

SURFACE VEHICLE RECOMMENDED PRACTICE

J2534™-1

OCT2015

Issued 2002-02
Revised 2015-10

Superseding J2534-1 DEC2004

(R) Recommended Practice for Pass-Thru Vehicle Programming

RATIONALE

Concerns from the Vehicle OEMs, Regulators, and Application Programmers have prompted the J2534-1 document to be updated to address the following items:

- Ensure that all compliant devices can be validated
- Ensure that all compliant devices act the same
- Concerns/issues from Vehicle OEMs and Device manufacturers
- Add new functionality to meet the needs of Vehicle OEMs that are improving the reprogramming process

To address these items, the following changes have been made:

1. To ensure testability and consistency among compliant devices
 - APIs have been clarified to eliminate ambiguities
 - Error case handling has been added/extended
 - Optional Content has been eliminated
 - Prohibit pop-ups or other types of user interaction from within the DLL
 - Specify behavior when a device disconnect occurs
2. To address Vehicle OEMs and Device manufacturers concerns
 - Interleave ISO 15765 messages
 - Individual timing/control settings for each ISO 15765 logical communication channel
 - Support ISO 15765 full and half-duplex

SAE Technical Standards Board Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2015 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
http://www.sae.org

SAE WEB ADDRESS:

Full document details: SAE J2534_1_201510

SAE values your input. To provide feedback on this Technical Report, please visit http://www.sae.org/technical/standards/J2534/1_201510

Documents

Document Sharing Hub

3. To meet new OEM requirements

- The API now supports raw CAN frames and ISO 15765 messages on the same physical link
- A Pass-Thru Device discovery mechanism has been added
- New device details to specifically support wireless Pass-Thru Devices
- Provided the ability to send and receive larger messages
- Provide a framework that is extendable with future protocols

TABLE OF CONTENTS

1.	SCOPE	6
2.	REFERENCES	7
2.1	Applicable Documents	7
2.1.1	SAE Publications	7
2.1.2	ISO Documents	7
3.	DEFINITIONS	8
3.1	ACRONYMS	8
3.2	TERMS	9
3.2.1	Active Message	9
3.2.2	Can Frame	9
3.2.3	Consecutive Frames	9
3.2.4	Data Link Controller	9
3.2.5	First Frame	9
3.2.6	High Impedance	9
3.2.7	Logical Communication Channel	9
3.2.8	Network Address Information	9
3.2.9	Pass-Thru Device	9
3.2.10	Pass-Thru Interface	9
3.2.11	Physical Communication Channel	9
3.2.12	Physical Vehicle Bus	10
3.2.13	Queued Message	10
3.2.14	Raw CAN Frame	10
3.2.15	Registry	10
3.2.16	Segmented Message / Segmented Periodic Message	10
3.2.17	Segmented Transfer in Progress	10
3.2.18	Single Frame	10
3.2.19	Servicing a Transmit Queue	10
3.2.20	Terminate a Message	10
3.2.21	Transmitted Message	10
3.2.22	Unsegmented Message	10
4.	DOCUMENTATION CONVENTIONS	10
5.	PASS-THRU INTERFACE CONCEPT	11
5.1	PC Requirements	12
5.1.1	32-bit versus 64-bit Operating Systems	12
5.2	Application Software Requirements and Assumptions	12

6.	PASS-THRU INTERFACE REQUIREMENTS	13
6.1	Connection to the PC	13
6.1.1	Interface between PC and Pass-Thru Device	13
6.1.2	Device disconnect from PC	13
6.2	Pass-Thru Interface DLL	13
6.3	Configuration Application.....	13
6.4	Connection to the Vehicle.....	13
6.5	Communication Protocols.....	14
6.5.1	ISO 9141.....	14
6.5.2	ISO 14230 (KWP2000).....	15
6.5.3	SAE J1850 41.6 kbps PWM (Pulse Width Modulation).....	15
6.5.4	SAE J1850 10.4 kbps VPW (Variable Pulse Width).....	15
6.5.5	CAN	15
6.5.6	ISO 15765 (transport layer for CAN)	16
6.5.7	SAE J2610 (also known as CHRYSLER SCI).....	16
6.6	Simultaneous Communication on Multiple Protocols	16
6.7	Programmable Power Supply.....	17
6.8	Pin Usage	18
6.9	Communication Channels	19
6.10	Data Buffering.....	19
6.10.1	Message reception	19
6.10.2	Message Transmission.....	20
6.11	Error Handling.....	24
6.11.1	Device Not Connected.....	24
6.11.2	Receive Buffer Overflow	24
6.11.3	Message Termination	25
6.11.4	Network Errors.....	25
7.	WIN32 APPLICATION PROGRAMMING INTERFACE	29
7.1	API Functions - Overview	29
7.2	Message Structure.....	30
7.2.1	C / C++ Definition	30
7.2.2	Description of Structure Elements.....	31
7.2.3	Protocol Specific Details for the PASSTHRU_MSG Structure.....	33
7.2.4	Format Checks for Messages Passed to the API.....	39
7.2.5	Conventions for Returning Messages from the API	40
7.2.6	Indications.....	40
7.3	API Functions – Detailed Information	43
7.3.1	PassThruScanForDevices	43
7.3.2	PassThruGetNextDevice	44
7.3.3	PassThruOpen.....	47
7.3.4	PassThruClose	48
7.3.5	PassThruConnect.....	49
7.3.6	PassThruDisconnect	56
7.3.7	PassThruLogicalConnect	57
7.3.8	PassThruLogicalDisconnect	62
7.3.9	PassThruSelect	63
7.3.10	PassThruReadMsgs	65
7.3.11	PassThruQueueMsgs	68
7.3.12	PassThruStartPeriodicMsg	70
7.3.13	PassThruStopPeriodicMsg	73
7.3.14	PassThruStartMsgFilter	74
7.3.15	PassThruStopMsgFilter	80
7.3.16	PassThruSetProgrammingVoltage	81
7.3.17	PassThruReadVersion	83
7.3.18	PassThruGetLastError.....	84
7.3.19	PassThruIoctl.....	85
7.4	IOCTL Section	88
7.4.1	GET_CONFIG	88
7.4.2	SET_CONFIG.....	89

7.4.3	READ_PIN_VOLTAGE.....	93
7.4.4	READ_PROG_VOLTAGE.....	94
7.4.5	FIVE_BAUD_INIT.....	94
7.4.6	FAST_INIT.....	95
7.4.7	CLEAR_TX_QUEUE.....	96
7.4.8	CLEAR_RX_QUEUE.....	96
7.4.9	CLEAR_PERIODIC_MSGS.....	96
7.4.10	CLEAR_MSG_FILTERS.....	97
7.4.11	CLEAR_FUNCT_MSG_LOOKUP_TABLE.....	97
7.4.12	ADD_TO_FUNCT_MSG_LOOKUP_TABLE.....	98
7.4.13	DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE.....	98
7.4.14	BUS_ON.....	99
8.	DLL INSTALLATION AND REGISTRY.....	99
8.1	Naming of Files.....	99
8.2	WIN32 Registry.....	99
8.2.1	User Application Interaction with the Registry.....	101
8.2.2	Attaching to the DLL from an Application.....	103
9.	RESOURCE SECTION.....	104
9.1	Bit Definitions for <RxStatus>.....	104
9.2	Bit Definitions for <TxFlags>.....	104
9.3	Values for <DeviceAvailable>.....	104
9.4	Values for <DeviceDLLFWStatus>.....	105
9.5	Values for <DeviceConnectedMedia>.....	105
9.6	Values for <ProtocolID>.....	105
9.7	Values for <Flags>.....	106
9.8	Values for <Connector>.....	106
9.9	Values for ErrInd Indications Description Codes.....	106
9.10	Values for <SelectType>.....	106
9.11	Values for <FilterType>.....	107
9.12	Values for <Voltage>.....	107
9.13	Values for <IoctlID>.....	107
9.14	Values for GET_CONFIG / SET_CONFIG <Parameters>.....	108
9.15	Pre-defined Values for GET_CONFIG / SET_CONFIG Parameters.....	109
9.16	Return Values.....	109
9.17	Structure Definitions for SDEVICE.....	111
9.18	Structure Definitions for RESOURCE_STRUCT.....	112
9.19	Structure Definitions for ISO15765_CHANNEL_DESCRIPTOR.....	112
9.20	Structure Definitions for SCHANNELSET.....	112
9.21	Structure Definitions for SCONFIG_LIST.....	112
9.22	Structure Definitions for SCONFIG.....	113
9.23	Structure Definitions for SBYTE_ARRAY.....	113
10.	NOTES.....	113
10.1	Revision Indicator.....	113
APPENDIX A.....		114
Figure 1	Pass-thru vehicle programming overview.....	11
Figure 2	Simultaneous communications on multiple protocols.....	17
Figure 3	SAE J1962 pin usage.....	18
Figure 4	Minimum receive message buffer sizes.....	20
Figure 5	Minimum transmit message buffer sizes.....	20
Figure 6	Transmission yield point for logical communication channels.....	21
Figure 7	Overview of the transmission process.....	21
Figure 8	Communication channel confuration for round-robin message transmission example.....	22
Figure 9	Service order for round-robin message transmission example.....	23
Figure 10	Message interleaving on the physical CAN layer.....	24

Figure 11	Message termination boundary	25
Figure 12	Network errors	28
Figure 13	SAE J2534 API functions	30
Figure 14	RxStatus bit definitions	32
Figure 15	TxFlags bit definitions	33
Figure 16	Allowed message sizes per protocol	34
Figure 17	List of Indications	41
Figure 18	Protocol / Indication Matrix	42
Figure 19	List of description codes for the ErrInd Indication	43
Figure 20	Return values for PassThruScanForDevices	44
Figure 21	Valid values for <DeviceAvailable>	45
Figure 22	Valid values for <DeviceDLLFWStatus>	46
Figure 23	Valid Values for <DeviceConnectedMedia>	46
Figure 24	Return values for PassThruGetNextDevice	46
Figure 25	Return values for PassThruOpen	48
Figure 26	Return values for PassThruClose.....	49
Figure 27	Values for <ProtocolID>	51
Figure 28	Values for <Flags>	52
Figure 29	Values for <BaudRate>	53
Figure 30	Values for <Connector>	53
Figure 31	Protocol Signal Offset in ResourceListPtr	54
Figure 32	Protocol Pin Usage	55
Figure 33	Return Values for PassThruConnect.....	56
Figure 34	Return values for PassThruDisconnect.....	57
Figure 35	Values for <ProtocolID>	59
Figure 36	Values for <Flags>	59
Figure 37	Valid TxFlags for ISO 15765 channel descriptors	61
Figure 38	Return values for PassThruLogicalConnect.....	61
Figure 39	Return Values for PassThruLogicalDisconnect.....	63
Figure 40	Return Values for PassThruSelect	65
Figure 41	Non-Blocking Reads	66
Figure 42	Blocking Reads.....	66
Figure 43	Return values for PassThruReadMsgs.....	68
Figure 44	Return Values for PassThruQueueMsgs.....	70
Figure 45	Periodic Interjection with a Segmented Transfer in Progress	71
Figure 46	Return values for PassThruStartPeriodicMsg	73
Figure 47	Return values for PassThruStopPeriodicMsg	74
Figure 48	PASS and BLOCK filter logic for physical communication channels that support logical communication channels (e.g., CAN)	75
Figure 49	PASS and BLOCK filter logic for physical communication channels that do NOT support logical communication channels (e.g., J1850PWM, J1580VPW, ISO9141, ISO14230, J2610, etc.)	76
Figure 50	PASS and BLOCK filter actions.....	76
Figure 51	Valid TxFlags for PASS and BLOCK filters	77
Figure 52	FilterType Values.....	79
Figure 53	Return Values for PassThruStartMsgFilter.....	79
Figure 54	Return Values for PassThruStopMsgFilter	80
Figure 55	Values for <Voltage>	82
Figure 56	Return Values for PassThruSetProgrammingVoltage.....	83
Figure 57	Return values for PassThruReadVersion	84
Figure 58	Return Values for PassGetLastError	85
Figure 59	Return values for PassThruIoctl	87
Figure 60	IOCTL Details	88
Figure 61	GET_CONFIG Details	89
Figure 62	SET_CONFIG Details.....	90
Figure 63	CONFIG Details.....	90

Figure 64	READ_VBATT Details	94
Figure 65	READ_PROG_VOLTAGE Details	94
Figure 66	FIVE_BAUD_INIT Details	95
Figure 67	FAST_INIT Details	95
Figure 68	CLEAR_TX_QUEUE Details	96
Figure 69	CLEAR_RX_QUEUE Details	96
Figure 70	CLEAR_PERIODIC_MSGS Details	97
Figure 71	CLEAR_MSG_FILTERS Details	97
Figure 72	CLEAR_FUNCT_MSG_LOOKUP_TABLE Details	97
Figure 73	ADD_TO_FUNCT_MSG_LOOKUP_TABLE Details	98
Figure 74	DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE Details	98
Figure 75	BUS_ON Details	99
Figure 76	WIN32 registry structure	100
Figure 77	WIN32 Registry Values	101
Figure 78	WIN32 Registry Example	101
Figure 79	Application Interaction with Registry	102
Figure 80	<RxStatus> Bit Definitions	104
Figure 81	<TxFlags> bit definitions	104
Figure 82	<DeviceAvailable> values	104
Figure 83	<DeviceDLLFWStatus> values	105
Figure 84	<DeviceConnectedMedia> values	105
Figure 85	<ProtocolID> values	105
Figure 86	<Flags> values	106
Figure 87	<Connector> values	106
Figure 88	ErrInd indication description code values	106
Figure 89	<SelectType> values	106
Figure 90	<FilterType> values	107
Figure 91	<Voltage> values	107
Figure 92	<IoctlID> values	107
Figure 93	GET_CONFIG / SET_CONFIG <Parameter> Values	108
Figure 94	Pre-defined values for GET_CONFIG / SET_CONFIG parameters	109
Figure 95	Return values	111
Figure 96	CAN Identifier Assignment Example	114
Figure 97	DataBuffer contains message bytes	125
Figure 98	DataBuffer contains extra data	125

1. SCOPE

This SAE Recommended Practice describes a standardized interface that connects between a standard personal computer (PC) and vehicle.

The purpose of this interface is to enable the reprogramming of emission-related control modules, in 2004 and later model year vehicles. The interface shall consist of the necessary hardware and/or software to support the requirements defined in this document. It is expected that vehicle manufacturers will provide the software application that will control the Pass-Thru Interface, to perform the actual reprogramming.

The goal of this document is to ensure that reprogramming software from any vehicle manufacturer is compatible with interface supplied by any tool manufacturer. A common interface for all vehicle manufacturers reduces the tool costs for aftermarket garages, while allowing each vehicle manufacturer to control the programming sequence for the electronic control units (ECUs) in their vehicles. Aftermarket garages will be able to obtain a standardized interface from a variety of tool suppliers.

This document does not limit the hardware possibilities for the connection between the PC and the interface (e.g., RS-232, RS-485, USB, Ethernet...). Tool suppliers are free to choose the connection that is most appropriate for their needs. Tools that are required to be SAE J2534-1 compliant must strictly adhere to the requirements defined in this document.

Applications that are required to be SAE J2534-1 compliant must strictly adhere to the requirements defined in this document. Applications that use features defined in the SAE J2534-2/xxxx document set are not considered to be SAE J2534-1 compliant.

2. REFERENCES

2.1 Applicable Documents

The following publications form a part of this specification to the extent specified herein. Unless otherwise indicated, the latest issue of SAE publications shall apply.

2.1.1 SAE Publications

Available from SAE International, 400 Commonwealth Drive, Warrendale, PA 15096-0001, Tel: 877-606-7323 (inside USA and Canada) or 724-776-4970 (outside USA), www.sae.org.

SAE J1850	Class B Data Communications Network Interface
SAE J1939	Truck and Bus Control and Communications Network (Multiple Parts Apply)
SAE J1962	Diagnostic Connector
SAE J2610	Serial Data Communication Interface
SAE J2284-1	High Speed CAN (HSC) for Vehicle Applications at 125 kbps
SAE J2284-2	High Speed CAN (HSC) for Vehicle Applications at 250 kbps
SAE J2284-3	High Speed CAN (HSC) for Vehicle Applications at 500 kbps

2.1.2 ISO Documents

Available from American National Standards Institute, 25 West 43rd Street, New York, NY 10036-8002, Tel: 212-642-4900, www.ansi.org.

ISO 7637-1:1990	Road Vehicles - Electrical Disturbance by Conduction And Coupling - Part 1: Passenger Cars and Light Commercial Vehicles With Nominal 12 V Supply Voltage
ISO 9141:1989	Road Vehicles - Diagnostic Systems - Requirements for Interchange of Digital Information
ISO 9141-2:1994	Road Vehicles - Diagnostic Systems - CARB Requirements for Interchange of Digital Information
ISO 11898-1:2003	Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling
ISO 11898-1:2003/Cor 1:2006 Corrigenda/Amendment	for Road vehicles -- Controller area network (CAN) -- Part 1: Data link layer and physical signalling
ISO 14230-4:2000	Road Vehicles - Diagnostic Systems - Keyword Protocol 2000 - Part 4: Requirements for Emission-Related Systems
ISO 15765-2:2004	Road Vehicles - Diagnostics on Controller Area Networks (CAN) - Network Layer Services
ISO 15765-4:2005	Road Vehicles - Diagnostics on Controller Area Networks (CAN) - Requirements for Emission-Related Systems

3. DEFINITIONS

3.1 ACRONYMS

API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
CAN	Controller Area Network
CRC	Cyclic Redundancy Check
DLL	Dynamic Link Library
ECU	Electronic Control Unit
ERR	Error
IOCTL	Input / Output Control
ISO	International Standards Organization
KW	Keyword
KWP	Keyword Protocol
OEM	Original Equipment Manufacturer
PC	Personal Computer
PCI	Protocol Control Information byte (as defined in ISO 15765-2)
PWM	Pulse Width Modulation
Rx	Receive
SAE	Society of Automotive Engineers
SCI	Serial Communications Interface (SAE J2610)
SDK	Software Development Kit
SCP	Standard Corporate Protocol
Tx	Transmit
USB	Universal Serial Bus
VPW	Variable Pulse Width

3.2 TERMS

3.2.1 Active Message

Refers to a message that has started the transmission process on the physical vehicle bus but has not completed it. (Examples of an active messages are: a 1024 byte ISO 15765 Segmented Message where the first 27 bytes have been transmitted but the remainder has not, a 6 byte ISO 9141 message where only four bytes have been transmitted, or a J1850 PWM frame that has not yet been fully transmitted by the data link controller.)

3.2.2 Can Frame

A filtered CAN frame that has neither been blocked (by a BLOCK_FILTER set on a Physical CAN Communication Channel) nor passed to a Logical Communication Channel.

3.2.3 Consecutive Frames

One or more CAN frames that are part of an ISO 15765 Segmented Transfer.

3.2.4 Data Link Controller

Refers to low level hardware/software responsible to network communications at the data link layer (as defined in the OSI seven layers). Examples of data link controllers would include: AN82527 (for CAN), an HBCC or LBCC (for SAE J1850 PWM), MC68HC56 (for SAE J1850 VPW), etc.

3.2.5 First Frame

The initial CAN frame that identifies the start of an ISO 15765 Segmented Transfer.

3.2.6 High Impedance

Defined as greater than 500 kOhm impedance relative to Signal Ground, and as greater than 500 kOhm impedance relative to Chassis Ground.

3.2.7 Logical Communication Channel

A specific path through the Pass-Thru Interface to the physical vehicle bus that uses an existing Physical Communication Channel but adds a Network and Transport layer (as defined in the OSI seven layers).

3.2.8 Network Address Information

Refers to the CAN ID and Extended Address (if applicable) portion of an ISO 15765 message

3.2.9 Pass-Thru Device

Refers only to the hardware portion of the SAE J2534 Interface.

3.2.10 Pass-Thru Interface

Refers to all components of a SAE J2534 Interface, including both hardware and software.

3.2.11 Physical Communication Channel

A specific path through the Pass-Thru Interface to the physical vehicle bus that encompasses the physical layer and data link layer (as defined in the OSI seven layers) and includes all the associated physical resources (such as, the data link controller, connector, pins, etc.) in the Pass-Thru Device.

3.2.12 Physical Vehicle Bus

A unique bus system in a vehicle that is used for serial data communication between ECUs and/or off-board devices, which includes the electrical interface as well as the communication protocol.

3.2.13 Queued Message

Refers to the acceptance of a full message from the application by the Pass-Thru Interface (via the API), which is intended to become an Active Message at some point in the future.

3.2.14 Raw CAN Frame

A CAN frame that will be treated as a CAN ID and up to eight data bytes. No special treatment will be given to the CAN ID or any of the data bytes. (That is, none of them will be treated as an ISO 15765 PCI Byte or Extended Address.)

3.2.15 Registry

A mechanism within Microsoft's Windows operating systems to handle hardware and software configuration information.

3.2.16 Segmented Message / Segmented Periodic Message

An ISO 15765 message that is made up of multiple CAN frames (consisting of a First Frame and one or more Consecutive Frames) using the transport layer described in ISO 15765-2.

3.2.17 Segmented Transfer in Progress

A segmented transfer in progress is considered to be an 'Active Message' after a First Frame has been transmitted by any node, including the Pass-Thru Interface.

3.2.18 Single Frame

An ISO 15765 message that can fit in a single CAN frame as described in ISO 15765-2, also known as an Unsegmented Message.

3.2.19 Servicing a Transmit Queue

Refers to the process, used by the Pass-Thru Device, of providing an opportunity for each Physical and/or Logical Communication Channel to transmit. This process will examine the Individual Write and Periodic Write transmit queues to determine if there are any messages to be transmitted.

3.2.20 Terminate a Message

Refers to the process of stopping reception or transmission of a message by the Pass-Thru Interface. (See Section 6.11.3 for more details.)

3.2.21 Transmitted Message

Refers to a message that has been successfully placed on the physical vehicle bus.

3.2.22 Unsegmented Message

An ISO 15765 message that can fit in a single CAN frame as described in ISO 15765-2, also known as a Single Frame.

4. DOCUMENTATION CONVENTIONS

In the document body and in tables, all J2534 API functions shall be in bold print (for example, **PassThruOpen**).

In the document body and in tables, all J2534 API variables shall be placed between "<" and ">" (for example, <DeviceID>).

In the document body and in tables, all J2534 API DEFINES shall be in upper-case print (for example, STATUS_NOERROR).

All numbers shall be in decimal unless otherwise indicated. (Hexadecimal values shall have a leading '\$'; for example, \$7AF or \$800 are hexadecimal values.)

5. PASS-THRU INTERFACE CONCEPT

The use of reprogrammable memory technology in vehicle electronic control units (ECUs) has increased in recent years, and is expected to continue in the future. Use of this technology has increased the flexibility of being able to use a single ECU hardware part in many different vehicle configurations, with the only difference being the software and calibrations programmed into the unit. Reprogramming of those ECUs in the service environment also allows for ease of field modification of system operation and calibrations. Variations in programming capability and the multiple tools necessary to program vehicles are a burden on aftermarket repair facilities that service different makes of vehicles.

The U.S. Environmental Protection Agency (EPA) and the California Air Resources Board (ARB) have been working with vehicle manufacturers to provide the aftermarket with increased capability to service emission-related ECUs for all vehicles with a minimal investment in hardware needed to communicate with the vehicles.

The Society of Automotive Engineers (SAE) developed the SAE J2534 set of Recommended Practices to minimize the equipment needed to service emission-related ECUs. SAE J2534 is comprised of multiple documents:

SAE J2534-1 defines a standard interface that includes a basic set of hardware capabilities as well as an Application Program Interface (API). This API can be used by Vehicle Manufacturers to meet the U.S. Environmental Protection Agency (EPA) and the California Air Resources Board (ARB) "OBD Service Information" regulations requiring programming of emission-related control modules, in 2004 and later model year vehicles, for all manufacturers by the aftermarket repair industry. This API can also be used for other purposes. For some vehicles, this API can also be used to reprogram emission-related control modules in vehicles prior to the 2004 model year and for nonemission-related control modules.

Figure 1 the relationship between the various components used during the programming of emission-related control modules as well as the responsibilities for each component:

Vehicle	Cable	Pass-Thru Device	Cable	Vehicle Programming PC		
				Low Level driver	API	Application
Vehicle manufacturer determines programming sequence and security access	Tool supplier defines connection with Pass-Thru Device SAE J1962 on vehicle end 5 meter max. length	Capabilities defined in SAE J2534-1 Hardware supplied by tool supplier Could be implemented in scan tool	Tool supplier defines cable requirements, if any, between PC and Pass-Thru Device	Tool supplier provides device driver, installation, and setup procedure based on hardware supported Examples are: RS-232, USB, PCMCIA, Ethernet, IEEE1394, Bluetooth	SAE J2534-1	Vehicle manufacturer programming application controls user interface, vehicle software and calibration selection, programming sequence, and security access
Vehicle manufacturer	Tool supplier				SAE J2534	Vehicle manufacturer

Figure 1 - Pass-thru vehicle programming overview

The vehicle manufacturer provides the programming application software, which shall run on a commonly available generic PC. This application must have complete knowledge of the programming requirements for the control module to be programmed and will control the programming event. This includes the user interface, selection criteria for downloadable software and calibration files, the security mechanism to control access to the programming capability, and the actual programming steps and sequence required to program each individual control module in the vehicle. If additional procedures must be followed after the programming event, such as clearing Diagnostic Trouble Codes (DTC), writing part numbers or variant coding information to the control module, or running additional setup procedures, then the vehicle manufacturer must also include these steps in the PC application.

The tool supplier provides the SAE J2534-1 Pass-Thru Interface as well as the connection to both the PC and the vehicle. In addition to the hardware, the tool supplier shall provide device driver software, a Windows installation/setup application (that will install the supplier's SAE J2534 DLL and other required files, and also update the Windows Registry). The interface between the PC and the Pass-Thru Device can be any technology chosen by the tool supplier, including RS-232, RS-485, USB, Ethernet, or any other current or future technology, including wireless technologies.

The API contains a set of functions that may be called by the vehicle programming application to control the Pass-Thru Interface, and to control the communications between the Pass-Thru Device and the vehicle. The Pass-Thru Interface shall not interpret the data content of the message, allowing any message strategy and message structure to be used that is understood by both the programming application and the ECU being programmed. Also, because the message will not be interpreted, the contents of the message cannot be used to control the operation of the Pass-Thru Interface. For example, if a message is transmitted to the ECU to go to high speed, a specific instruction must also command the Pass-Thru Device to go to high speed.

5.1 PC Requirements

A generic PC running the Microsoft Windows Operating System, that is either WIN32 or capable of supporting 32 bit emulation. The PC should be capable of connection to the Internet. Application providers shall identify their minimum PC requirements.

5.1.1 32-bit versus 64-bit Operating Systems

Newer 64-bit operating systems may still be used so long as they provide a 32-bit emulation layer. Through the emulation layer, 32-bit applications can call 32-bit DLLs. However, if the Pass-Thru Interface requires a 64-bit kernel driver, it must also be provided.

5.2 Application Software Requirements and Assumptions

Applications shall be built for a 32-bit environment and can assume that the PC will be connected to the Internet, although not all applications will require this.

Applications shall ensure that only one thread at a time (which may or may not always be the same thread) is calling functions in the Pass-Thru Interface DLL. **If multiple application threads are used to interact with the Pass-Thru Interface DLL, it is the application's responsibility to ensure that these function calls do NOT overlap (that is, one function call must complete BEFORE another one is called).** The requirement to have only a single thread accessing the Pass-Thru Interface DLL applies even if multiple channels are connected.

Applications may use multiple protocols, but only in certain combinations (see Section 6.6).

Applications shall make no assumptions or impose any restrictions on how long any J2534-1 API function may take to complete. Applications are responsible for setting up and controlling any tester present messages; the Pass-Thru Interface will not automatically handle tester present messages – for any protocol.

6. PASS-THRU INTERFACE REQUIREMENTS

6.1 Connection to the PC

6.1.1 Interface between PC and Pass-Thru Device

The manufacturer of the Pass-Thru Interface shall determine the interface between the PC and the Pass-Thru Device. This can be UART, USB, Ethernet, Firewire, wireless (like Bluetooth, 802.11, Cellular, etc.) or any other connection that allows the Pass-Thru Interface to meet all other requirements of this document, including timing requirements. The tool manufacturer is also required to include the device driver that supports this connection so that the actual interface used is transparent to both the PC programming application and the vehicle.

6.1.2 Device disconnect from PC

An interface disconnect between the PC and the Pass-Thru Device must be detected:

- If the disconnect is present when an API function is called
- If the disconnect has caused the loss of data or configuration between API calls

A Device disconnect shall be reported using the return value `ERR_DEVICE_NOT_CONNECTED` in response to all subsequent Pass-Thru function calls (except **PassThruGetLastError**, **PassThruScanForDevices**, and **PassThruGetNextDevice**) until the application calls **PassThruClose** (which may not be successful) and then successfully call **PassThruOpen** before proceeding. See Section 6.11.1 for more details.

6.2 Pass-Thru Interface DLL

The Pass-Thru Interface DLL:

- Shall support back-to-back function calls (that is, no required delay between consecutive function calls)
- Shall never open a window, write to STDOUT, or read from STDIN
- Is not required to be thread-safe

6.3 Configuration Application

If device configuration is required, then the Pass-Thru Device manufacturer shall provide a configuration application where the user can set the different parameters required for successfully using the device, like COM port, Ethernet address, etc. In addition, if the Pass-Thru Device supports a firmware reflash, this feature shall be included in the configuration application as well. Pass-Thru Device manufacturers should also consider using the configuration application to help the User confirm that the Pass-Thru Device is operational and provide guidance when it is not.

6.4 Connection to the Vehicle

The interface between the Pass-Thru Device and the vehicle shall be an SAE J1962 connector for serial data communications. The maximum cable length between the Pass-Thru Device and the vehicle is five (5) meters. The interface shall include an insulated banana jack that accepts a standard 0.175" diameter banana plug as the auxiliary pin for connection of programming voltage to a vehicle specific connector on the vehicle.

If powered from the vehicle, the Pass-Thru Interface shall:

- Operate normally within a vehicle battery voltage range of 8.0 to 18.0 VDC,
- Survive a vehicle battery voltage of up to 24.0 VDC for at least 10 minutes,
- Survive, without damage to the Pass-Thru Interface, a reverse vehicle battery voltage of up to 24.0 VDC for at least 10 minutes.
- Not draw current exceeding the value specified in SAE J1962, through the power contacts of the diagnostic connector.

6.5 Communication Protocols

6.5.1 ISO 9141

The following specifications clarify and, if in conflict with ISO 9141, override any related specifications in ISO 9141:

- The maximum sink current to be supported by the Pass-Thru Interface is 100 mA.
- The range for all tests performed relative to ISO 7637-1 is -1.0 to $+40.0$ V.
- During 5 Baud initialization, the default bus idle period (W0_MIN) before the Pass-Thru Interface shall transmit an address shall be 300 ms.
- Shall support following baud rate with $\pm 0.5\%$ tolerance: 10400.
- Shall support following baud rate with $\pm 1\%$ tolerance: 10000.
- Shall support following baud rates with $\pm 2\%$ tolerance: 4800, 9600, 9615, 9800, 10870, 11905, 12500, 13158, 13889, 14706, 15625, 19200, and 115200.
- The baud rate shall be set by the application, not determined by the Pass-Thru Interface. The Pass-Thru Interface shall not support baud rate detection based on the synchronization byte.
- Support odd and even parity in addition to the default of no parity, with seven or eight data bits. Always one start bit and one stop bit.
- Support for timer values that are less than or greater than those specified in ISO 9141 (see Figure 63).
- Support ability to disable automatic ISO 9141-2 / ISO 14230 checksum verification by the Pass-Thru Interface to allow vehicle manufacturer specific error detection.
- If the ISO 9141 checksum is verified by the Pass-Thru Interface, and the checksum is incorrect, the message will be discarded.
- Support **PassThruIoctl** with the IOCTL ID equal to FAST_INIT and FIVE_BAUD_INIT.
- The Pass-Thru Interface shall not adjust timer parameters based on Keyword values.
- The Pass-Thru Interface shall not enforce any delays between a Fast Initialization and a 5-baud Initialization. It is the application's responsibility to ensure these actions do not interfere with the vehicle's initialization process.
- The Pass-Thru Interface shall receive and filter all messages on the data link without transmitting a request.
- The Pass-Thru Interface shall be capable of handling P1_MIN = 0.

- The Pass-Thru Interface shall use P1_MAX to indicate the end of a message.
- After transmitting a request, the Pass-Thru Interface shall be capable of handling an immediate response (equivalent to P2_MIN = 0). If there is no response, the Pass-Thru Interface shall wait P2_MAX (P3_MIN when there is a response) before transmitting the next message.
- The Pass-Thru Interface shall not check for P2_MAX violations.
- The Pass-Thru Interface shall detect corrupted transmissions and not attempt to resend the message. (A corrupt transmission is when the transmit byte does not match the corresponding byte from the physical vehicle bus.)
- The Pass-Thru Interface shall not perform any special handling for the \$78 response code. Any message received with a \$78 response code will be passed from the Pass-Thru Interface to the application. The application is required to handle any special timing requirements based on receipt of this response code, including stopping any periodic messages.

6.5.2 ISO 14230 (KWP2000)

The ISO 14230 protocol has the same specifications as the ISO 9141 protocol as outlined in the previous section. In addition, the following specifications clarify and, if in conflict with ISO 14230, override any related specifications in ISO 14230:

- The Pass-Thru Interface shall use the header length field and P1_MAX, whichever occurs first, to determine the end of the message.

6.5.3 SAE J1850 41.6 kbps PWM (Pulse Width Modulation)

In addition to the requirements of SAE J1850, the Pass-Thru Interface must support the following features:

- Shall only support 41.6 kbps and 83.3 kbps.
- The size of the Functional Message Lookup Table shall be eight addresses.
- In addition to communicating on both BUS + and BUS -, new hardware designs should consider supporting network line selection (the ability to communicate on BUS – only or BUS + only).

6.5.4 SAE J1850 10.4 kbps VPW (Variable Pulse Width)

In addition to the requirements of SAE J1850, the Pass-Thru Interface must support the following features:

- Shall only support 10.4 kbps and 41.6 kbps
- Shall support up to a maximum of 4128 byte block transfer
- Shall return to 10.4 kbps after a Break signal is detected on the physical vehicle bus

6.5.5 CAN

In addition to the requirements of ISO 11898 (CAN), the Pass-Thru Interface must support the following features:

- Support only 125000, 250000, and 500000 data rates.
- Support 11 and 29 bit identifiers.
- Based on data rate, use any of the CAN Controller settings in the corresponding SAE J2284-1 (for 125kbps), SAE J2284-2 (for 250kbps), or SAE J2284-3 (500kbps).
- The Pass-Thru Interface shall have the physical layer specified by ISO 15765-4:2005.



6.5.6 ISO 15765 (transport layer for CAN)

In addition to the requirements of ISO 15765, the following features must be supported by the Pass-Thru Interface:

- To maintain acceptable programming times, the transport layer Flow Control function, as defined in ISO 15765-2, must be incorporated in the Pass-Thru Device. If the application does not use the ISO 15765-2 transport layer Flow Control functionality, the CAN protocol will allow for any custom transport layer.
- Receive a Segmented Message with an ISO15765_BS of 0 and an ISO15765_STMIN of 0, as defined in ISO 15765-2. Transmit a segmented message with an actual STmin no greater than 1 millisecond more than the STmin value specified in the corresponding Flow Control frame.
- No Single Frame or Segmented messages can be received without a corresponding logical communication channel. No Segmented Messages can be transmitted without a corresponding logical communication channel.
- When frames are padded by the Pass-Thru Interface, the pad bytes shall be defined by IOCTL configuration attribute ISO15765_PAD_VALUE, which defaults to \$00 unless modified.
- The Pass-Thru Interface shall accept any value for pad bytes from the ECU.
- The Pass-Thru Interface shall check each message for padding and set the ISO15765_PADDING_ERROR bit in the RxStatus element of the associated message when a padding violation is detected.
- The Pass-Thru Interface shall support both full and half-duplex communication, as defined ISO 15765-2, for each logical communication channel. Refer to ISO 15765-2 for details on handling the unexpected arrival of N_PDU.
- The Pass-Thru Interface shall have the physical layer specified by ISO 15765-4:2005.

6.5.7 SAE J2610 (also known as CHRYSLER SCI)

In addition to the requirements of SAE J2610 Information Report, the Pass-Thru Interface must support the following features:

- Support only 7812.5 and 62500 data rates
- SCI A and SCI B are mutually exclusive and do not operate simultaneously.

When transmitting in the half-duplex mode (when SCI_MODE of <TxFlags> is set to 1), every data byte transmitted is expected to be "echoed" by the ECU. The next data byte shall not be transmitted until the echo byte has been received and verified. If the echoed byte received doesn't match the transmitted byte, or if after a period of T1_MAX no response was received, then the transmission shall be terminated. Matching echoed bytes will not be placed in the receive message queue.

When receiving messages from a controller, received bytes shall never be "echoed" by the Pass-Thru Interface.

6.6 Simultaneous Communication on Multiple Protocols

The Pass-Thru Interface shall support simultaneous communication on multiple protocols during a single programming event. Figure 2 indicates the only combinations of protocols shall be supported. If SAE J2610 communication is not required during the programming event, the Pass-Thru Interface shall be capable of simultaneous communication on one protocol from data link set 1, one protocol data link set 2, and one protocol data link set 3. If SAE J2610 communication is required during the programming event, the Pass-Thru Interface shall be capable of simultaneous communication on one SAE J2610 protocol and one protocol from data link set 1.

	Data Link Set 1	Data Link Set 2	Data Link Set 3
Without SAE J2610 Active	SAE J1850 VPW SAE J1850 PWM	ISO 9141 ISO 14230	CAN (and/or ISO 15765)
With SAE J2610 Active	SAE J1850 VPW SAE J1850 PWM	SCI A (either Engine or Transmission) SCI B (either Engine or Transmission)	None

Figure 2 - Simultaneous communications on multiple protocols

6.7 Programmable Power Supply

The Pass-Thru Interface shall have a programmable power supply that meets the following requirements:

- Minimum 5 V DC
- Maximum 20 V DC
- Resolution 0.1V DC
- Accuracy $\pm 2\%$ of requested voltage
- Maximum source current 150 mA
- Maximum sink current 300mA (only for SHORT_TO_GROUND on pin 15). The maximum output voltage at SHORT_TO_GROUND is 0.5 V DC. The maximum input voltage is 20 V DC.
- Maximum 1 ms settling time (required for SAE J2610 channels only, reference SAE J2610 Information Report for more details)
- Pin assignment software selectable
- Setting the pin to PIN_OFF shall place the pin at high impedance (500 KOhm), except for SAE J2610 channels (where the pin shall not return to high impedance until the channel has been disconnected).
- The ground reference for both the SHORT_TO_GROUND and the variable voltage output shall be Signal Ground, defined as pin 5 on the J1962 connector.

The programmable power supply shall be capable of supplying to all of the following pins (one pin at any given time):

- SAE J1962 connector pin 6
- SAE J1962 connector pin 9
- SAE J1962 connector pin 11
- SAE J1962 connector pin 12
- SAE J1962 connector pin 13
- SAE J1962 connector pin 14
- An auxiliary pin (female banana jack) that would be connected to the vehicle via a vehicle specific cable (see Section 7.3.16).

The Pass-Thru Interface shall also be capable of shorting pin 15 on SAE J1962 connector to ground.

6.8 Pin Usage

Figure 3 indicates the SAE J2534-1 uses for each pin of the SAE J1962 connector and for the auxiliary pin. This figure also indicates the default state for each pin. When the Pass-Thru Interface is initially powered or when a pin is no longer used to supply a programming voltage, to supply a short to ground, or for serial data communication, it shall be placed in the default state. In the figure, high impedance is defined as greater than 500 kW impedance relative to Signal Ground, and as greater than 500 kΩ impedance relative to Chassis Ground.

Pin Number	Possible Uses	Default State
Aux	Auxiliary programming voltage (not part of SAE J1962 connector)	High impedance
1	Shall not be used for SAE J2534-1	High impedance
2	SAE J1850 (+)	High impedance or SAE J1850 (+)*
3	Shall not be used for SAE J2534-1	High impedance
4	Chassis Ground	Chassis Ground
5	Signal Ground	Signal Ground
6	ISO 15765/CAN High Programming Voltage SCI A Engine (Rx)	High impedance
7	ISO 9141/ ISO 14230 K-Line SCI A Engine (Tx) SCI A Trans (Tx) SCI B Engine (Tx)	High impedance
8	Shall not be used for SAE J2534-1	High impedance
9	SCI B Trans (Rx) Programming Voltage	High impedance
10	SAE J1850 (-)	High impedance or SAE J1850(-)*
11	Programming Voltage	High impedance
12	SCI B Engine (Rx) Programming Voltage	High impedance
13	Programming Voltage	High impedance
14	ISO 15765/ CAN Low Programming Voltage SCI A Trans (Rx)	High impedance
15	ISO 9141/ ISO 14230 L-line Short to Ground SCI B Trans (Tx)	High impedance
16	Unswitched battery voltage	Unswitched battery voltage

* It is recommended that new Pass-Thru Interfaces default to high impedance

Figure 3 - SAE J1962 pin usage

6.9 Communication Channels

SAE J2534 uses the concept of a 'channel' to describe how the Pass-Thru Device will interact with the vehicle. This section provides a brief overview of their types and limitations.

A communication channel can be thought of as a path through the Pass-Thru Interface to the vehicle. The SAE J2534 API assigns an identifier, called a 'Protocol ID', to each type of path. There are two classes of communication channels defined in SAE J2534: physical communication channels and logical communication channels.

A physical communication channel defines a specific path through the Pass-Thru Interface to the physical vehicle bus that encompasses the physical layer and data layer (as defined in the OSI seven layers) and includes all the physical resources (data link controllers, connectors, pins, etc.). A physical communication channel has its own receive and transmit queues. Examples of a physical communication channel are SAE J1850 VPW, SAE J1850 PWM, ISO 9141, ISO 14230, SAE J2610, and CAN.

A logical communication channel defines a specific path through the Pass-Thru Interface to the physical vehicle bus that uses an existing physical communication channel but adds a Network and Transport layer (as defined in the OSI seven layers). The only logical communication channel defined in SAE J2534-1 is ISO 15765, which overlays the CAN physical communication channel with an additional protocol that includes a transport mechanism.

Typically, logical communication channels are used for direct communication between two nodes. Logical communication channels have their own transmit and receive queues and can receive Indications (like, Start, TxDone or ErrInd). An example of a logical communication channel would be ISO 15765 communications between a J2534 device and the engine control unit on a vehicle. The ISO 15765 Flow Control messages exchanged between the two nodes allows them to share information regardless of the other activity on the network.

There can be up to ten logical communication channels per physical communication channel. Messages written on any given logical communication channel shall appear on the physical communication channel in the order they were written. However, messages from a logical communication channel may be interleaved with messages from other logical communication channels (that use the same physical communication channel) so long as it does not violate the protocol. Messages written on the physical communication channel shall be transmitted without regard to the protocol of any associated logical communication channels. See **PassThruQueueMsgs** for more details.

Messages received by the SAE J2534-1 Interface follow the physical communication channel through the Pass-Thru Interface back to the application. A copy of the messages may also follow the logical communication channel(s) through the Pass-Thru Interface back to the application. See **PassThruReadMsgs** for more details.

Both physical and logical communication channels can be configured using the IOCTL SET_CONFIG, but there are some configuration attributes that can only be set on physical communication channels and others that can only be set on logical communication channels. The configuration attributes set on one logical communication channel shall have no effect on the configuration of other logical communication channels, even if they are associated with the same physical communication channel. See **PassThruIoctl** for more details.

Message filters can only be started on physical communication channels. Each physical communication channel can have up to ten pass/block message filters (in any combination). All incoming messages shall be subject to any filters defined. Filters set on the physical communication channel shall have no impact on logical communication channels. See **PassThruStartMsgFilter** for more details.

Periodic messages can be started on both physical and logical communication channels. See **PassThruStartPeriodicMsg** for details and restrictions on periodic message transmissions.

6.10 Data Buffering

6.10.1 Message reception

The Pass-Thru Interface shall provide a receive buffer as specified in Figure 4 and shall be capable of receiving and buffering until the receive buffer is full. The number of messages buffered depends on the size of the messages received.

This capability shall be available at all supported baud rates.

<ProtocolID>	Maximum Size of EACH Message	Minimum Receive Message Buffer Size *
CAN	12 Bytes	4128 Bytes
ISO15765	4100 Bytes	4128 Bytes per Logical Communication Channel
J1850PWM	11 Bytes	4128 Bytes
J1850VPW	4128 Bytes	4128 Bytes
ISO9141	4128 Bytes	4128 Bytes
ISO14230	260 Bytes	4128 Bytes
J2610	4128 Bytes	4128 Bytes

* The Pass-Thru Interface shall include additional storage for message overhead (for example, timestamp, etc.)

Figure 4 - Minimum receive message buffer sizes

6.10.2 Message Transmission

The Pass-Thru Interface shall be capable of queuing the number of bytes specified in Figure 5 for individual writes (filled via **PassThruQueueMsgs**). The number of messages buffered depends on the size of the messages being transmitted.

<ProtocolID>	Maximum Size of EACH Message	Minimum Transmit Queue Size for Individual Writes *
CAN	12 Bytes	4128 Bytes
ISO15765	4100 Bytes	4128 Bytes per Logical Communication Channel
J1850PWM	11 Bytes	4128 Bytes
J1850VPW	4128 Bytes	4128 Bytes
ISO9141	4128 Bytes	4128 Bytes
ISO14230	260 Bytes	4128 Bytes
J2610	4128 Bytes	4128 Bytes

* The Pass-Thru Interface shall include additional storage for message overhead (for example, TxFlags, etc.)

Figure 5 - Minimum transmit message buffer sizes

In addition to the transmit queue for individual writes, all physical and logical communication channels shall be capable of queuing periodic messages (filled via **PassThruStartPeriodicMsg**). Physical communication channels without any associated logical communication channels shall queue up to 10 messages. Physical communication channels that have associated logical communication channels must share these 10 periodic messages among all channels (that is, the total of periodic messages for a physical communication channel and all of its associated logical communication channels shall not exceed 10). The periodic messages that are queued for transmission shall have priority over individual messages queued for transmission. Once a periodic message has been queued for transmission, the associated time interval does not start until that message has been successfully transmitted. This eliminates the possibility that multiple copies of the same periodic message will 'stack-up'.

All physical communication channels shall have equal priority and shall be serviced for transmission in a 'round-robin' fashion. At a high level, this means that each channel, in turn, shall be given the opportunity to transmit. Physical communication channels that have associated logical communication channels shall service those channels in a 'round-robin' fashion. However, if there is nothing to transmit or transmission is not allowed at that time (prevented by the physical or protocol layer), then the next channel shall be given the opportunity to transmit. This does NOT imply that transmission must be complete before the next channel can be serviced. Multiple physical communication channels are expected to communicate simultaneously. Multiple logical communication channels are expected to communicate simultaneously and their messages interleaved, where permitted by the underlying protocol. When messages from multiple logical channels are interleaved, the interleaving shall take place at the transmission yield points.

<ProtocolID>	Transmission Yield Point
ISO15765	CAN Frame

Figure 6 - Transmission yield point for logical communication channels

Figure 7 is a conceptual overview of the various operations involved in the transmission process. It is meant to help summarize the message transmission concept (it is not intended to prescribe a specific implementation).

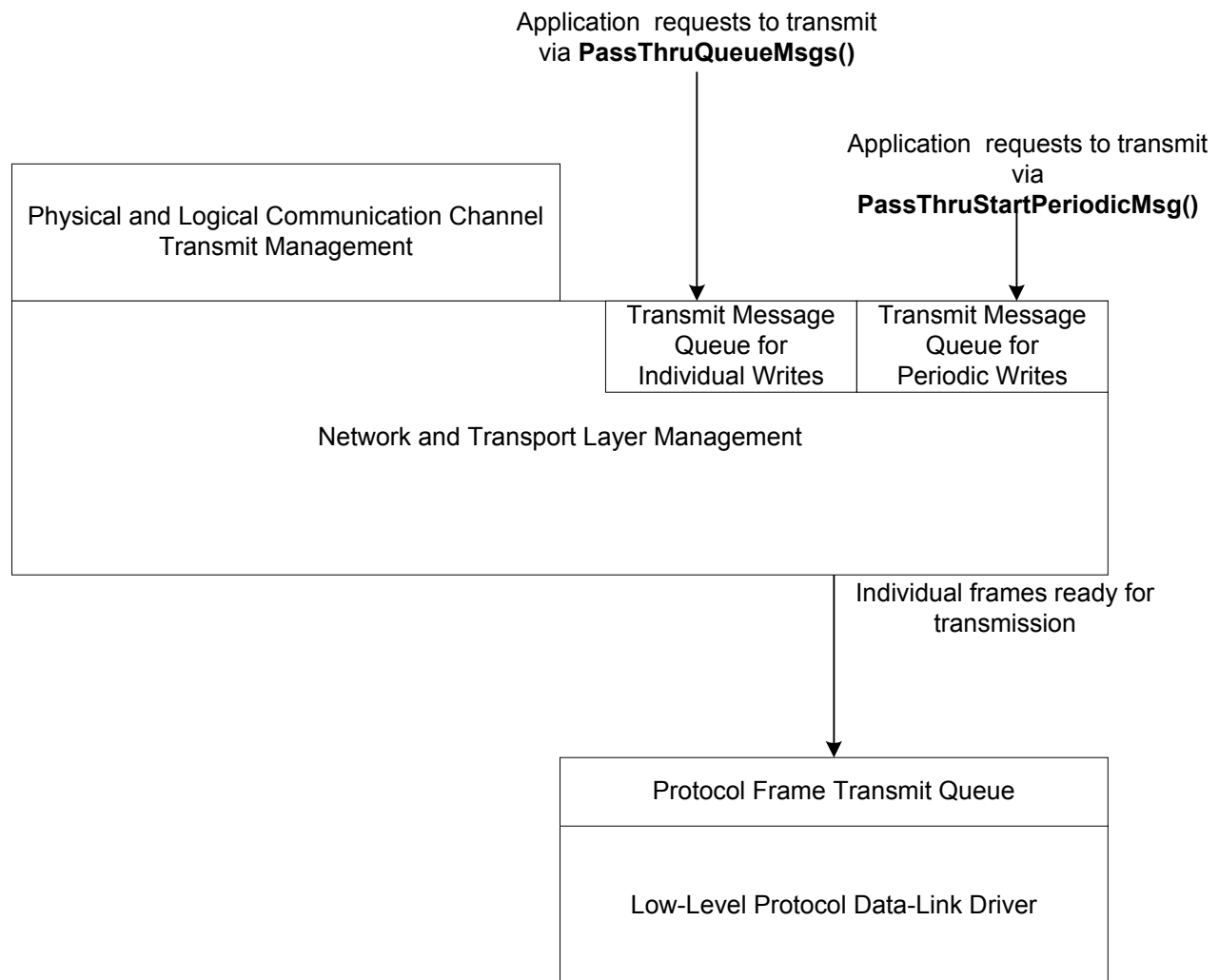


Figure 7 - Overview of the transmission process

In this overview, the properties of the Physical and Logical Communication Channel Transmit Management are:

- Implements the round-robin strategy to ensure no communication channel is starved
- Applies a second level round-robin strategy to Physical Communication Channels with one or more Logical Communication Channels
- Determines if the selected Physical/Logical Communication Channel has a message to transmit

The properties of Physical and Logical Communication Channels are:

- Implements the network and transport layers (where applicable)
- Maintains its own queue for messages transmission requests via `PassThruStartPeriodicMsg()`, which have a higher priority than messages transmission requests via `PassThruQueueMsgs()`
- Maintains its own queue for messages transmission requests via `PassThruQueueMsgs()`
- Provides the next message frame, when given the opportunity to transmit

The properties of Low-Level Data Link Driver are:

- Operates on individual Protocol Frames (for example, CAN Frames)
- Operates asynchronously, independent of the Application's interaction with the Physical/Logical Communication Channel
- Operates on the transmit queue in a FIFO basis
- When the bus is ready/idle, it retrieves the next Protocol Frame from the transmit queue and begins transmission
- As an example, consider the case depicted in Figure 8 where there are three physical communication channels: ISO 9141, SAE J1850 PWM, CAN, and three logical communication channels (ISO 15765 #1, ISO 15765 #2, and ISO 15765 #3) that use CAN physical communication channel.

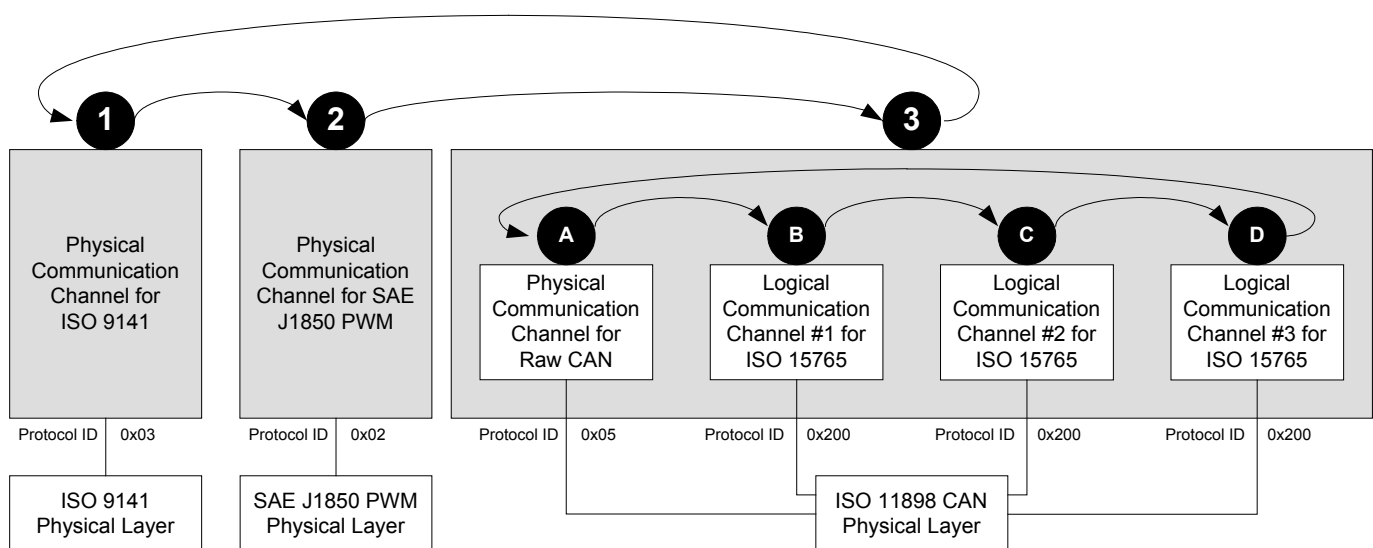


Figure 8 - Communication channel configuration for round-robin message transmission example

Figure 9 indicated the order in which the transmit queues would be serviced.

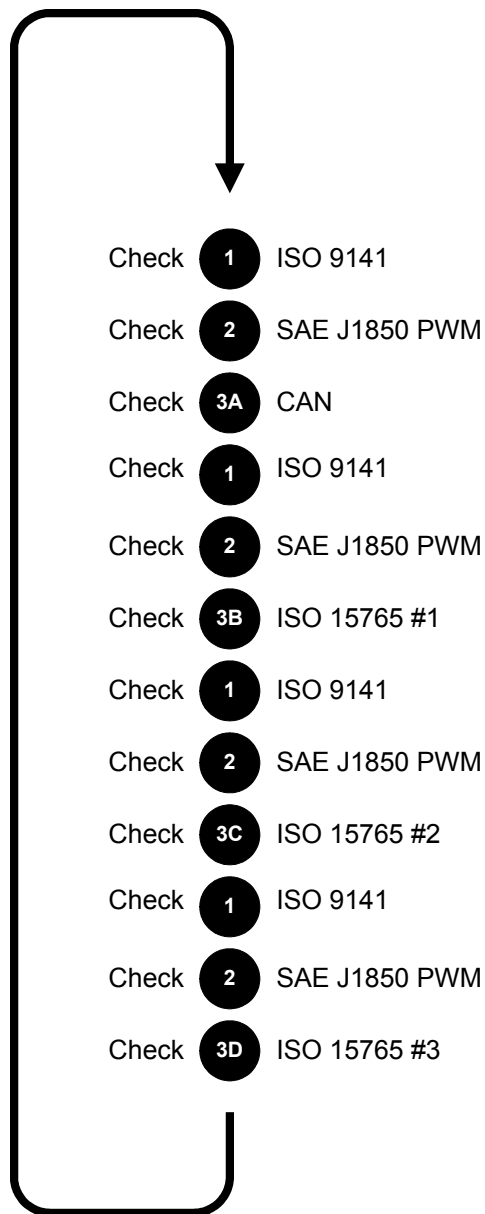


Figure 9 - Service order for round-robin message transmission example

It is assumed that once a message has been handed off to the low level channel transmit routine the transmission will take place in the background along with the continued servicing of the transmit queues. Message transmission continues, by the low level channel transmit routine, until the entire message has been transmitted. However, when a physical communication channel has logical communication channel(s) associated with it, then all of the associated channels must share access to the physical layer in a round-robin fashion. In this case, the point at which a channel shall yield is protocol specific and is listed in Figure 6.

If we continue the example above, assuming that each ISO 15765 channel had at least one Segmented Message in the queue for transmission and that the CAN channel had a number of CAN Frames in the queue for transmission; then Figure 10 show how the message interleaving would look on the CAN physical layer.

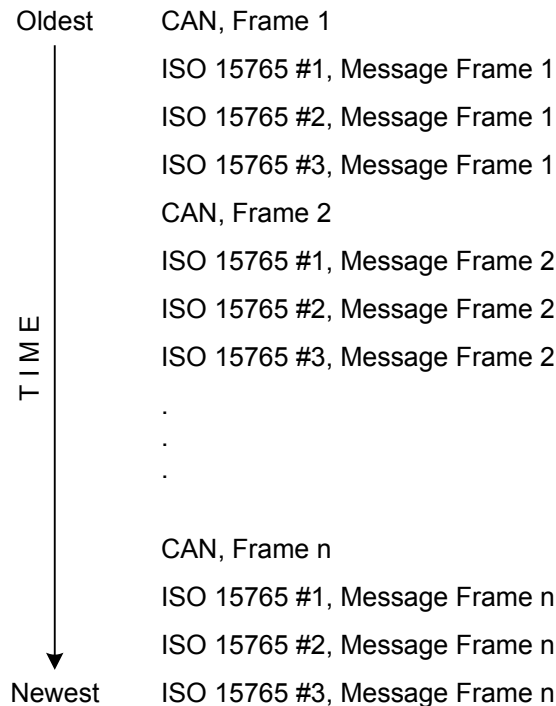


Figure 10 - Message interleaving on the physical CAN layer

6.11 Error Handling

6.11.1 Device Not Connected

The return value `ERR_DEVICE_NOT_CONNECTED` indicates that the Pass-Thru Interface DLL has, at some point, failed to communicate with the Pass-Thru Device – even though it may not currently be disconnected.

6.11.1.1 Pass-Thru Device

When an `ERR_DEVICE_NOT_CONNECTED` error has been detected, all subsequent function calls (with the exception of **PassThruGetLastError**, **PassThruScanForDevices**, and **PassThruGetNextDevice**) shall return `ERR_DEVICE_NOT_CONNECTED` and the Device shall act as if **PassThruClose** has been called. (This assumes that no other higher priority return value is applicable. See Section 7.1 for more details on the prioritization of return values.)

6.11.1.2 Application

When a function returns the value `ERR_DEVICE_NOT_CONNECTED`, the application must re-establish the communication with the Pass-Thru Device. To do this, the application shall call **PassThruClose** (which may return an error code) and then successfully call **PassThruOpen** before proceeding.

6.11.2 Receive Buffer Overflow

A receive buffer overflow may occur at a variety of locations within the Pass-Thru Interface (for example, the data link controller, the device receive buffer, the DLL receive buffer, etc.). When a receive buffer overflow condition occurs, the Pass-Thru Interface shall:

- Discard any partial message received prior to the overflow condition.
- Discard new messages, in their entirety, until the overflow condition has cleared.

- Discard new Indications received until the overflow condition has cleared. (Discarding an Indication shall not alter the normal transmission process.)
- On ISO15765 channels during the reception of a Segmented Message, the FlowStatus field in the Flow Control response message transmitted by the Pass-Thru Interface shall indicate the overflow condition as described in ISO 15765-2.

The loss of messages/Indications shall be conveyed to the application by setting the BUFFER_OVERFLOW bit of the <RxStatus> in the PASSTHRU_MSG structure for the first successfully received message after the overflow condition has cleared.

6.11.3 Message Termination

Unless otherwise noted, when terminating a reception any partial message received shall be discarded. Message reception shall begin again at the start of the next message boundary. (For example, on ISO 15765 logical communication channels the Pass-Thru Interface shall not process any subsequent Consecutive Frames on that logical communication channel until a subsequent First Frame or Single Frame has been issued or received.)

Unless otherwise noted, when terminating a transmission the active message shall be stopped at the next termination boundary (the smallest component of a message like a CAN Frame, UART Byte, etc.) and the associated message discarded from the transmit queue. Any portion of the transmission that is already loaded into the controller when the terminating event occurs shall continue to completion. (For example, on ISO 15765 logical communication channels the Pass-Thru Interface shall not issue any subsequent Consecutive Frames until a subsequent First Frame or Single Frame has been issued or received.) A TxFailed Indication shall be generated if the associated message had requested a TxDone Indication. Any subsequent message transmissions shall follow the protocol specification. Figure 11 lists the message termination boundaries for each protocol.

<ProtocolID>	Termination Boundary
CAN	CAN Frame
ISO15765	CAN Frame
J1850PWM	J1850 Frame
ISO14230	Byte
ISO9141	Byte
J1850VPW	Byte
J2610	Byte

Figure 11 - Message termination boundary

6.11.4 Network Errors

The Pass-Thru Interface shall be capable of detecting transmit and receive errors. To deal with these errors, recovery strategies have been defined in SAE J2534-1. Figure 12 details the types of errors to be detected and the associated recovery strategy as well as the required Application Notification Mechanism that shall occur at the end of the designated Recovery Strategy, which are detailed in Sections 6.11.4.1 through 6.11.4.4.

<ProtocolID>	Error	Application Notification Mechanism	Recovery Strategy
CAN	Acknowledgement error or Loss of arbitration	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy
	Bus warning	NONE	Controller Retry Strategy
	Overrun error	Set BUFFER_OVERFLOW in <RxStatus>	NONE (the message has already been lost)
	During message reception: Form error, Stuff error, CRC error, Bit 0 error, Bit 1 error	NONE	NONE
	During message transmission: Form error, Stuff error, CRC error, Bit 0 error, Bit 1 error	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy
	BUS OFF	Generate LINK_DOWN ErrInd Indication	Passive Operation Strategy
ISO15765	Acknowledgement error or Loss of arbitration	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy
	Bus warning	NONE	Controller Retry Strategy
	Overrun error	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	During message reception: Form error, Stuff error, CRC error, Bit 0 error, Bit 1 error, or errors defined in ISO 15765 (N_TIMEOUT_A, N_ERROR, N_TIMEOUT_CR, N_WRONG_SN, or N_UNEXP_PDU)	NONE	NONE
	During message transmission: Form error, Stuff error, CRC error, Bit 0 error, Bit 1 error	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy

	During message transmission: errors defined in ISO 15765 (N_TIMEOUT_A, N_BUFFER_OVERFLOW, N_TIMEOUT_BS, N_INVALID_FS, or N_ERROR), or ISO15765_WAIT_LIMIT (set in PassThruOctl) has been exceeded	NONE or TxFailed Indication (if a TxDone was requested)	Drop Strategy
	BUS OFF	Generate LINK_DOWN ErrInd Indication	Passive Operation Strategy
J1850PWM	No acknowledgement or Loss of arbitration	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy
	Bus fault	NONE	Controller Retry Strategy
	Receiver overrun	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	Receive error, CRC error, Unable to acknowledge, or Network Fault	NONE	Drop Strategy
J1850VPW	Loss of Arbitration or Network Fault (Transmit error, or Bus short error)	NONE or TxFailed Indication (if a TxDone was requested)	Controller Retry Strategy
	Overrun error	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	CRC error or Network Fault (Symbol error, Framing error, or Bit timing)	NONE	Drop Strategy
	Break	Generate Break Indication	Not Applicable
ISO9141	Receiver overrun UART error	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	During message reception: All UART errors (except receiver overrun), or Invalid checksum (if enabled)	NONE	Drop Strategy
	During message transmission: All UART errors (except receiver overrun), receive byte mismatch, or Transceiver detects fault	NONE or TxFailed Indication (if a TxDone was requested)	Drop Strategy

ISO14230	Receiver overrun UART error	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	During message reception: All UART errors (except receiver overrun), Invalid checksum (if enabled), Incorrect length, or Invalid format byte in header (size mismatch or invalid A0/A1)	NONE	Drop Strategy
	During message transmission: All UART errors (except receiver overrun), receive byte mismatch, or Transceiver detects fault	NONE or TxFailed Indication (if a TxDone was requested)	Drop Strategy
J2610	Receiver overrun UART error	Set BUFFER_OVERFLOW in <RxStatus>	Drop Strategy
	During message reception: All UART errors (except receiver overrun)	NONE	Drop Strategy
	During message transmission: Echo byte mismatch (SCI_MODE = 1 and transmitting a message)	NONE or TxFailed Indication (if a TxDone was requested)	Drop Strategy
	Break	Generate Break Indication	Not Applicable

Figure 12 - Network errors**6.11.4.1 Controller Retry Strategy**

The Controller Retry strategy is when the data link controller automatically retries without any external intervention. This may happen because of a loss of arbitration or some other event that the controller is capable of handling without external intervention (typically, these features are built-in to the network controller chip). The controller may try to send the message a specific number of times or may continually attempt until it is successful, so long as it follows the protocol specification. If the data link controller stops retrying without successfully sending the packet, the Pass-Thru Interface shall treat the current message transmission as unsuccessful and remove it from the transmit queue. If the message being retried is designated to generate a TxDone Indication, then a single copy of that Indication shall be placed in the receive queue but only after transmission is successful. The unsuccessful transmission of a message designated to generate a TxDone Indication, shall result in the generation of a single TxFailed Indication but only after the transmission attempts have stopped. If the data link controller does not support retries, the firmware/software shall not retry.

6.11.4.2 Interface Retry Strategy

The Interface Retry strategy shall keep trying to send the message until it is successful or the associated Timeout expires and/or after a specified number of retries. Even if the data link controller does not support retries, the firmware/software shall continue to retry. If the message being retried is designated to generate a TxDone Indication, then a single copy of that Indication shall be placed in the receive queue but only after transmission is successful. The unsuccessful transmission of a message designated to generate a TxDone Indication, shall result in the generation of a TxFailed Indication.

6.11.4.3 Drop Strategy

The Drop strategy shall immediately terminate the message and discard it as specified in Section 6.11.3.

6.11.4.4 Passive Operation Strategy

The data link controller in the Pass-Thru Interface shall initiate the Passive Operation strategy, when the designated error conditions are detected. Channels that are in Passive Operation shall stop participation in message transmissions and possible message receptions (depending upon the capabilities of the Pass-Thru Interface).

If this occurs during transmission and the data link controller is restored to operational state before the appropriate protocol timer(s) have expired, then the transmission shall continue with the last incomplete transmission. Otherwise, the Pass-Thru Interface shall terminate all active messages as specified in Section 6.11.3.

If this occurs during reception and the data link controller is restored to operational state before the appropriate protocol timer(s) have expired, then the reception shall continue. Otherwise, the Pass-Thru Interface shall terminate the message as specified in Section 6.11.3.

The application shall use any one of the mechanisms described below, to return the Pass-Thru Interface to the operational state:

1. Active Reconnect – Call **PassThruDisconnect** followed by a call to **PassThruConnect**. This will disconnect then reconnect the physical communication channel, resetting it to its default state. This means that any calls to configure the channel, set filters, or start periodic messages must be redone. Additionally, any calls to connect, configure, or start periodic messages on logical communication channels must also be redone.
2. Command BUS ON – Call **PassThruIoctl** with the IOCTL ID of BUS_ON and the Channel ID of the physical communication channel to restore the controller to the configuration just before it moved into the BUS OFF state. In this case, there is no need to redo any calls to connect, configure, set filters, or start periodic messages on the physical communication channel or any of its associated logical communication channels. However, any message(s) being transmitted when the channel moved to the BUS OFF state will be lost. The application is responsible for re-transmitting these message(s), if desired.

Previous versions of SAE J2534-1 had the Pass-Thru Interface automatically reset the CAN controller and attempt the transmission again. This is no longer the case, it is now the responsibility application to attempt to re-establish communications.

7. WIN32 APPLICATION PROGRAMMING INTERFACE

7.1 API Functions - Overview

The vendor supplied API implementation shall be a 32-bit DLL, which shall support all the functions included in Figure 13.

Function	Description
PassThruScanForDevices	Search for all connected Pass-Thru Devices.
PassThruGetNextDevice	Return the list of Pass-Thru Devices, which was created by the last call to PassThruScanForDevices.
PassThruOpen	Establish a connection with a specific Pass-Thru Device.
PassThruClose	Terminate a connection with a specific Pass-Thru Device.
PassThruConnect	Establish a physical connection to the vehicle using the interface hardware in a given Pass-Thru Device.
PassThruDisconnect	Terminate a physical connection to the vehicle in a specific Pass-Thru Device and return the interface hardware to a default state.
PassThruLogicalConnect	Establish a logical communication channel over an existing physical connection.
PassThruLogicalDisconnect	Terminate a logical communication channel over an existing physical connection.
PassThruSelect	Select the channels to be monitored.

Function	Description
PassThruReadMsgs	Read message(s) from a channel.
PassThruQueueMsgs	Queues message(s) to a channel for transmission.
PassThruStartPeriodicMsg	Start sending a message at a specified time interval on a channel.
PassThruStopPeriodicMsg	Stop a periodic message.
PassThruStartMsgFilter	Start filtering incoming messages on a channel.
PassThruStopMsgFilter	Stops filtering incoming messages on a channel.
PassThruSetProgrammingVoltage	Set a programming voltage on a specific pin.
PassThruReadVersion	Returns the API, DLL, and Firmware version information.
PassThruGetLastError	Gets the text description of the last error.
PassThruIoctl	General I/O control functions for protocol specific control (e.g. specialized initializations, set/get configuration parameters, clearing queues, etc.).

Figure 13 - SAE J2534 API functions

The API definition of each function includes a list of return values, which shall be used to convey the success or failure of the function call. These return values are ordered according to importance - highest (at the top of the table) to lowest (at the bottom of the table). In general, return values have the following order (highest to lowest): errors related to connection, errors related to in use, errors related to special parameters, errors related to normal parameters (in order of use). If multiple errors occur during a function call, the appropriate return value with the highest importance shall always be returned. The return values ERR_FAILED is intended for debugging purposes ONLY. A fully compliant SAE J2534-1 Pass-Thru Interface shall never return ERR_FAILED or ERR_NOT_SUPPORTED.

The API also defines a common structure to hold all messages. Under certain circumstances, the Pass-Thru Interface may also use this structure to provide a notification that a specific event has occurred. Called "Indications", these events will be marked by the creation of a 'pseudo' message that is event specific, which shall be placed in the receive queue along with the other incoming messages in chronological order. (See Section 7.2.6 for more details.)

7.2 Message Structure

Messages are passed between the application and the Pass-Thru Interface via the PASSTHRU_MSG structure. This structure is protocol independent, but the exact content and limitations of some fields is not. The following structure will be used for messages and Indications.

7.2.1 C / C++ Definition

```
typedef struct {
    unsigned long ProtocolID;
    unsigned long MsgHandle;
    unsigned long RxStatus;
    unsigned long TxFlags;
    unsigned long Timestamp;
    unsigned long DataLength;
    unsigned long ExtraDataIndex;
    unsigned char *DataBuffer;
    unsigned long DataBufferSize;
} PASSTHRU_MSG;
```

7.2.2 Description of Structure Elements

<ProtocolID>	The Protocol type; For physical communication channels, this shall match the <ProtocolID> that was passed into PassThruConnect when the channel was connected; For logical communication channels, this shall match the <ProtocolID> that was passed into PassThruLogicalConnect when the channel was connected.
<MsgHandle>	A number, assigned by the application, which can be used by the application to uniquely identify the corresponding message. This value will be returned in the <MsgHandle> of a TxDone Indication to identify the successful transmission of the corresponding message. Setting this value to 0 shall disable the generation of a TxDone/TxFailed Indications for the message.
<RxStatus>	The receive message status. See Section 7.2.2.1 for more details.
<TxFlags>	The transmit message flags. See Section 7.2.2.2 for more details.
<Timestamp>	The message/event timestamp (in microseconds) with a resolution of at least 1000 microseconds. The timestamp shall be a 'free-running' 32-bit timer that starts at 0 on power-up. The counter shall not be reset unless it rolls over. For messages, the <Timestamp> marks the end of the last bit/frame of the message on the bus. For Indications, the <Timestamp> is event specific – see Section 7.2.6 for more details.
<DataLength>	The number of bytes in the array <DataBuffer> that contain valid data.
<ExtraDataIndex>	The zero-based offset into the array <DataBuffer> where the extra data begins. <ExtraDataIndex> shall never be larger than <DataLength>. For SAE J2534-1, this shall only be used for Indications.
<DataBuffer>	The array of data bytes, including message headers, message body, and extra data bytes (if any). For some Indications, this may contain event related data. The application is responsible for the allocation and freeing of this memory. The Pass-Thru Interface shall never use this <DataBuffer> after the function call returns; the persistence of the <DataBuffer> is only guaranteed within the current function call.
<DataBufferSize>	The number of bytes, allocated by the application, for the array <DataBuffer>. The Pass-Thru Interface shall never change this value.

7.2.2.1 <RxStatus>

The <RxStatus> element provides additional details to the application regarding the message/Indication. This is a bit mapped field, which is described in Figure 14. The corresponding bit positions are defined in Section 9.1. As some bits are 'Reserved' or may not be applicable to all channels, the application shall ignore any bits that do not apply to the current channel. All unused/reserved bits shall be set to zero (0).

Name	Description	Value
TX_FAILED	Indicates that a transmission (where a TxDone Indication is required) has failed	0 = No transmission failure 1 = Transmission failure
CAN_29BIT_ID	Indicates the CAN ID Type for CAN and ISO 15765 messages	0 = 11-bit Identifier 1 = 29-bit Identifier
ISO15765_ADDR_TYPE	Indicates the ISO 15765-2 Addressing Method	0 = No extended address 1 = Extended address is first byte after the CAN ID
BUFFER_OVERFLOW	Indicated that the Pass-Thru Interface had a buffer overflow.	0 = No buffer overflow 1 = Buffer overflow occurred prior to this message – data was lost.
ERROR_INDICATION	Indicates that the Pass-Thru Interface detected an error. The message contains error information. (See Section 7.2.6 for more details.)	0 = No ErrInd Indication 1 = ErrInd Indication
ISO15765_PADDING	Indicates if a CAN frame was received with less than 8 data bytes during ISO 15765 communications.	0 = Padded 1 = Not Padded
TX_SUCCESS	Indicates that the message is a TxDone Indication. (This signifies that the corresponding message has been successfully transmitted. See Section 7.2.6 for more details.)	0 = No TxDone Indication 1 = TxDone Indication
RX_BREAK	Indicates that the message is a Break Indication. (See Section 7.2.6 for more details.)	0 = No Break Indication 1 = Break Indication
START_OF_MESSAGE	Indicates that the message is a Start Indication. (See Section 7.2.6 for more details.)	0 = No Start Indication 1 = Start Indication
TX_MSG_TYPE	Indicates that the source of the message was a transmission	0 = Received Message 1 = Transmitted Message

Figure 14 - RxStatus bit definitions

7.2.2.2 <TxFlags>

The <TxFlags> element provides additional details to the Pass-Thru Interface regarding the transmission of the message. It is a bit mapped field and is described in Figure 15. The corresponding bit positions are defined in Section 9.2. All unused bits shall be set to zero (0).

Name	Description	Value
SCI_TX_VOLTAGE	Enables the SAE J2610 programming voltage	0 = no voltage after message transmit 1 = apply 20V after message transmit as specified in SAE J2610
SCI_MODE	Identifies the SAE J2610 transmit mode	0 = Transmit using SCI full-duplex mode (ECU does not echo transmitted byte) 1 = Transmit using SCI half-duplex mode (ECU echo's transmitted byte)
WAIT_P3_MIN_ONLY	Modifies message timing for ISO 9141 or ISO 14230. (The intent is to decrease programming time if the application knows only one response will be received.)	0 = If an ECU response has been received, transmission cannot occur until the greater of P2_MAX or P3_MIN expires; without an ECU response, transmission cannot occur until P2_MAX expires. 1 = With or without an ECU response the Pass-Thru Interface shall ignore P2_MAX and wait P3_MIN only, regardless of address type.
CAN_29BIT_ID	Identifies the type of CAN ID	0 = 11-bit 1 = 29-bit
ISO15765_ADDR_TYPE	Identifies the ISO 15765-2 addressing method	0 = no extended address 1 = extended address is first byte after the CAN ID
ISO15765_FRAME_PAD	Enables ISO 15765-2 Frame Padding	0 = no padding 1 = pad the message to a full CAN frame(s) using the value specified by ISO15765_PAD_VALUE

Figure 15 - TxFlags bit definitions**7.2.2.3 <ExtraDataIndex>**

Under certain conditions, the Pass-Thru Interface needs to return additional data or non-message data to the application via the PASSTHRU_MSG Structure. These extra bytes shall immediately follow the message (if any) in the array <DataBuffer>. <ExtraDataIndex> contains the zero-based offset into the array <DataBuffer> where the first extra byte resides. When no extra bytes are present, <ExtraDataIndex> shall be set equal to <DataLength>. If <ExtraDataIndex> equals zero, then all bytes in the array are extra bytes. SAE J1850 (VPW and PWM) In-Frame Responses and CRC's as well as ISO 9141 and ISO 14230 Checksums shall not be returned as extra data.

7.2.3 Protocol Specific Details for the PASSTHRU_MSG Structure

Figure 16 details the protocol specific fields in the PASSTHRU_MSG structure. This includes the minimum/maximum values for <DataLength>, the applicable <TxFlags>, and the general format of the array <DataBuffer>.

<ProtocolID>	Minimum / Maximum Values of <DataLength> for				Applicable <TxFlags> PassThruQueueMsgs and PassThruStartPeriodicMsg (See PassThruStartMsgFilter for applicable <TxFlags> for filters)
	PassThruReadMsgs	PassThruQueueMsgs	PassThruStatPeriodicMsg	PassThruStatMsgFilter	
CAN	4 / 12	4 / 12	4 / 12	1 / 12	CAN_29BIT_ID
ISO15765	5 / 4099	5 / 4099	5 / 11	N/A	CAN_29BIT_ID ISO15765_ADDR_TYPE ISO15765_FRAME_PAD
ISO15765 (Extended Address)	6 / 4100	6 / 4100	6 / 11	N/A	CAN_29BIT_ID ISO15765_ADDR_TYPE ISO15765_FRAME_PAD
J1850PWM	3 / 11	3 / 10	3 / 10	1 / 10	No TxFlags allowed
J1850VPW	1 / 4128	1 / 4128	1 / 12	1 / 12	No TxFlags allowed
ISO9141 (PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	1 / 4128	1 / 4128	1 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO9141 (PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	1 / 4128	1 / 4128	1 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (No Address, no Length byte, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	2 / 64	2 / 64	2 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (No Address, Length byte, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	3 / 257	3 / 257	3 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (Exception Mode (CARB) where Address, no Length byte, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	4 / 10	4 / 10	4 / 10	1 / 10	WAIT_P3_MIN_ONLY
ISO14230 (Address, no Length byte, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	4 / 66	4 / 66	4 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (Address, Length byte, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 0)	5 / 259	5 / 259	5 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (No Address, no Length byte, Checksum, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	3 / 65	3 / 65	3 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (No Address, Length byte, Checksum, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	4 / 258	4 / 258	4 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (Exception Mode (CARB) where Address, no Length byte, Checksum, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	5 / 11	5 / 11	5 / 11	1 / 11	WAIT_P3_MIN_ONLY
ISO14230 (Address, no Length byte, Checksum, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	5 / 67	5 / 67	5 / 12	1 / 12	WAIT_P3_MIN_ONLY
ISO14230 (Address, Length byte, Checksum, PassThruConnect parameter <Flags> has CHECKSUM_DISABLED set to 1)	6 / 260	6 / 260	6 / 12	1 / 12	WAIT_P3_MIN_ONLY
J2610	1 / 4128	1 / 4128	1 / 12	1 / 12	SCI_TX_VOLTAGE SCI_MODE

Figure 16 - Allowed message sizes per protocol

7.2.3.1 DataBuffer definition for <ProtocolID> = CAN

- <DataBuffer[0]> is CAN ID bits 28-24 (the three most significant bits shall be zero)
- <DataBuffer[1]> is CAN ID bits 23-16
- <DataBuffer[2]> is CAN ID bits 15-8
- <DataBuffer[3]> is CAN ID bits 7-0
- <DataBuffer[4...(x-1)]> is up to 8 data bytes

7.2.3.2 DataBuffer definition for <ProtocolID> = ISO15765

- <DataBuffer[0]> is CAN ID bits 28-24 (the three most significant bits shall be zero)
- <DataBuffer[1]> is CAN ID bits 23-16
- <DataBuffer[2]> is CAN ID bits 15-8
- <DataBuffer[3]> is CAN ID bits 7-0
- <DataBuffer[4...(x-1)]> is up to 4095 data bytes

7.2.3.3 DataBuffer definition for <ProtocolID> = ISO15765 (Extended Address)

- <DataBuffer[0]> is CAN ID bits 28-24 (the three most significant bits shall be zero)
- <DataBuffer[1]> is CAN ID bits 23-16
- <DataBuffer[2]> is CAN ID bits 15-8
- <DataBuffer[3]> is CAN ID bits 7-0
- <DataBuffer[4]> is the extended address
- <DataBuffer[5...(x-1)]> is up to 4095 data bytes

7.2.3.4 DataBuffer definition for <ProtocolID> = J1850PWM

- <DataBuffer[0]> is Priority/Type for transmitted messages; for received messages this byte shall be ignored by the application – see Appendix A.2.17 for more details
- <DataBuffer[1]> is Destination Address or Function Code
- <DataBuffer[2]> is Source Address
- <DataBuffer[3...(x-1)]> is up to 7 data bytes

7.2.3.5 DataBuffer definition for <ProtocolID> = J1850VPW

- Format not defined

7.2.3.6 DataBuffer definition for <ProtocolID> = ISO9141 (**PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- Format not defined

7.2.3.7 DataBuffer definition for <ProtocolID> = ISO9141 (**PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- Format not defined

7.2.3.8 DataBuffer definition for <ProtocolID> = ISO14230 (No Address, no Length byte, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 0, and L5 to L0 are non-zero
- <DataBuffer[1...(x-1)]> is 1 to 63 data bytes

NOTE: No address information; Length byte is not present

7.2.3.9 DataBuffer definition for <ProtocolID> = ISO14230 (No Address, Length byte, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 0, and L5 to L0 are 0
- <DataBuffer[1]> is Length byte
- <DataBuffer[2...(x-1)]> is 1 to 255 data bytes

NOTE: No address information; Length byte is present

7.2.3.10 DataBuffer definition for <ProtocolID> = ISO14230 (Exception Mode (CARB) where Address, no Length byte, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 1, and L5 to L0 are non-zero
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3...(x-1)]> is 1 to 8 data bytes

NOTE: Address information; Length byte is not present

7.2.3.11 DataBuffer definition for <ProtocolID> = ISO14230 (Address, no Length byte, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- <DataBuffer[0]> is Format byte where: A1 is 1, A0 is 1 or 0, and L5 to L0 are non-zero
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3...(x-1)]> is 1 to 63 data bytes

NOTE: Address information; Length byte is not present

7.2.3.12 DataBuffer definition for <ProtocolID> = ISO14230 (Address, Length byte, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 0)

- <DataBuffer[0]> is Format byte where: A1 is 1, A0 is 1 or 0, and L5 to L0 are 0
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3]> is Length byte
- <DataBuffer[4...(x-1)]> is 1 to 255 data bytes

NOTE: Address information; Length byte is present

7.2.3.13 DataBuffer definition for <ProtocolID> = ISO14230 (No Address, no Length byte, Checksum, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 0, and L5 to L0 are non-zero
- <DataBuffer[1...(x-1)]> is 1 to 63 data bytes
- <DataBuffer[x]> is Checksum

NOTE: No address information; Length byte is not present

7.2.3.14 DataBuffer definition for <ProtocolID> = ISO14230 (No Address, Length byte, Checksum, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 0, and L5 to L0 are 0
- <DataBuffer[1]> is Length byte
- <DataBuffer[2...(x-1)]> is 1 to 255 data bytes
- <DataBuffer[x]> is Checksum

NOTE: No address information; Length byte is present

7.2.3.15 DataBuffer definition for <ProtocolID> = ISO14230 (Exception Mode (CARB) where Address, no Length byte, Checksum, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- <DataBuffer[0]> is Format byte where: A1 is 0, A0 is 1, and L5 to L0 are non-zero
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3...(x-1)]> is 1 to 63 data bytes
- <DataBuffer[x]> is Checksum

NOTE: Address information; Length byte is not present

7.2.3.16 DataBuffer definition for <ProtocolID> = ISO14230 (Address, no Length byte, Checksum, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- <DataBuffer[0]> is Format byte where: A1 is 1, A0 is 1 or 0, and L5 to L0 are non-zero
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3...(x-1)]> is 1 to 63 data bytes
- <DataBuffer[x]> is Checksum

NOTE: Address information; Length byte is not present

7.2.3.17 DataBuffer definition for <ProtocolID> = ISO14230 (Address, Length byte, Checksum, **PassThruConnect** parameter <Flags> has CHECKSUM_DISABLED set to 1)

- <DataBuffer[0]> is Format byte where: A1 is 1, A0 is 1 or 0, and L5 to L0 are 0
- <DataBuffer[1]> is Target Address byte
- <DataBuffer[2]> is Source Address byte
- <DataBuffer[3]> is Length byte
- <DataBuffer[4...(x-1)]> is 1 to 255 data bytes
- <DataBuffer[x]> is Checksum

NOTE: Address information; Length byte is present

7.2.3.18 DataBuffer definition for <ProtocolID> = J2610

- Format not defined

7.2.3.19 Network and Transport Layer Details

The Pass-Thru Interface shall transparently handle the following Network and Transport Layer details:

For ISO 15765 channels, the Pass-Thru Interface shall automatically generate and add the PCI byte(s) to the outgoing CAN frames and automatically remove them from incoming CAN frames. The PCI byte(s) shall not appear in the array <DataBuffer> of the PASSTHRU_MSG structure. The Pass-Thru Interface shall also discard Flow Control frames and not add them to the receive queue.

For SAE J1850 PWM and SAE J1850 VPW channels, the Pass-Thru Interface shall automatically generate the CRC and add it to outgoing messages. The Pass-Thru Interface shall automatically remove the CRC from incoming messages. The CRC byte shall not appear in the array <DataBuffer> of the PASSTHRU_MSG structure. Additionally, for SAE J1850 PWM channels, <DataBuffer[0]> of the PASSTHRU_MSG structure (returned by **PassThruReadMsgs**) may not contain the corresponding byte from the actual message. Applications shall always ignore this byte. See Appendix A.2.16 for a more detailed description.

For ISO 9141 and ISO 14230 channels when the <Flags> have CHECKSUM_DISABLED is set to 0, the Pass-Thru Interface shall automatically generate the Checksum and add it to outgoing messages. Additionally, the Pass-Thru Interface shall automatically check and remove the Checksum from incoming messages. If an incoming message has an invalid Checksum it shall be discarded and not added to the receive queue.

For ISO 9141 and ISO 14230 channels when the <Flags> have CHECKSUM_DISABLED is set to 1, the Pass-Thru Interface shall neither add a Checksum to outgoing messages nor validate the Checksum on incoming messages. It is the application's responsibility to manage the Checksum as part of the data bytes in the array <DataBuffer> of the PASSTHRU_MSG structure.

For SAE J2610 channels, the Pass-Thru Device shall automatically check and remove the echoed bytes for outgoing messages when SCI_MODE is set to 1. The echoed bytes shall not appear in the array <DataBuffer> of the PASSTHRU_MSG structure. For the purpose of generating a timestamp, the last echoed byte (if applicable) is considered to be the last byte of the message.

7.2.4 Format Checks for Messages Passed to the API

The Pass-Thru Interface shall evaluate all PASSTHRU_MSG structures passed in from the application for correctness. Unless otherwise specified, the following items shall be checked and if an error is detected, the indicated action shall be taken:

- ERR_INVALID_MSG shall be returned when a <DataLength> violates the specified minimum or maximum for the corresponding Pass-Thru function call indicated in Figure 16.
- ERR_INVALID_MSG shall be returned when a bit in <TxFlags> is used that is not specified in the "Applicable <TxFlags>" column of Figure 16 or the use of a valid bit in TxFlags that is not applicable to the current configuration (such as setting the CAN_29BIT_ID on a CAN channel that is configured to be 11-bit only).
- ERR_INVALID_MSG shall be returned when a <DataLength> is larger than <DataBufferSize>.
- On CAN or ISO 15765 channels, ERR_INVALID_MSG shall be returned when inappropriate values in the bytes designated to be the CAN ID are non-zero. For example:
 - CAN ID greater than \$7FF for an 11-bit CAN message
 - CAN ID greater than \$1FFFFFFF for a 29-bit CAN message
- On ISO 14230 channels, ERR_INVALID_MSG shall be returned when a message contains a Format and/or Length byte that are invalid, incorrect, or that are inconsistent with the <DataLength>, except when passed from **PassThruStartMsgFilter**.
- On ISO 14230 channels, ERR_INVALID_MSG shall be returned when a Format byte that indicates Exception Mode (CARB, where A1=0 and A0=1) and is not \$48 or \$68, except when passed from **PassThruStartMsgFilter**.
- On SAE J1850 PWM channels, ERR_INVALID_MSG shall be returned when a Source Address (<DataBuffer[2]>) that is different from the Node Address (set via the **PassThruIoctl** function SET_CONFIG using the parameter NODE_ADDRESS), except when passed from **PassThruStartMsgFilter**.

NOTE: The <RxStatus>, <ExtraDataIndex>, and <Timestamp> elements in the PASSTHRU_MSG structure shall not be evaluated by the API.