for default rate, and
- Initialization to perform on socket open (none, fast, slow, carb, or auto).

2.1.3 Network Interface

- todo:
  - sockopts
    - network protocol, speed, init
    - address
  - promiscuous mode for sniffing
  - support AF_PACKET

2.1.4 OBD Over IP Protocol

todo: OBD over other stuff, too?

todo: encapsulated packet format, including bus type, IFR, normalization bit

2.2 Drivers

2.2.1 Kernel Driver

A kernel driver is required to perform the interrupt and timing functions for each software data link device. There is a different kernel driver for each variant of the software data link device.

2.2.2 Driver Daemon (ottod)

The driver daemon provides a unified interface to all the various hardware interfaces. The daemon also provides some buffering and other higher level functions. For the software data link interfaces, data link decode is done in ottod. Multiple applications may connect to a single driver daemon. A single ottod may support more than one bus through one or more devices. The driver daemon communicates with physical devices through a kernel device driver, either a standard serial device for hardware data link devices or the custom device driver for software data link devices. Applications communicate with the driver daemon either through local AF_UNIX sockets or through a network interface of type AF_OBD.

Local socket communication is specified in section 2.1.2. Data is passed through one socket per bus interface while control information is passed through another socket.

The network interface implementation is specified in section 2.1.3. Data is passed through a socket of AF_OBD type, while control information is passed via getsockopt() and setsockopt().

The driver daemon is configured at runtime to associate bus interfaces with local sockets or network interfaces. Configuration of the driver daemon includes the following for each bus interface:

- Device file to physical device,
- Device type,
- Device specific options,
- Bus number within device,
- Allowed bus types,
- Default bus type and speed, and
- Default initialization type.

2.3 Libraries

The library functionality is minimally split up to allow logical groups of functions to be separated from others without having an inordinate number of libraries to keep track of. Functions that are shared between different platforms are in their

own library since they will be built differently from other functions. Notably, they will require much broader cross-compilation support. Functions related to a single standard, governing body, manufacturer, or governmental mandate, will be grouped together in their own library. Only where a specific need arises will functions be split into libraries further than this.

2.3.1 Data Link Decode (libobd2-link)

The data link decode library contains functionality necessary to decode the data link layer on software data link devices. Data link layer is specified in [SAE J1850] for VPW and PWM and [ISO 14230-1] for ISO. Some devices may only support a subset of the data link layers if they cannot meet the timing requirements necessary for the faster bus types. The library can be configured at compile time as well as run time to omit the unsupported bus types. Compile time configuration is particularly necessary to reduce code size for hardware data link devices. The data link decode routines are shared, where possible, between all software and hardware data link decode platforms. These platforms include the host computer, the development board, general purpose microcontrollers, and the USB serial adapter.

Constants are provided to specify all the relevant pulses for each bus.

The library provides the following functions:

otto_datalinkdecode Decode a series of pulses into data bytes.

otto_datalinkencode Encode data bytes into a series of pulses.

2.3.2 Vehicle Identification Number (libvin)

The vehicle identification number (VIN) library contains routines to decode the VIN of a vehicle. For vehicles manufactured after 1980, the format of the VIN is specified in [ISO 3779, ISO 3780]. From the VIN it is possible to determine specifications of a vehicle including make (manufacturer), model, model year, trim level, country of origin, and other vehicle information. The library also contains a routine to verify the VIN checksum. Some information may not be available for all vehicles, depending on many factors including the willingness of the manufacturer to provide such information to the public. Each entry is specified as a key/value pair: the key is a pattern match expression on the VIN and the value is a description of some property of the vehicle. Multiple matches are allowed to prevent redundancy in the database.

The library will define the data type as specified in table 11 for representing a decoded VIN.

The library provides the following functions:

otto_vinchecksum Calculate the checksum for a VIN. This value may be compared to the appropriate position to verify a checksum or assigned to that position to create a valid VIN.

otto_vinlookup Lookup the specified VIN in

member name	description
country	Country of manufacture.
manufacturer	Manufacturer name (e.g., Ford).
make	Make or division (e.g., Land Rover).
model	Model name (e.g., Discovery).
series	Series or model variant (e.g., SE).
body	Body type (number of doors, etc.).
engine	Engine size and fuel type.
transmission	Transmission, weight class.
weight	Gross vehicle weight rating (GVWR).
emission	Emission system or rating.
restraint	Restraint system.
year	Model year.
plant	Assembly plant.
serial	Manufacture sequence number.

Table 11: Decoded VIN Data Type

2.3.3 Diagnostic Test Modes (libobd2)

The OBD2 library implements the majority of the SAE OBD2 specification. Support for each group of functionality is described below. See the referenced specifications for more detailed information about the features.

For certain data, such as DTC tables and physical addresses, the library contains SAE specified data as well as manufacturer specified data. Manufacturer data applicability is determined based on VIN pattern matching.

Network layer operation Support for the network layer packet as specified in [SAE J1850]. A data structures is defined to facilitate formatting of data into network packets for requests and decoding data from replies. The data structure contains the entire packet, including header and checksum.

The network layer support will provide the following functions:

otto_send Write a data packet on the specified bus.

otto_recv Read a data packet from the specified bus.

otto_chksum Compute the checksum of a packet.

Diagnostic test modes Support for the diagnostic test modes as specified in [SAE J1979]. Diagnostic test modes supported are as follows:

- Mode 0x01: Request current powertrain diagnostic data,
- Mode 0x02: Request powertrain freeze frame data,
- Mode 0x03: Request emission-related powertrain diagnostic trouble codes,

- Mode 0x04: Clear/reset emission-related diagnostic information,
- Mode 0x05: Request oxygen sensor monitoring test results,
- Mode 0x06: Request on-board monitoring test results for non-continuously monitored systems,
- Mode 0x07: Request on-board monitoring test results for continuously monitored systems,
- Mode 0x08: Request control of on-board system, test, or component, and
- Mode 0x09: Request vehicle information.

For each test mode, data structures are defined to facilitate formatting of data into network packets for requests and decoding data from replies. Data structures shall be defined based on the formats defined in [SAE J1979]. All modes shall have a request format and a reply format defined. Modes 0x01, 0x02, 0x05, and 0x06 have a additional reply format defined for messages that request the mode functions (PIDs, test IDs) supported.

Tables of constants will be also be defined based on [SAE J1979]. The following tables shall be defined:

- PIDs for modes 0x01 and 0x02,
- bitmapped data for PIDs 0x01, 0x03, 0x12, 0x13, 0x1e in modes 0x01 and 0x02,
- constants for PID 0x1c in modes 0x01 and 0x02,
- test IDs for mode 0x05,
- minimum, maximum, and scaling values for tests in mode 0x05 (in unified SLOT definitions?),
- test IDs for mode 0x06,
- test IDs for mode 0x08, and
- information type IDs for mode 0x09.

Diagnostic trouble codes shall be handled separately, via additional library functions, so that they may customized to each vehicle.

Enhanced diagnostic test modes Support for enhanced diagnostic test modes as specified in [SAE J2190]. Enhanced diagnostic test modes support are the following:

- Mode 0x10: Initiate diagnostic operation,
- Mode 0x11: Request module reset,

- Mode 0x12: Request diagnostic freeze frame data,
- Mode 0x13: Request diagnostic trouble code information,
- Mode 0x14: Clear diagnostic information,
- Mode 0x17: Request status of diagnostic trouble codes,
- Mode 0x18: Request diagnostic trouble codes by status,
- Mode 0x20: Return to normal operation,
- Mode 0x21: Request diagnostic data by offset,
- Mode 0x22: Request diagnostic data by parameter identification,
- Mode 0x23: Request diagnostic data by memory address,
- Mode 0x24: Request scaling and offset or PID,
- Mode 0x25: Stop transmitting requested data,
- Mode 0x26: Specify data rates,
- Mode 0x27: Security access mode,
- Mode 0x28: Disable normal message transmission,
- Mode 0x29: Enable normal message transmission,
- Mode 0x2A: Request diagnostic data packet(s),
- Mode 0x2B: Dynamically define data packet by single byte offsets,
- Mode 0x2C: Dynamically define diagnostic data packet,
- Mode 0x2F: Input/output control by PID,
- Mode 0x30: Input/output control by data value ID,
- Mode 0x31: Perform diagnostic routine by test number – start routine,
- Mode 0x32: Perform diagnostic routine by test number – stop routine,
- Mode 0x33: Perform diagnostic routine by test number – request routine results,
- Mode 0x34: Data transfer – download (tool to module),
- Mode 0x35: Data transfer – upload (module to tool),
- Mode 0x36: Data transfer – transfer,
- Mode 0x37: Data transfer – exit,

- Mode 0x38: Perform diagnostic routine at a specified address – enter routine,
- Mode 0x39: Perform diagnostic routine at a specified address – exit routine,
- Mode 0x3A: Perform diagnostic routine at a specified address – request routine results,
- Mode 0x3B: Write data block,
- Mode 0x3C: Test device present, and
- Mode 0x7F: General response message.

For each test mode, data structures are defined to facilitate formatting of data into network packets for requests and decoding data from replies. Data structures shall be defined based on the formats defined in [SAE J2190]. All modes except mode 0x3F and 0x7F shall have a request format and a reply format defined. Modes 0x13, 0x17, 0x18 have an additional reply format to return the number of DTCs stored as well as the actual DTCs. Mode 0x27 has an additionally reply format for optional additional reply data. Mode 0x3F has a single request format. Mode 0x7F has a single response format, the general response that may be used in response to any enhanced diagnostic test mode request.

Tables of constants will be also be defined based on [SAE J2190]. The following tables shall be defined:

- level of diagnostics for mode 0x10,
- bitmapped status data for mode 0x18,
- response repeat options for modes 0x21, 0x22, and 0x23,
- scaling bytes for mode 0x24 (defined in general SLOTs?),
- date rate constants for mode 0x26,
- dynamic packet definitions for mode 0x2C, and
- response codes for mode 0x7F.

Diagnostic trouble codes shall be handled separately, via additional library functions, so that they may customized to each vehicle.

Header and packet decode Support for header and packet decode as specified in [SAE J1850, SAE J2178-1]. Data structures are defined to facilitate formatting of data into network packets for requests and decoding data from replies. Data structures defined are the following:

- single byte header,

- one byte consolidated header, and
- three byte consolidated header.

Tables of constants are defined for the following information:

- header flags,
- message types,
- message operations,
- extended addressing types, and
- geographical address map.

Physical address lookup Physical addressing as specified in [SAE J2178-1]. Standard defined types will be supported as well as manufacturer specific addresses. Addresses will be maintained in an extensible database to allow a user to easily provide additional address information.

Data parameter assignment database Support data parameter definitions as specified in [SAE J2178-2]. Data structures and tables of constants are defined to facility decode the various data parameter assignments. Where applicable, manufacturer definitions are supported and defined separately from standard definitions. The following data structures are defined: PID bit mapping.

For ease of diagnostics, data is referenced according to the parameter reference number (PRN) structure. A table of PRN groupings is defined, as well as detailed PRN assignments for the following groups:

- [SAE J1979] compatible,
- engine,
- transmission,
- brakes/tires/wheels,
- steering,
- suspension,
- restraints,
- driver information,
- HVAC,
- convenience,

- security,
- electric vehicle energy transfer system,
- configuration codes, and
- miscellaneous.

Values are specified according to the scaling, limit, offset, and transfer function (SLOT) definitions. This provides the meaning for any particular data. The following SLOT definitions are used:

- packeted (PKT),
- bitmapped without mask (BMP),
- unsigned numeric (UNM):
  - zero,
  - short (<8 bit),
  - 8 bit,
  - 16 bit,
  - 24 bit,
  - 32 bit,
- two's complement signed numeric (SNM), and
- state encoded (SED).

Single byte header messages Support for single byte messages as specified in [SAE J2178-3]. A table of constants shall be defined for frame IDs for all one byte headers, both single byte headers and consolidated three byte headers. A table of constants shall also be defined for the secondary IDs for electric vehicle energy transfer system (EV-ETS) messages.

Three byte header messages Support for three byte header messages as specified in [SAE J2178-4]. A table of constants shall be defined for primary IDs for functional addressing with three byte headers. Additionally, tables of secondary IDs shall be defined for the following message types:

- engine torque,
- engine air intake,
- throttle,
- air conditioning clutch,

- engine RPM,
- wheels,
- vehicle speed,
- traction control,
- brakes,
- steering,
- transmission,
- engine sensors – other,
- engine coolant,
- engine oil,
- engine systems – other,
- suspension,
- vehicle speed control,
- electric vehicle energy transfer system,
- charging system,
- electrical energy management,
- odometer,
- fuel system,
- ignition switch/starter,
- tell tales,
- climate control (HVAC),
- window wiper/washer,
- mirrors,
- door locks,
- external access,
- seat motion/control,
- windows,
- steering column,

- seat switches,
- restraints,
- exterior lamps outage,
- exterior lamps,
- interior lamps outage,
- interior lamps,
- tires,
- defrost,
- displays,
- exterior environment,
- interior environment,
- time/date,
- vehicle identification, and
- network control.

Tables will be defined for extended addresses for each of the following function:

- brakes, tires, and wheels,
- HVAC zones,
- window wiper/washer, defrost, photocell,
- doors and door locks,
- seats and restraints,
- windows,
- external lamps, and
- internal lamps.

Diagnostic trouble code databases Support for a diagnostic trouble codes (DTC) as specified in [SAE J2012]. A database shall be defined to store and manage DTCs as defined by standards as well as by manufacturer. The databases will be in a form that can be updated easily at runtime by the user. The database schema for storing DTCs is specified in table .

The database access functions will provide system group information when an unknown DTC is requested. The database will provide the following functions:

otto_dtclookup Returns a diagnostic message from looking up the specified DTC in the database.

member	description
dtc	DTC specified in all numeric format (P0000 is 0000).
message	Description.

Table 12: DTC Database Schema

2.3.4 non-obd protocols

Non-OBD protocols will be supported as a single library per protocol. Initially no such protocols will be supported.

2.3.5 High Level Application Functions (libotto)

todo: much work needed here, define this functionality
todo: compute horsepower
todo: fuel economy calc

functions todo

2.4 Applications

2.4.1 Interface Configuration (ottoconfig)

The interface configuration utility provides a convenient method to query and modify the settings for a bus interface. All of the settings available at the interface level, including network type and speed and initialization type, are available.

2.4.2 Network Monitor (ottodump)

The network monitor provides the ability to analyze and log network traffic. The application will connect to a single bus and dump network traffic. This program shall decode and display network traffic in a compact textual representation, log network traffic to a file in a form that can be read back later, read network traffic from a file saved previously, and filter traffic displayed or logged based on a user supplied expression.

The decoded packet data will include, where applicable, the following information:

- Network protocol used,
- Header information as specified in [SAE J2178-1],
- Data parameter adjusted value, as specified in [SAE J2178-2], and
- Frame ID meaning as specified in [SAE J2178-3, SAE J2178-4].

2.4.3 Network Scanner (ottoprobe? ottomap? ottoscan?)

The network scanner tool provides the ability to run scans of the automotive network to determine what devices are available. The scanner will attempt to determine all the modules present in the vehicle, and identify each module as much as possible.

2.4.4 Network Exploration Tool (ottocat)

The network exploration tool provides the ability to read and write arbitrary data on the automotive network. Data received on standard input is passed to the network and data from the network is relayed back on standard output. Data is escaped according to the OBD over stream protocol specified in section 2.1.2 to preserve packet boundaries.

2.4.5 Scan Tool (scantool/xscantool)

Scan Tool Functionality The scan tool provides scan tool functionality as specified by [SAE J1978]. Accordingly, communication between the scan tool and the vehicle will support:

- Automatic determination of busses present in the vehicle from the hardware interface present,
- Completion and support of on-board system readiness tests,
- Malfunction indicator light status and, if applicable, reason or reasons for illumination,
- Obtaining and displaying emissions related diagnostic trouble codes, and
- Obtaining and displaying emissions related current data, freeze frame data, and test parameters and results.

Where the scan tool is only specified to support a limited range of diagnostic procedures, the scan tool application will be expanded to provide the full range of similar functionality. For example, all diagnostic trouble codes, and all freeze frame data and test parameters will be supported, not only those related to emissions control.

The user interface of the scan tool will support a line oriented text interface and a graphical interface mode.

Textual User Interface The textual user interface to the scan tool will be line oriented. Functionality that would be awkward with a text display, such as graphing and continuously updated results, will not be supported with this interface.

Graphical User Interface The graphical user interface to the scan tool will provide a simple, intuitive interface to the common scan tool functions. The scan tool window will provide icons for initiating commands scan tool commands. Results will be displayed alphanumerically or graphically in the same window. Displaying multiple results will be supported. The user will be able select whether or not results are continuously updated. The user may also initiate an update of a result value.

2.4.6 Otto Mann (ottomann)

The Otto Mann tool provides high level control and monitoring of vehicle parameters via the automotive bus. The application will provide both text and graphical interfaces.

Shell The application will contain a text based command line interface. This interface will allow the user to initiate commands to the application (and, depending on the command, requests on the automotive bus). This interface will also allow, where applicable, return of alphanumeric results of commands. The shell interface is an intrinsic part of the application; it is not optional as a plugin.

Windows The application will provide graphic windows for graphical access to similar functionality available through the shell. This functionality includes scantool commands and results as well as configuration of the application and configuration of plugins.

Analysis The application will provide augmented vehicle operation facilities. These facilities will be available as commands to the shell, and results will be available to graphical and alphanumeric result plugins. The facilities provided include:

- Estimated power calculation (dynamometer),
- Fuel economy calculation, including moving average fuel consumption and estimated distance for remaining fuel,
- Wheel speed and slip display, and
- ABS and ETC activity.

The application will provide a configurable facility to recommend proper vehicle operation procedures. These procedures will include:

- Headlights operation as compared to sunrise and set time and outside brightness,
- Door position as compared to vehicle speed,
- Door locks as compared to vehicle speed, and
- Wiper operation as compared to headlight operation.

Extended Commands The application will allow extended control over vehicles functions, including:

- Disable ABS/ETC functionality, and
- Manual control of ABS/ETC operation on a per wheel, per axle, or per side basis.

Plugins The application will support a plugin method whereby optional functionality can be implemented separate from the main application. The plugin architecture will allow a plugin to support one or more of the following facilities: command and result. Plugins will be allowed the ability to have more than one instance of the plugin active at one time. This will allow two or more different plugin configurations for a single plugin to be active at one time.

The following standard plugins will be implemented:

- X11 window, providing a themable graphic display of data,
- Audio out, providing an audible output of data as sound effects or text to speech,
- Bitmap output, providing the ability to write static graphical representation of data to a file,
- Fifo control, providing an interface similar to the shell interface on a local filesystem fifo, and
- External program, providing the ability to run an arbitrary program based on configurable events.

A OSI Model

Layer 1 Physical layer, defined in [SAE J1850] for VPW and PWM, in [ISO 9141-1, ISO 14230-1] for ISO, and in [SAE J2284] for CAN.

Layer 2 Data Link layer, defined in [SAE J1850] for VPW and PWM, in [ISO 9141-2, ISO 14230-2] for ISO, and in [SAE J2284] for CAN.

Layer 3 Network layer, defined in [SAE J2178-1, SAE J2178-2, SAE J2178-3, SAE J2178-4].

Layer 4 Transport layer, not defined by SAE or ISO.

Layer 5 Session layer, not defined by SAE or ISO.

Layer 6 Presentation layer, not defined by SAE or ISO.

Layer 7 Application layer, defined in [SAE J2178-1, SAE J2178-2, SAE J2178-3, SAE J2178-4] for VPW and PWM, and in [ISO 14230-3] for ISO.

B Interface Hardware Support

todo: reference all devices mentioned by freediag, on opendiag and other lists

B.1 Supported Devices

none yet!

B.2 Unsupported Devices

OpenOtto Devices Devices defined by the OpenOtto Project.

- **Serial** Supports VPW, PWM, ISO, and CAN? bus types.
- **Parallel** Supports VPW, PWM, ISO, and CAN? bus types.
- **GPIO** Support depends on processor performance.
- **Microcontroller** Support depends on microcontroller performance and firmware.
- **Programmable Hardware** Support depends on hardware performance and firmware.
- **Development Board** Supports VPW, PWM, ISO, and CAN? bus types.

Andy Whittaker Interface schematic that supports ISO bus type.

B. Roadman Complete device that supports VPW, PWM, and ISO bus types.

Jeff Noxon Interface schematic that supports ISO bus type.

ELM3xx Interface ICs that support VPW, PWM, and/or ISO bus types, depending on model.

Multiplex Engineering Complete devices that support VPW, PWM, and/or ISO bus types, depending on model.

Mobydic Interface IC that supports VPW, PWM, ISO, and CAN bus types. (todo: CAN support limited?)

Silicon Engines Complete device that supports ISO bus type.

C Protocol Support

The status of support for other protocols, including proprietary and obsolete protocols, is described below.

C.1 Supported

[ISO 3779, ISO 3780] VIN support is incomplete due to unavailability of spec.

[SAE J1850] VPW and PWM bus physical layer supported by hardware.

[SAE J1978] OBD2 scan tool implemented in

[SAE J1979, SAE J2190] OBD2 diagnostic test modes implemented in

[SAE J2012] SAE DTC definitions

[SAE J2178-1, SAE J2178-2, SAE J2178-3, SAE J2178-4] OBD2 network messages

[SAE J2186] Data link security

[SAE J2284] OBD2 CAN support

[ISO 9141-1, ISO 9141-2, ISO 9141-3] ISO bus format

[ISO 14230-1, ISO 14230-2, ISO 14230-3, ISO 14230-4] ISO bus format for 24V vehicles

C.2 Support in Development

everything still working

C.3 Unsupported

The following protocols are not (yet) supported. The status of each protocol is given below.

Standard

EOBD ISO 15031-5, Europe OBD?

OBDI On-Board Diagnostics I.

Proprietary

VAG VW/Audi Group

KWP82 No info.

KWP1281 VAG 1552?

KWP2000 VW version of [ISO 14230-1, ISO 14230-2, ISO 14230-3, ISO 14230-4]. Possibly the same?

GM General Motors

ALDL Assembly Line Data Link. GM vehicles 1982-1986?

GM ECU Not yet investigated.

Mercedes Benz

Gearbox Not yet investigated.

Nissan

Consult Not yet investigated.

References

[SAE J1850] SAE J1850 – Class B Data Communications Network Interface

[SAE J1962] SAE J1962 – Diagnostic Connector

[SAE J1978] SAE J1978 – OBD II Scan Tool

[SAE J1979] SAE J1979 – E/E Diagnostic Test Modes

[SAE J2012] SAE J2012 – Recommended Practice for Diagnostic Trouble Code Definitions

[SAE J2178-1] SAE J2178-1 – Class B Data Communication Network Messages – Detailed Header Formats and Physical Address Assignments

[SAE J2178-2] SAE J2178-2 – Class B Data Communication Network Messages – Data Parameter Definitions

[SAE J2178-3] SAE J2178-3 – Class B Data Communication Network Messages – Frame IDs for Single-Byte Forms of Headers

[SAE J2178-4] SAE J2178-4 – Class B Data Communication Network Messages – Message Definitions for Three Byte Headers

[SAE J2186] SAE J2186 – E/E Data Link Security

[SAE J2190] SAE J2190 – Enhanced E/E Diagnostic Test Modes

[SAE J2284] SAE J2284 – High-Speed CAN (HSC) for Vehicle Applications at 500 KBPS

[ISO 9141-1] ISO 9141-1 – Road Vehicles – Diagnostic Systems – Requirements for interchange of digital information

[ISO 9141-2] ISO 9141-2 – Road Vehicles – Diagnostic Systems – Part 2: CARB requirements for interchange of digital

[ISO 9141-3] ISO 9141-3 – Road Vehicles - Diagnostic Systems – Part 3: Verification of the communication between vehicle and OBD II scan tool

[ISO 14230-1] ISO 14230-1 – Road Vehicles – Diagnostic Systems – Part 1: Physical Layer

[ISO 14230-2] ISO 14230-2 – Road Vehicles – Diagnostic Systems – Part 2: Data Link Layer

[ISO 14230-3] ISO 14230-3 – Road Vehicles – Diagnostic Systems – Part 3: Application Layer

[ISO 14230-4] ISO 14230-4 – Road Vehicles – Diagnostic Systems – Part 4: Requirements for emission-related systems

[TIA-232] TIA-232 – Interface Between Data Terminal Equipment and Data Circuit-Terminating Equipment Employing Serial Binary Data Interchange

[IEEE 1284] IEEE 1284 – Standard Signaling Method for a Bidirectional Parallel Peripheral Interface for Personal Computers

[49CFR565] United States Code of Federal Regulations, Title 49, Volume 5, Part 565 – Vehicle Identification Number Requirements

[ISO 3779] ISO 3779 – Road Vehicles – Vehicle identification number (VIN) – Content and structure

[ISO 3780] ISO 3780 – Road vehicles – World manufacturer identifier (WMI) code