SURFACE VEHICLE RECOMMENDED PRACTICE

J2534™-2 DEC2020

Issued 2006-03
Revised 2020-12

Superseding J2534-2 JAN2019

Optional Pass-Thru Features

RATIONALE

This update to API version 04.04 of the SAE J2534-2 specification is to add support for Ethernet NDIS protocol and the read voltage on an SAE J1962 pin functionality.

TABLE OF CONTENTS

1. SCOPE

SAE J2534-1 defines a standard vehicle network interface that can be used to reprogram emission-related control modules. However, there is a need to support vehicles prior to the 2004 model year, as well as non-emission related control modules.

The SAE J2534-2 document meets these needs by detailing extensions to API version 04.04 of the SAE J2534-1 specification. It is not required for an interface to be fully compliant with API version 04.04 of the SAE J2534-1 specification to implement some of the features specified in this document. Together, these extensions provide the framework for a common interface to protect the software investment of the vehicle OEMs and scan tool manufacturers.

Only the optional features will be described by this document and are based on the December 2004 publication of SAE J2534-1.

1.1 Purpose

Each section included in this document specifies features that extend API version 04.04 of the SAE J2534-1 specification. The specific feature operation will be described directly or reference another existing specification. In each case, the required calling structure, via the SAE J2534-1 API version 04.04, will be documented and coordinated by this document.

Extending the protocols supported by SAE J2534-1, this document adds two new types of ProtocolIDs.

1. ProtocolIDs with the suffix "_PS" for connecting to a vehicle, via the SAE J1962 connector, using the technique outlined in the section titled "Pin Selection."

2. Generic ProtocolIDs, with the suffixes "_CH1" through "_CH128" for protocols that terminate at a vendor-specific connector on the device. See the section titled "Access to Additional Channels."

1.2 Documentation Convention

For each protocol defined in this document:

- Unless explicitly specified otherwise, all SAE J2534-1 PassThru function calls will be supported.

- SAE J2534-1 ConnectFlags, RxStatus, TxFlags, and Indications are not supported unless explicitly specified.

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 +1 724-776-4970 (outside USA), www.sae.org.

SAE J1939	Serial Control and Communications Heavy Duty Vehicle Network - Top Level Document
SAE J1939-21	Data Link Layer
SAE J1939-81	Network Management
SAE J1962	Diagnostic Connector
SAE J2284-3	High-Speed CAN (HSC) for Vehicle Applications at 500 Kbps
SAE J2284-4	High-Speed CAN (HSC) for Vehicle Applications at 500 Kbps with CAN FD Data at 2 Mbps

SAE J2284-5	High-Speed CAN (HSC) for Vehicle Applications at 500 Kbps with CAN FD Data at 5 Mbps
SAE J2534-1	Recommended Practice for Pass-Thru Vehicle Programming
SAE J2740	General Motors UART Serial Data Communications
SAE J2809	Honda Diagnostic Serial Data Link Protocol - ABS/VSA System
SAE J2818	Keyword Protocol 1281
SAE J2819	TP2.0 Vehicle Diagnostic Protocol

2.1.2 IHS Markit

The current published manufacturer-specific documents may be acquired from the following URL: http://global.ihs.com/.

GMW3089	GMLAN Single Wire CAN Physical and Data Link Layers Specification Definitions
GMW3110	GMLAN Enhanced Diagnostics Test Mode Specifications
GMW3173	Architecture & Bus Wiring Requirements

2.1.3 ISO Documents

Copies of these documents are available online at http://webstore.ansi.org/.

ISO9141	Road Vehicles - Diagnostic Systems - CARB Requirements for Interchange of Digital Information
ISO9141	Road Vehicles - Diagnostic Systems - Requirements for Interchange of Digital Information
ISO9141-2	Road Vehicles - Diagnostic Systems - CARB Requirements for Interchange of Digital Information
ISO11898-1:2015	Road Vehicles - Controller Area Network (CAN) – Part 1: Data Link Layer and Physical Signalling
ISO11898-2	Road Vehicles - Controller Area Network (CAN) - Part 2: High-Speed Medium Access Unit
ISO11898-3	Road Vehicles - Controller Area Network (CAN) - Part 3: Low-Speed, Fault-Tolerant, Medium-Dependent Interface
ISO13400-2	Diagnostic Communication over Internet Protocol (DoIP) - Transport Protocol and Network Layer Services
ISO13400-3:2016	Diagnostic Communication over Internet Protocol (DoIP) - Wired Vehicle Interface Based on IEEE 802.3
ISO15765-2	Road Vehicles - Diagnostics Communication Over Controller Area Networks (DoCAN) - Transport Protocol and Network Layer Services
ISO15765-4	Road Vehicles - Diagnostics Communication Over Controller Area Networks (DoCAN) - Requirements for Emission-Related Systems

2.1.4 IEEE Publications

Available from IEEE Operations Center, 445 and 501 Hoes Lane, Piscataway, NJ 08854-4141, Tel: 732-981-0060, www.ieee.org.

IEEE 802.1Q	IEEE Standard for Local and Metropolitan Area Networks
IEEE 802.3	IEEE Standard for Ethernet

3. DEFINITIONS

3.1 Definitions for Analog Inputs

CHANNEL: An analog-to-digital channel.

SUBSYSTEM: A collection of similar channels.

3.2 Definitions for Honda DIAG-H

DIAG-H: Honda proprietary single wire UART-based diagnostic communication bus.

92 Hm/2: Honda proprietary diagnostic communication protocol.

3.3 Definitions for TP 2.0

TELEGRAM: A CAN message used on an established transport protocol connection.

3.4 Definitions for CAN FD

ACTIVE MESSAGE: Refers to a message that has started the transmission process on the network.

PASS-THRU DEVICE: Refers only to the physical hardware portion of the SAE J2534 interface, where the circuitry resides.

PASS-THRU INTERFACE: Refers to all components of an SAE J2534 interface, including both hardware and software.

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 active at some point in the future.

REGISTRY: A mechanism within Windows operating systems to handle hardware and software configuration information.

TERMINATE A MESSAGE: Refers to the process of stopping reception or transmission of a message by the pass-thru interface.

TRANSMITTED MESSAGE: Refers to a message that has been successfully placed on the network.

CAN MESSAGE FORMAT: Refers to the message format of the CAN frame which can be either CAN 2.0 (Classical CAN) or CAN FD.

CAN 2.0 STANDARD FRAME: Classical base frame format.

CAN 2.0 EXTENDED FRAME: Classical extended frame format.

CAN FD STANDARD FRAME: FD base frame format.

CAN FD EXTENDED FRAME: FD extended frame format.

3.5 Definitions for Ethernet

AUTO-MDI(X): Automatic medium-dependent interface crossover device that allows the Ethernet hardware to decide whether a cross-linked cable or a one-to-one connected (patch) cable is used for the connection between two Ethernet ports and which configures the physical layer transceiver (PHY) according to the type of cable in order to correctly connect Tx and Rx lines.

DOIP EDGE NODE: Host inside the vehicle, where an Ethernet activation line in accordance with ISO13400-3:2016 is terminated and where the link from the first node/host in the external network is terminated (from ISO13400-3:3016).

4. ACRONYMS

4.1 Acronyms for Single Wire CAN

GMLAN: General Motors Local Area Network

SWCAN: Single Wire CAN

4.2 Acronyms for GM UART

UART: Universal Asynchronous Receiver/Transmitter

4.3 Acronyms for Analog Inputs

A/D: Analog to Digital

4.4 Acronyms for UART Echo Byte Protocol

ABS: Anti-Lock Braking System

VSA: Vehicle Stability Assist

4.5 Acronyms for SAE J1939

BAM: Broadcast Announce Message

4.6 Acronyms for TP 2.0

TP: Transport Protocol

4.7 Acronyms for Fault-Tolerant CAN

FTCAN: Fault-Tolerant CAN (refer to ISO11898-3)

4.8 Acronyms for CAN FD and ISO15765 on CAN FD

API: Application Programming Interface

BRS: Bit Rate Switch

CAN: Controller Area Network

CAN FD: Flexable Datarate CAN (refer to ISO11898-1:2015)

CRC: Cyclic Redundancy Check

EDL: Extended Data Length

FDF: Flexible Data-Rate Format Indicator (also known as EDL)

IOCTL: Input/Output Control

CAN 2.0 : Refers to Classical CAN

4.9 Acronyms for ETHERNET_NDIS

IP: Internet Protocol

MAC: Media Access Control

NIC: Network Interface Card

NDIS: Network Driver Interface Specification

RNDIS: Remote Network Driver Specification

WINSOCK: Windows Sockets API

5. ENABLING SAE J2534-2 FEATURES AND ACCESSING MULTIPLE SAE J2534 DEVICES

To enable SAE J2534-2 features, call PassThruOpen with pName pointing to a NULL terminated string that begins with the ASCII characters “J2534-2:”

If pName is NULL, only SAE J2534-1 features shall be allowed.

If pName is not NULL and it does not begin with “J2534-2:” no assumptions can be made regarding the SAE J2534-1 or SAE J2534-2 features.

To use multiple devices from the same PC, the application shall append the vendor-specific device name to the string identified above.

Example:

“J2534-2:” to open the SAE J2534-2 default device as determined by the vendor.
“J2534-2:VendorX Device1” to open an SAE J2534-2 device identified as “VendorX Device1”

A device, to be compliant to the SAE J2534-2 specification, shall implement the Discovery Mechanism and shall support all the parameters specified for GET_DEVICE_INFO and GET_PROTOCOL_INFO IoctlIDs. See Section 25 for details.

6. PIN SELECTION

6.1 Scope of the Pin Selection Optional Feature

This section identifies the pin selection mechanism for ProtocolIDs that have the “_PS” suffix. The API extensions detailed here describe the method of specifying the pin(s) to which the selected protocol should be connected. While the API allows for all combinations of protocol/pin assignments, the actual combinations implemented are vendor specific.

6.2 Pass-Thru System Requirements

6.2.1 Pin Usage

The set of pins that can be switched is dependent on the set of optional protocols supported by the interface. A new set of SET_CONFIG parameters are defined for the application to specify the pins to be switched on various standard cables.

6.3 Win32 Application Programming Interface

6.3.1 API Functions - Overview

The new ProtocolIDs with “_PS” suffix indicates that the protocol physical layer is not connected to any pins on PassThruConnect. A new IOCTL configuration parameter is also added, that allows connection of a physical layer to specific pins on the specified cable.

For support of SAE J2534-1 protocols on different pins than those defined in SAE J2534-1, new ProtocolIDs are assigned to enable the pin-switching feature as defined in Table 1.

Table 1 - Protocol ID values (SAE J2534-1 defined)

Definition	Description
J1850VPW_PS	GM/Chrysler CLASS2
J1850PWM_PS	Ford SCP
ISO9141_PS	ISO9141 and ISO9141-2
ISO14230_PS	ISO14230-4 (Keyword Protocol 2000)
CAN_PS	Raw CAN (flow control not handled automatically by interface)
ISO15765_PS	ISO15765-2 (flow control enabled)
J2610_PS	SAE J2610 (Chrysler SCI)

NOTE: This is not an exhaustive list of _PS protocols. See Section 24 for a complete list.

As an example, in order to utilize a CAN channel connected to pins 3 and 11 (often used for medium-speed CAN network), the new CAN_PS ProtocolID is used in PassThruConnect. The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID. (The only exception to this is the mixed format frames feature on a CAN network.)

Note that SAE J2610 (Chrysler SCI) protocols are consolidated into a single new ProtocolID, J2610_PS. However, the SAE J2534-1 SCI protocols shall continue to be supported as defined in SAE J2534-1.

New SAE J2534-2 optional feature protocols use the ProtocolIDs defined in Section 24.

The interface must manage resource locking of pins that are in use. If an existing channel is using a pin that is requested by a PassThruIoctl call, the new request shall be rejected. The interface must also prevent a channel from being assigned to different pins on different cables at the same time.

Table 2 summarizes the changes to the SAE J2534-1 API functions.

Table 2 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	For all protocols with the "_PS" suffix defined above, no connection to any pins is made on the call to PassThruConnect.
PassThruIoctl	Add a new configuration parameter to allow selection of cable and pins.

6.3.2 API Functions - Detailed Information

6.3.2.1 PassThruConnect

When PassThruConnect is called, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG, J1962_PINS, J1939_PINS, or J1708_PINS is made.

There are two major differences from PassThruConnect usage in SAE J2534-1.

- The new ProtocolIDs (ending in "_PS") are not assigned pins upon connection. No transmission or reception on a channel is possible until the pins are assigned using the IOCTL parameter J1962_PINS, J1939_PINS, or J1708_PINS.
- If the interface supports multiple instances of a given protocol device, the "_PS" ProtocolIDs can be opened multiple times. For example, if an interface device supports two independent CAN controllers, a program could open CAN_PS and request pins 3 and 11, then open CAN_PS again and request pins 1 and 12.

Devices that are not physically capable of supporting the requested protocol shall return ERR_NOT_SUPPORTED.

Example 1: Devices that do not support any transceivers of a particular type (e.g., SW_CAN_PS).

Example 2: Devices with only one controller of a particular type (e.g., one physical CAN controller) shall disallow opening that "_PS" ProtocolID multiple times (since it could never satisfy the second request).

For further information regarding failure conditions, see 6.3.2.7.

6.3.2.2 PassThruDisconnect

This API function shall return the SAE J1962 pins to the default state, as specified by SAE J2534-1. The SAE J1939 and SAE J1708 pins shall be returned to a high-impedance state.

6.3.2.3 PassThruReadMsgs

If pins have not been assigned, ERR_PIN_INVALID shall be returned if this function is called.

6.3.2.4 PassThruWriteMsgs

If pins have not been assigned, ERR_PIN_INVALID shall be returned if this function is called.

6.3.2.5 PassThruStartPeriodicMsg

If pins have not been assigned, ERR_PIN_INVALID shall be returned if this function is called.

6.3.2.6 PassThruStartMsgFilter

If pins have not been assigned, ERR_PIN_INVALID shall be returned if this function is called.

6.3.2.7 PassThruIoctl

Pins are assigned via SET_CONFIG with the J1962_PINS, J1939_PINS, or J1708_PINS parameters. See 6.3.3.2 for details regarding this parameter. All other PassThruIoctl functions for this Channel ID shall return ERR_PIN_INVALID until the pin assignment is successfully completed.

The application must call PassThruDisconnect before attempting to reassign pins.

6.3.3 IOCTL

6.3.3.1 GET_CONFIG

The following configuration parameters are supported:

J1962_PINS

J1939_PINS

J1708_PINS

See Table 3 for more details.

6.3.3.2 SET_CONFIG

The configuration parameters J1962_PINS, J1939_PINS, and J1708_PINS are added as defined in Table 3. For the protocol channel referenced in the ChannelID parameter of the SET_CONFIG call, these parameters specify which connector and pin or pins this protocol's physical layer is to be connected to. The act of setting this parameter causes the connection of the protocol physical layer to the specified pins. At the vendors discretion, detection of the correct cable can be used to reject an invalid request with a return code of ERR_INVALID_IOCTL_VALUE.

NOTE: Unpredictable results may occur if cable detection is not implemented and an incorrect cable is connected.

Table 3 - IOCTL GET_CONFIG/SET_CONFIG parameter details

Parameter	Valid Values for Parameter	Default Value (decimal)	Description
J1962_PINS	0x0000PPSS where: PP: 0x00 - 0x10 SS: 0x00 - 0x10 PP ! = SS, except when set to 0x0000 Exclude pins 4, 5, & 16	0	For a channel of any protocol type, this selects the SAE J1962 pin, or pair of pins, onto which the physical layer is to be switched. NOTE: A value of 0 can never be set. Reading a value of 0 indicates that pin selection has not been performed. PP is the pin number for the primary signal, e.g., ISOK-line, CAN-H, +ve, SCI Tx, DIAG-H, SAE J1850+, etc. SS is the pin number for the secondary signal, where a secondary signal is present, e.g., ISOL-line, CAN-L, -ve, SCI Rx, SAE J1850-, etc. SS shall equal 0x00 if no secondary pin is required or enabled.
J1939_PINS	0x0000PPSS where: PP: 0x00 - 0x09 SS: 0x00 - 0x09 PP ! = SS, except when set to 0x0000 Exclude pins 1 & 2		For a channel of any protocol type, this selects the SAE J1939-13 pin, or pair of pins, onto which the physical layer is to be switched.

SAE J1939-13 PIN	Function	PIN Number
A	Ground	1
B	Battery+	2
C	CAN_H	3
D	CAN_L	4
E	Shield/NC	5
F	J1708+	6
G	J1708-	7
H	-	8
J	-	9

			NOTE: A value of 0 can never be set. Reading a value of 0 indicates that pin selection has not been performed. PP is the pin number for the primary signal, e.g., ISOK-line, CAN-H, +ve, SCI Tx, DIAG-H, SAE J1850+, etc. SS is the pin number for the secondary signal, where a secondary signal is present, e.g., ISOL-line, CAN-L, -ve, SCI Rx, SAE J1850-, etc. SS shall equal 0x00 if no secondary pin is required or enabled.
J1708_PINS	0x0000PPSS where: PP: 0x00 - 0x09 SS: 0x00 - 0x09 PP ! = SS, except when set to 0x0000 Exclude pins 3 & 5		For a channel of any protocol type, this selects the pin or pair of pins on the SAE J1708 6 pin Deutch connector, onto which the physical layer is to be switched. SAE J1708 PIN / Function / PIN Number: A / J1708+ / 1 B / J1708- / 2 C / Battery+ / 3 D / - / 4 E / Ground / 5 F / - / 6 NOTE: A value of 0 can never be set. Reading a value of 0 indicates that pin selection has not been performed. PP is the pin number for the primary signal, e.g., ISOK-line, CAN-H, +ve, SCI Tx, DIAG-H, SAE J1850+, etc. SS is the pin number for the secondary signal, where a secondary signal is present, e.g., ISOL-line, CAN-L, -ve, SCI Rx, SAE J1850-, etc. SS shall equal 0x00 if no secondary pin is required or enabled.

SAE J1708 PIN	Function	PIN Number
A	J1708+	1
B	J1708-	2
C	Battery+	3
D	-	4
E	Ground	5
F	-	6

For existing SAE J2534-1 base protocols, the physical interface is connected to the SAE J1962 pins automatically when PassThruConnect is called, as defined in SAE J2534-1. The J1962_PINS parameter does not need to be supported for SAE J2534-1 base protocols, but the use of these protocols will impact the pins that are available for the _PS protocols.

For protocols with the "_PS" suffix, at least one of the J1962_PINS, J1939_PINS, or J1708_PINS parameters must be supported, and the default value of the parameter shall be 0x0000. Certain combinations of SAE J1962, SAE J1939, or SAE J1708 pins could result in vehicle or interface damage. ERR_PIN_INVALID is returned for combinations that are outside of the defined range. The application programmer is responsible for determining if the potential exists for damage to the vehicle and taking the necessary steps to prevent it.

Only one SET_CONFIG, with the parameter J1962_PINS, J1939_PINS, or J1708_PINS can be performed for a given Channel ID. After a successful call to SET_CONFIG, if a second call for a given Channel ID is attempted, ERR_CHANNEL_IN_USE shall be returned. PassThruDisconnect is required before an alternate cable or pin selection may be attempted.

For the J1962_PINS, J1939_PINS, and J1708_PINS parameters, the following error handling shall be applied:

- PassThruIoctl requests unsupported pin combination.

  PassThruIoctl, SET_CONFIG, is called, requesting a value of the J1962_PINS, J1939_PINS, or J1708_PINS parameter that are valid but can't be supported by the hardware or having the potential of damaging the interface device.

  Result: PassThruIoctl returns ERR_NOT_SUPPORTED.

- PassThruIoctl requests pin combination that would cause conflict.

  PassThruIoctl, SET_CONFIG, is called, setting the J1962_PINS, J1939_PINS, or J1708_PINS parameter to a value that causes a pin conflict with an existing channel.

  Result: PassThruIoctl returns ERR_PIN_IN_USE.

6.4 Discovery Support

The device shall report support for pin switching and associated parameters through the discovery mechanism defined in Section 25.

7. ACCESS TO ADDITIONAL CHANNELS

This section defines an optional mechanism available to initiate and use multiple channels of the same protocol, if vendor hardware provides the support. The channels addressed this way are not be tied to pins on the SAE J1962 connector. What channel appears at what pins on the interface device will depend on the vendor configuration. The channels are typically accessed using custom cable built to hardware vendor's specification of connection details.

For example, a particular vendor's hardware may support four channels of dual-wire high-speed CAN. These additional channels will be available as separate ProtocolIDs defined by SAE J2534-2. This document will allocate 128 predefined ProtocolIDs for each protocol to target a possible maximum of 128 channels of each protocol. The ProtocolIDs will follow the following format:

CAN_CH1
CAN_CH2
…………..
CAN_CH128

This scheme of providing additional ProtocolIDs will apply to both SAE J2534-1 and SAE J2534-2 defined protocols. A complete list of ProtocolIDs for multiple channel access can be found in Section 24.

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID. (The only exception to this is the mixed format frames feature on a CAN network.)

Based on vendor implementation protocols initiated using J2534-1, _PS, or _CHx ProtocolIDs may use the same hardware resources. When there is a resource conflict, a call to PassThruConnect will return ERR_RESOURCE_IN_USE. In the case where the requested resource does not exist, ERR_NOT_SUPPORTED is returned.

7.1 Discovery Support

The device shall report support for multiple channels through the discovery mechanism defined in Section 25.

8. MIXED FORMAT FRAMES ON A CAN NETWORK

8.1 Scope of the Mixed Format Frames on a CAN Network Optional Feature

This section details the extensions to SAE J2534-1 that will allow the simultaneous reception and transmission of ISO15765 messages and unformatted CAN frames on an ISO15765 channel. The ProtocolID (in the PASSTHRU_MSG structure) will be used to identify the format of the associated message. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed.

8.2 Win32 Application Programming Interface

8.2.1 API Functions - Overview

Connecting to an ISO15765 channel and then setting the IOCTL configuration parameter CAN_MIXED_FORMAT to CAN_MIXED_FORMAT_ON shall allow messages to be processed as either unformatted CAN frames or as ISO15765 messages. Additionally, setting the IOCTL configuration parameter CAN_MIXED_FORMAT to CAN_MIXED_FORMAT_ALL_FRAMES shall allow the individual frames of an ISO15765 message (including flow control) to also be processed in a parallel path as an unformatted CAN frame. Unformatted CAN frames shall have the ProtocolID in the PASSTHRU_MSG structure set to an appropriate CAN Protocol ID and shall follow the requirements and restrictions for that protocol. ISO15765 messages shall have the ProtocolID in the PASSTHRU_MSG structure set to an appropriate ISO15765 Protocol ID and shall follow the requirements and restrictions for that protocol. For example, CAN is associated with ISO15765, SW_CAN_PS is associated with SW_ISO15765_PS, CAN_PS is associated with ISO15765_PS, etc.

When transmitting a message or starting a periodic message, the Protocol ID in the PASSTHRU_MSG structure shall be used to identify how the associated message shall be processed. If the configuration setting LOOPBACK is set to ON, then transmitted messages (including flow control) shall be processed in the same manner as received messages. As with SAE J2534-1, messages will be sent one at a time. This makes it possible for an ISO15765 message to block other messages until transmission is complete, including unformatted CAN frames.

The function PassThruStartMsgFilter shall be used to identify ISO15765 messages as well as unformatted CAN frames. A received message that matches a FLOW_CONTROL_FILTER shall be processed as an ISO15765 message and shall have the appropriate ProtocolID in the PASSTHRU_MSG structure before it is added to the receive queue. Additionally, based upon the setting of CAN_MIXED_FORMAT, the CAN frames may also be subject to the PASS_FILTERs and BLOCK_FILTERs, where the ProtocolID in the PASSTHRU_MSG structure shall be set to reflect an unformatted CAN frame before it is added to the receive queue. Figures 1 and 2 outline how received messages are processed for the various settings of CAN_MIXED_FORMAT.

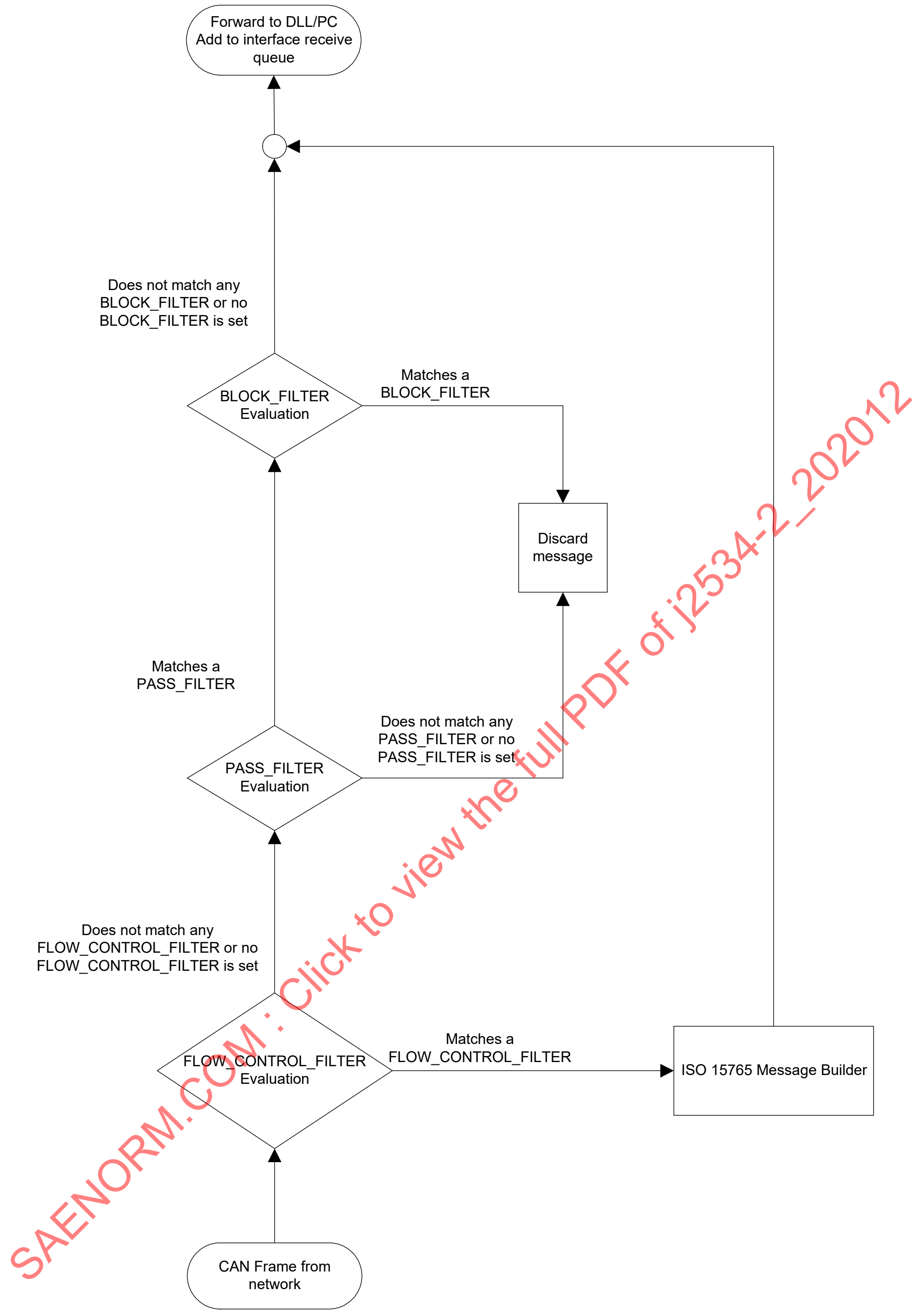

Forward to DLL/PC
Add to interface receive queue
Does not match any BLOCK_FILTER or no BLOCK_FILTER is set
Matches a BLOCK_FILTER
BLOCK_FILTER Evaluation
Discard message
Matches a PASS_FILTER
Does not match any PASS_FILTER or no PASS_FILTER is set
PASS_FILTER Evaluation
Does not match any FLOW_CONTROL_FILTER or no FLOW_CONTROL_FILTER is set
FLOW_CONTROL_FILTER Evaluation
Matches a FLOW_CONTROL_FILTER
ISO 15765 Message Builder
CAN Frame from network

Figure 1 - Processing of received messages when CAN_MIXED_FORMAT is CAN_MIXED_FORMAT_ON

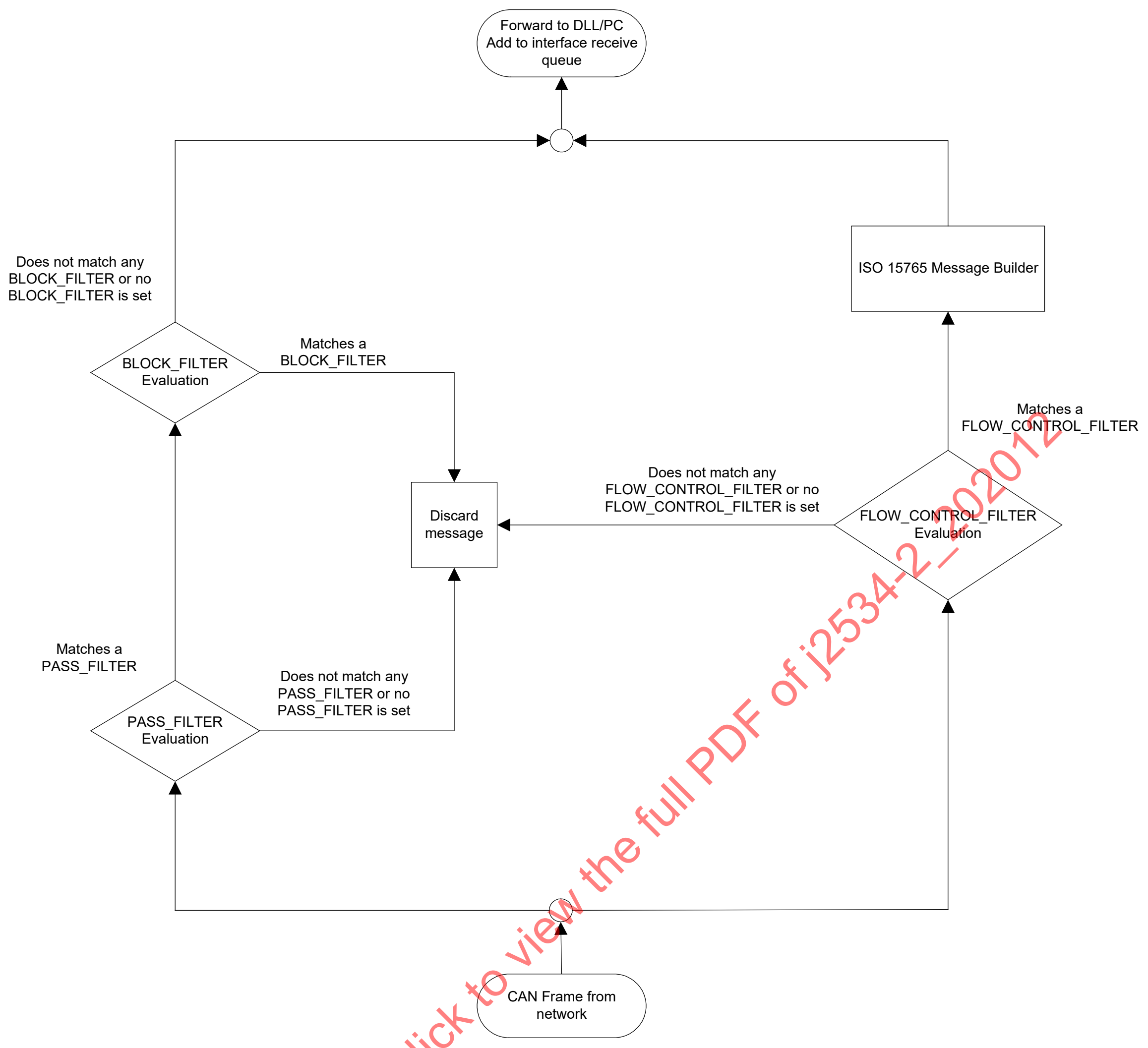

Forward to DLL/PC Add to interface receive queue
ISO 15765 Message Builder
Does not match any BLOCK_FILTER or no BLOCK_FILTER is set
Matches a BLOCK_FILTER
BLOCK_FILTER Evaluation
Matches a FLOW_CONTROL_FILTER
Discard message
Does not match any FLOW_CONTROL_FILTER or no FLOW_CONTROL_FILTER is set
FLOW_CONTROL_FILTER Evaluation
Matches a PASS_FILTER
Does not match any PASS_FILTER or no PASS_FILTER is set
PASS_FILTER Evaluation
CAN Frame from network

Figure 2 - Processing of received messages when CAN_MIXED_FORMAT is CAN_MIXED_FORMAT_ALL_FRAMES

If the optional feature is not supported on the current ISO15765 channel, the call to get or set the IOCTL configuration parameter CAN_MIXED_FORMAT shall return the value ERR_NOT_SUPPORTED.

Table 4 summarizes the changes to the SAE J2534-1 API functions.

Table 4 - SAE J2534 API functions

Function	Description of Change
PassThruStartMsgFilter	Increase the minimum number of FLOW_CONTROL_FILTERs to 64 and PASS_FILTERs/BLOCK_FILTERs to a total of 10.
PassThruIoctl	Add a new configuration parameter.

8.2.2 API Functions - Detailed Information

8.2.2.1 PassThruReadMsgs

There is no change to this function. However, only ISO15765 channels will allow the IOCTL configuration parameter CAN_MIXED_FORMAT to be either CAN_MIXED_FORMAT_ON or CAN_MIXED_FORMAT_ALL_FRAMES. In these cases, the ProtocolID in the PASSTHRU_MSG structure shall reflect either an unformatted CAN frame (e.g., CAN, SW_CAN_PS, etc.) or an ISO15765 message (e.g., ISO15765, SW_ISO15765_PS, etc.). Consult the appropriate SAE J2534 document for the requirements, restrictions, and error conditions for the specific protocol.

Additionally, each time the IOCTL configuration parameter CAN_MIXED_FORMAT is set, the receive queue shall be cleared.

8.2.2.2 PassThruWriteMsgs

There is no change to this function. However, only ISO15765 channels will allow the IOCTL configuration parameter CAN_MIXED_FORMAT to be either CAN_MIXED_FORMAT_ON or CAN_MIXED_FORMAT_ALL_FRAMES. In these cases, the ProtocolID in the PASSTHRU_MSG structure shall reflect either an unformatted CAN frame (e.g., CAN, SW_CAN_PS, etc.) or an ISO15765 message (e.g., ISO15765, SW_ISO15765_PS, etc.). Consult the appropriate SAE J2534 document for the requirements, restrictions, and error conditions for the specific protocol. This function will return a value of ERR_MSG_PROTOCOL_ID if the IOCTL configuration parameter CAN_MIXED_FORMAT is set to CAN_MIXED_FORMAT_OFF and the ProtocolID reflects an unformatted CAN frame.

Additionally, each time the IOCTL configuration parameter CAN_MIXED_FORMAT is set, the transmit queue shall be cleared.

The SAE J2534 device will not protect the user from writing an unformatted CAN frame that may be interpreted as a valid ISO15765 frame. The consequences of this action are undefined and may disrupt ISO15765 communications taking place on the network.

8.2.2.3 PassThruStartPeriodicMsg

There is no change to this function. However, only ISO15765 channels will allow the IOCTL configuration parameter CAN_MIXED_FORMAT to be either CAN_MIXED_FORMAT_ON or CAN_MIXED_FORMAT_ALL_FRAMES. In these cases, the ProtocolID in the PASSTHRU_MSG structure shall reflect either an unformatted CAN frame (e.g., CAN, SW_CAN_PS, etc.) or an ISO15765 message (e.g., ISO15765, SW_ISO15765_PS, etc.). Consult the appropriate SAE J2534 document for the requirements, restrictions, and error conditions for the specific protocol. This function will return a value of ERR_MSG_PROTOCOL_ID if the IOCTL configuration parameter CAN_MIXED_FORMAT is set to CAN_MIXED_FORMAT_OFF and the ProtocolID reflects an unformatted CAN frame.

Additionally, each time the IOCTL configuration parameter CAN_MIXED_FORMAT is set, periodic messages with ProtocolID of CAN shall be deleted.

The SAE J2534 device will not protect the user from writing a CAN message that may be interpreted as a valid ISO15765 frame. The consequences of this action are undefined and may disrupt ISO15765 communications taking place on the network.

8.2.2.4 PassThruStartMsgFilter

Each ISO15765 channel shall support a minimum of 64 FLOW_CONTROL_FILTERs, as well as ten PASS_FILTERs/BLOCK_FILTERs.

This function shall be used to identify ISO15765 messages, as well as unformatted CAN frames. A received message that matches a FLOW_CONTROL_FILTER shall be processed as an ISO15765 message and shall have the appropriate ProtocolID in the PASSTHRU_MSG structure before it is added to the receive queue. Additionally, based upon the setting of CAN_MIXED_FORMAT, the CAN frames may also be subject to the PASS_FILTERs and BLOCK_FILTERs, where the ProtocolID in the PASSTHRU_MSG structure shall be set to reflect an unformatted CAN frame before it is added to the receive queue.

Only ISO15765 channels will allow the IOCTL configuration parameter CAN_MIXED_FORMAT to be set to either CAN_MIXED_FORMAT_ON or CAN_MIXED_FORMAT_ALL_FRAMES. In these cases, the ProtocolID in the PASSTHRU_MSG structure must reflect an unformatted CAN frame (e.g., CAN, SW_CAN_PS, etc.) for a PASS_FILTER or a BLOCK_FILTER and an ISO15765 message (e.g., ISO15765, SW_ISO15765_PS, etc.) for a FLOW_CONTROL_FILTER. Consult the appropriate SAE J2534 document for the requirements, restrictions, and error conditions for the specific protocol and filter type. This function will return a value of ERR_MSG_PROTOCOL_ID if the ProtocolID is not appropriate for the type of filter being started.

Additionally, each time the IOCTL configuration parameter CAN_MIXED_FORMAT is set, all PASS_FILTERs and BLOCK_FILTERs shall be deleted.

8.2.3 IOCTL Section

If this feature is not supported on the current ISO15765 channel, the call to get or set the IOCTL configuration parameter CAN_MIXED_FORMAT shall return the value ERR_NOT_SUPPORTED.

8.2.3.1 GET_CONFIG

The configuration parameter CAN_MIXED_FORMAT has been defined but it is only applicable to ISO15765 channels. See Table 5 for more details.

8.2.3.2 SET_CONFIG

The configuration parameter CAN_MIXED_FORMAT has been defined but it is only applicable to ISO15765 channels. CAN_MIXED_FORMAT configuration parameter is defined in Table 5.

Table 5 - IOCTL GET_CONFIG/SET_CONFIG parameter details

Parameter	Valid Values for Parameter	Default Value	Description
CAN_MIXED_FORMAT	0 (CAN_MIXED_FORMAT_OFF) 1 (CAN_MIXED_FORMAT_ON) 2 (CAN_MIXED_FORMAT_ALL_FRAMES)	0 (CAN_MIXED_FORMAT_OFF)	For ISO15765 channels only, this enables the transmission and reception of messages with the ProtocolID of ISO15765 or unformatted CAN frames. FLOW_CONTROL_FILTERs will identify messages with the ProtocolID that reflects an ISO15765 message, while PASS_FILTERs and BLOCK_FILTERs will identify messages with the ProtocolID that reflects an unformatted CAN frame. Any time this parameter is set, the transmit and receive queues shall be cleared, all PASS_FILTERs and BLOCK_FILTERs shall be deleted, and periodic messages whose ProtocolID reflects an unformatted CAN frame shall be deleted. 0 = Messages will be treated as ISO15765 only. 1 = Messages will be treated as either ISO15765 or an unformatted CAN frame. 2 = Messages will be treated as ISO15765, an unformatted CAN frame, or both.

8.2.4 Message Structure

8.2.5 Elements

There is no change to any of the elements, but it should be noted that the ProtocolID in the PASSTHRU_MSG structure shall identify the message type (e.g., ISO15765, CAN, SW_ISO15765_PS, SW_CAN_PS, etc.). Consult the appropriate SAE J2534 document for the requirements, restrictions, and error conditions for the specific protocol.

8.3 Discovery Support

The device shall report support for mixed format frames on the CAN network through the discovery mechanism defined in Section 25.

9. SINGLE WIRE CAN

9.1 Scope of the Single Wire CAN Optional Feature

Information contained in this section will define extensions to a compliant SAE J2534-1 interface to support single wire CAN.

9.2 Pass-Thru System Requirements

9.2.1 Pin Usage

Single wire CAN is connected to pin 1 of the SAE J1962 connector.

Note that when PassThruConnect is called, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG, J1962_PINS is made.

9.3 Win32 Application Programming Interface

9.3.1 API Functions - Overview

Information contained in this section is intended to define the API resources required to incorporate an optional protocol channel. This channel, identified as single wire CAN (SWCAN), will require hardware and software API support to fully implement this feature.

The details on the physical implementation of SWCAN are defined in GMW3089, titled “GMLAN Single Wire CAN Physical and Data Link Layers Specification.”

This section outlines the requirements for providing an interface channel supporting this protocol.

At a high level, a new ProtocolID has been defined to indicate the use of the single wire CAN physical layer. Unless indicated otherwise, SAE J2534-1 definitions of requirements, restrictions, and error conditions for CAN and ISO15765 protocols will apply to single wire CAN channels. Additionally, flags required to support high-voltage wakeup and high/normal speed are defined as required. The device is required to support high-voltage wakeup and time-critical data rate changes defined in SWCAN specification. This section details the API resources required to enable use of the SAE J2534 API as applied to SWCAN.

If this feature is not supported an error code, ERR_NOT_SUPPORTED, will be returned by the call PassThruConnect.

The IOCTL and Configuration settings related to SW_CAN speed change and load resistor control are designed for use when the SAE J2534-2 hardware interface is used in either of two modes:

1. As a hardware interface for a flash-programming application.

2. As a hardware interface for any other application that could be used to monitor traffic during a flash-programming event performed by another test tool on the bus.

GMW3110 requires a maximum transition time of 30 ms to switch between bus speeds. Depending on the individual SAE J2534-2 hardware interface and its associated PC communications interface and application processing speed, the 30 ms transition time may or may not be met using speed transition logic that is embedded in the PC application. The IOCTL and configuration settings allow for this speed change logic to be controlled by either the application or the SAE J2534-2 hardware interface itself.

Since not all application/SAE J2534-2 interface combinations can control speed transition timing within the GMW3110 specification, a method is required to allow the SAE J2534-2 hardware interface to automatically switch speeds while performing a flash-programming operation. The following example illustrates this sequence:

Assume all settings are in the default mode.

1. The application sets the configuration of the SW_CAN_RES_SWITCH parameter - AUTO_RESISTOR.

2. The application sets the configuration of SW_CAN_SPEEDCHANGE_ENABLE - ENABLE_SPDCHANGE.

3. The application sends the correct GMW 3110 HS programming message sequence (0xA5 0x02, 0xA5 0x03).

4. The SAE J2534-2 hardware interface monitors the (0xA5 0x02, 0xA5 0x03) sequence and automatically switches in the load resistor, changes the SWCAN transceiver mode, and reconfigures the CAN controller. Note that this must be accomplished within 30 ms of the 0xA5 0x03 frame. A SW_CAN_HS_RX indication will be generated upon completion of this transition.

5. Upon completion of the flash programming event, the application transmits the return to normal mode command (mode $20). A SW_CAN_NS_RX indication will be generated upon completion of this transition.

6. The SAE J2534-2 hardware interface responds to the return to normal mode command, which reconfigures the CAN controller, changes the transceiver mode, and disconnects the load resistor. Note that this must be accomplished within 30 ms of the transmission of the return to normal mode frame.

There are also times when the SAE J2534-2 hardware interface could be used with a separate application to monitor a flash-programming event being performed by another test tool. In this case, the SAE J2534-2 hardware interface would need to be able to perform speed transitions within the GMW3110 specified limit. An SAE J2534-2 hardware interface used in this manner should not connect its load resistor, as this functionality should already be contained in the test tool performing the flash-programming event.

The following example illustrates the setup for monitoring such an event:

Assume all settings are in the default mode.

1. The application sets the configuration of SW_CAN_SPEEDCHANGE_ENABLE - ENABLE_SPDCHANGE.

2. The SAE J2534-2 hardware interface monitors the (0xA5 0x02, 0xA5 0x03) sequence and automatically changes the SWCAN transceiver mode and reconfigures the CAN controller. (It does not switch in the load resistor.) Note that this must be accomplished within 30 ms of the 0xA5 0x03 frame. A SW_CAN_HS_RX indication will be generated upon completion of this transition.

3. Upon completion of the flash programming event, the SAE J2534-2 hardware interface receives the return to normal mode command (mode $20). A SW_CAN_NS_RX indication will be generated upon completion of this transition.

4. The SAE J2534-2 hardware interface responds to the return to normal mode command, which reconfigures the CAN controller and changes the transceiver mode. Note that this must be accomplished within 30 ms of the transmission of the return to normal mode frame.

Although the option of manual load resistor activation (CONNECT_LOAD_RESISTOR) is not shown in any of the above examples, this capability was included to accommodate special test applications that might require this feature.

Table 6 summarizes the changes to the SAE J2534-1 API functions.

Table 6 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.
PassThruIoctl	Added GET_CONFIG and SET_CONFIG parameters. Added new IOCTLs to support SWCAN capability.

9.3.2 API Functions - Detailed Information

9.3.2.1 PassThruConnect

When PassThruConnect is called with either the SW_CAN_PS or SW_ISO15765_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

9.3.2.2 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 7. The actual value is defined in Section 24.

Table 7 - ProtocolID descriptions

Definition	Description
SW_CAN_PS	Raw single wire CAN messages with pin selection.
SW_ISO15765_PS	Single wire CAN adhering to ISO15765-2 flow control with pin selection.
SW_CAN_CHx	Additional channels of raw single wire CAN messages.
SW_ISO15765_CHx	Additional channels of single wire CAN adhering to ISO15765-2 flow control.

9.3.2.3 PassThruReadMsgs

The RxStatus indications identified in 9.4.1.1 will be received for both commanded and automatic speed changes.

9.3.3 IOCTL Section

The Protocol IDs SW_ISO15765_PS and SW_ISO15765_CHx support the same IOCTL IDs as defined in the SAE J2534-1 specification for ISO15765. SW_CAN_PS and SW_CAN_CHx support the same IOCTL IDs as defined in the SAE J2534-1 specification for CAN. This section details additional IOCTLs defined in this document.

Table 8 provides the details of the IOCTLs available through PassThruIoctl function.

Table 8 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr Represents	Purpose
SW_CAN_HS	NULL pointer	NULL pointer	Initiates the transition of the SW_CAN channel from SW_CAN_NS (normal speed) mode to SW_CAN_HS (high speed) mode. This transition includes resetting the SW_CAN transceiver mode to the HS setting and changing the SW_CAN controller configuration to the SW_CAN_HS_DATA_RATE.
SW_CAN_NS	NULL pointer	NULL pointer	Initiates the transition of the SW_CAN channel from SW_CAN_HS (high speed) mode to SW_CAN_NS (normal speed) mode. This transition includes resetting the SW_CAN transceiver mode to the normal mode setting and changing the SW_CAN controller configuration to the DATA_RATE.

9.3.3.1 GET_CONFIG

Support three new parameters added to GET_CONFIG. See 9.3.3.2 and Table 9 for more details.

9.3.3.2 SET_CONFIG

The Protocol IDs SW_ISO15765_PS and SW_ISO15765_CHx support the same Get/Set parameter IDs as defined in SAE J2534-1 specification for ISO15765. SW_CAN_PS and SW_CAN_CHx support the same Get/Set parameter IDs as defined in the SAE J2534-1 specification for CAN. This section details additional IOCTLs defined in this document.

SW_CAN_HS mode is to be used exclusively for the reprogramming of devices. It requires the coordinated and selective configuration of three pieces of hardware: the load resistor, the SW_CAN transceiver, and the SW_CAN controller. Specific information regarding each piece is as follows:

1. A load resistor is connected to the SW_CAN bus within the tool, which helps compensate for reduced bit times by decreasing the active to passive transition times. To prevent excessive electrical loading of the SW_CAN bus, this feature shall only be activated by the programming device. All other devices or tools used to monitor high-speed communication shall remain in the normal impedance state.

2. The SW_CAN transceiver is placed into a mode which also compensates for the reduced bit times by disabling waveshaping and decreasing the passive to active transition times.

3. The CAN controller is configured to provide the appropriate high-speed data rate.

Table 9 - IOCTL GET_CONFIG/SET_CONFIG parameter details

Parameter	Valid Values for Parameter	Default Value (Decimal)	Description
SW_CAN_HS_DATA_RATE	5-100000 (Some transceivers support 100k)	83333	The data rate to be used in response to a call to SW_CAN_HS IOCTL.
SW_CAN_SPEEDCHANGE_ENABLE	0 (DISABLE_SPDCHANGE) 1 (ENABLE_SPDCHANGE)	0	Control the behavior of the SAE J2534 device in response to speed change SW_CAN messages 0 = Ignore all bus speed transition messages on the bus. 1 = Process transmitted and received bus speed transition messages (refer to GMW3110 Section 10.17.5.2).
SW_CAN_RES_SWITCH	0 (DISCONNECT_RESISTOR) 1 (CONNECT_RESISTOR) 2 (AUTO_ RESISTOR)	0	Control Load Resistor switching 0 = Default value. Disable automatic switching and disconnect load resistor. 1 = Disable automatic switching and connect load resistor. 2 = Automatically switch in the load resistor when transitioning to high speed (and switch off the load resistor while transitioning back to normal speed).

9.3.3.3 SW_CAN_HS

The IoctlID value of SW_CAN_HS is used to initiate a transition of the single wire CAN bus to high speed mode. A successful transition will be noted by a SW_CAN_HS_RX indication. The speed transitioned to is the value of the SW_CAN_HS_DATA_RATE parameter. Parameter definition for SW_CAN_HS is defined in Table 10.

Table 10 - SW_CAN_HS details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect.
IoctlID	Is set to SW_CAN_HS.
InputPtr	Is a NULL pointer, as this parameter is not used.
OutputPtr	Is a NULL pointer, as this parameter is not used.

9.3.3.4 SW_CAN_NS

The IoctlID value of SW_CAN_NS is used to initiate a transition of the single wire CAN bus to normal speed mode. A successful transition will be noted by a SW_CAN_NS_RX indication. The speed transitioned to is the value of the DATA_RATE parameter. Parameter definition for SW_CAN_NS is detailed in Table 11.

Table 11 - SW_CAN_NS details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect.
IoctlID	Is set to SW_CAN_NS.
InputPtr	NULL pointer.
OutputPtr	NULL pointer.

9.4 Message Structure

9.4.1 Elements

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID. (The only exception to this is the mixed format frames feature on a CAN network.)

9.4.1.1 RxStatus

This section defines RxStatus bits in addition to the ones defined for CAN/ISO15765 sections of the SAE J2534-1 specification.

Definitions for RxStatus bits are defined in Table 12.

Table 12 - RxStatus bit definitions

Definition	RxStatus Bit(s)	Description	Value
SW_CAN_NS_RX	18	Indicates that the single wire CAN bus has transitioned to normal speed. All communication after this event will occur in normal-speed mode. The message data in this message is undefined.	0 = no event 1 = transition to normal speed
SW_CAN_HS_RX	17	Indicates that the single wire CAN bus has transitioned to high speed. All communication after this event will occur in high-speed mode. The message data in this message is undefined.	0 = no event 1 = transition to high speed
SW_CAN_HV_RX	16	Indicates that the single wire CAN message received was high-voltage message.	0 = normal message 1 = high-voltage message

9.4.1.2 TxFlags

This section defines TxStatus bits are in addition to the ones defined for CAN/ISO15765 sections of SAE J2534-1 specification.

Table 13 defines the TxFlags bit definition.

Table 13 - TxFlags bit definitions

Definition	TxFlags Bit(s)	Description	Value
SW_CAN_HV_TX	10	Indicates that the single wire CAN message should be transmitted as a high-voltage message. Simultaneously transmitting in high voltage and high speed mode will result in undefined behavior.	0 = normal message 1 = high-voltage message

9.5 Discovery Support

The device shall report support for single wire CAN network and associated parameters through the discovery mechanism defined in Section 25.

10. ANALOG INPUTS

10.1 Scope of the Analog Inputs Optional Feature

This section details the extensions to SAE J2534-1 that define the common method of supporting analog input channels. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed.

This standard does not specify the timing between the same subsystem or different subsystems. Depending on the device, all the active channel readings could be made simultaneously, or could be spaced out in time.

10.2 Pass-Thru System Requirements

10.2.1 Analog Inputs

Information contained in this section will define extensions to a compliant SAE J2534-1 interface. This section specifically defines the common method of supporting analog input channels.

10.2.2 Simultaneous Communication on Multiple Protocols

The operation of the A/D subsystem shall be independent of the operation of the communications protocols. The interface must support simultaneous collection of analog data and communication on multiple protocols as specified in SAE J2534-1 and SAE J2534-2.

10.3 Win32 Application Programming Interface

10.3.1 API Functions - Overview

Information contained in this section is intended to define the API resources required to incorporate analog input channels on a PassThru device. Analog inputs will require hardware and software API support to fully implement this feature. It allows compliant devices to acquire analog data in an efficient and deterministic manner. Physical connection of the SAE J2534-2 interface to the vehicle is defined by the interface manufacturer.

This new feature allows an application to open a connection to an analog subsystem via PassThruConnect. The subsystem parameters can be set via the GET_CONFIG/SET_CONFIG IOCTLs. The actual analog readings can be obtained with PassThruReadMsgs, using the ChannelID from PassThruConnect.

Table 14 summarizes the changes to the SAE J2534-1 API functions.

Table 14 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Add new ProtocolID values.
PassThruReadMsgs	Format of returned analog readings.
PassThruIoctl	Add new configuration parameters to control A/D.

10.3.2 API Functions - Detailed Information

10.3.2.1 PassThruConnect

The new protocol identifiers ANALOG_IN_1 through ANALOG_IN_32 connect to analog subsystems. Each subsystem can have up to 32 discrete equivalent channels, allowing for as many as 1024 analog inputs to be supported.

The various parameters (such as sample rate and averaging method) apply to all channels within a subsystem. The parameters can be different on different subsystems. A device with eight A/D channels should have eight subsystems only if each A/D channel can be controlled independently. The device should have one subsystem if all eight channels must have the same sample rate.

ProtocolIDs beyond those supported by the device shall return ERR_NOT_SUPPORTED.

10.3.2.2 ProtocolID Values

Only the definition and description of the ProtocolID value is defined Table 15. The actual value is defined in section titled "SAE J2534-2 Resource." Protocol values for the analog feature are identified in Table 15.

Table 15 - ProtocolID values

Definition	Description
ANALOG_IN_1	Analog subsystem 1
ANALOG_IN_2	Analog subsystem 2
ANALOG_IN_3	Analog subsystem 3
…	…
ANALOG_IN_32	Analog subsystem 32

10.3.2.3 PassThruReadMsgs

To the application, each analog subsystem appears like all the other vehicle protocols. An analog subsystem will periodically generate PASSTHRU_MSG structures which are placed in the queue where the application can read them. The normal PassThruReadMsg features, such as waiting (using timeout) and gathering multiple messages (using *pNumMsgs) are supported.

See the Message Structure sub-sections for the formatting of the samples within a message.

10.3.2.4 PassThruWriteMsgs

This function will return ERR_NOT_SUPPORTED if passed a ChannelID opened for analog input.

10.3.2.5 PassThruStartPeriodicMsg

This function will return ERR_NOT_SUPPORTED if passed a ChannelID opened for analog input.

10.3.2.6 PassThruStartMsgFilter

This function will return ERR_NOT_SUPPORTED if passed a ChannelID opened for analog input.

10.3.3 IOCTL Section

Each analog subsystem shall support 3 IOCTL functions: GET_CONFIG, SET_CONFIG, and CLEAR_RX_BUFFER. The CLEAR_RX_BUFFER IOCTL shall remove any queued messages for the subsystem.

10.3.3.1 GET_CONFIG

There are several new parameters that are used to setup and control the A/D subsystem. See Table 16 for more details.

10.3.3.2 SET_CONFIG

The following parameters control the analog subsystem. Note that there is no way to set parameters for each channel individually. See Table 16 for more details.

Table 16 - IOCTL GET_CONFIG/SET_CONFIG parameter details

Parameter	Valid Values for Parameter	Default (decimal)	Description
ACTIVE_CHANNELS	0 - 0xFFFFFFFF	Hardware dependent	Bitmask of channels being sampled
SAMPLE_RATE	0 - 0xFFFFFFFF	0	Samples/second or seconds/sample
SAMPLES_PER_READING	1 - 0xFFFFFFFF	1	Samples to average into a single reading
READINGS_PER_ MSG	1 - 0x00000408 (1 - 1032)	1	Number of readings for each active channel per PASSTHRU_MSG structure
AVERAGING_METHOD	0 - 0xFFFFFFFF	0	The way in which the samples will be averaged
SAMPLE_RESOLUTION	0x1-0x20 (1 - 32)	Hardware dependent	The number of bits of resolution for each channel in the subsystem (read only)
INPUT_RANGE_LOW	0x80000000 - 0x7FFFFFFF (-2147483648 - 2147483647)	Hardware dependent	Lower limit in millivolts of A/D input (example: 0xFFFFB1E0 = -20.00V) (read only)
INPUT_RANGE_HIGH	0x80000000 - 0x7FFFFFFF (-2147483648 - 2147483647)	Hardware dependent	Upper limit in millivolts of A/D input (example: 20000 = +20.00V) (read only)

10.3.3.2.1 ACTIVE_CHANNELS

The ACTIVE_CHANNELS parameter controls the number of channels that are actively read into the PASSTHRU_MSG structure. The ACTIVE_CHANNELS parameter is a 32-bit unsigned long bit mask. Each bit that is set indicates that the corresponding channel is active.

Changes to the ACTIVE_CHANNELS take effect after the completion of the current message (i.e., specified readings per message).

The interface must reject combinations of ACTIVE_CHANNELS and READINGS_PER_MSG that would result in a message that is larger than the size of a PASSTHRU_MSG structure (1032 data points). In this case, the error returned shall be ERR_INVALID_IOCTL_VALUE. The interface may not reject valid combinations of ACTIVE_CHANNELS and READINGS_PER_MSG.

The default value for ACTIVE_CHANNELS is for all available channels to be active. For example, a subsystem with seven channels will set ACTIVE_CHANNELS to 0x7F (127) initially. Trying to set bits for channels that don't exist will return error ERR_INVALID_IOCTL_VALUE.

10.3.3.2.2 SAMPLE_RATE

The SAMPLE_RATE parameter sets the number of samples per second or the number of seconds per sample for each of the active channels. If the SAMPLE_RATE is less than 0x80000000, then the SAMPLE_RATE represents the number of samples per second. For example, 0x7D0 represents 2000 samples/second for each channel. On the other hand, values above 0x80000000 represent seconds per sample (minus the most significant bit). For example, 0x8000000A would be one sample on each channel every 10 seconds. Note that 0x80000001 is the same as 0x00000001. 0x80000000 should be treated the same as 0.

Setting this value to zero has the effect of disabling the associated A/D subsystem. No new messages will be queued, but the existing messages will not be cleared.

If the device does not support the requested sample rate, the device must return ERR_INVALID_IOCTL_VALUE. Changes to this value take effect at the end of the current cycle or immediately if the subsystem was disabled.

The default value for SAMPLE_RATE is zero (subsystem disabled).

NOTE: Setting SAMPLE_RATE to 0 value will stop the data streaming.

10.3.3.2.3 SAMPLES_PER_READING

The SAMPLES_PER_READING parameter sets the number of samples per reading. The parameter AVERAGING_METHOD determines how the reading will be derived from the collected samples.

As you increase the SAMPLES_PER_READING, you increase the number of samples required to fill a PASSTHRU_MSG structure. For example, setting SAMPLES_PER_READING to three (without changing other parameters) will make the messages come three times slower.

If the device does not support the requested value, the device must return ERR_INVALID_IOCTL_VALUE. The device must support the default value of one, even if the device does not support averaging. A value of one means that averaging is off.

The default value for SAMPLES_PER_READING is one.

The return code ERR_NOT_SUPPORTED will be returned if the sample rate is not zero.

10.3.3.2.4 READINGS_PER_MSG

The READINGS_PER_MSG parameter sets the number of readings of each active channel that will be placed in a PASSTHRU_MSG structure.

The readings will be placed in the PASSTHRU_MSG message in channel order starting from lowest active channel to highest active channel. This format will repeat "READINGS_PER_MSG" times.

The interface must reject combinations of ACTIVE_CHANNELS and READINGS_PER_MSG that would result in a message that is larger than the size of a PASSTHRU_MSG structure (1032 data points). In this case, the error returned shall be ERR_INVALID_IOCTL_VALUE. The interface shall not reject valid combinations of ACTIVE_CHANNELS and READINGS_PER_MSG.

Setting this value to zero has the effect of disabling the associated A/D subsystem.

Changes to this value take affect at the end of the current cycle or immediately if the subsystem was disabled.

The default value for READINGS_PER_MSG is one.

The return code ERR_NOT_SUPPORTED will be returned if the sample rate is not zero.

10.3.3.2.5 AVERAGING_METHOD

When SAMPLES_PER_READING is above one, each reading will consist of several samples. The AVERAGING_METHOD specifies how each reading will be computed. If the device does not support a particular value, ERR_INVALID_IOCTL_VALUE shall be returned. The default value (SIMPLE_AVERAGE) must be supported, even if the device does not support averaging. See Table 17 for more details.

Table 17 - Values for the AVERAGING_METHOD parameter

Method	Value	Description
SIMPLE_AVERAGE	0x00000000	Simple arithmetic mean
MAX_LIMIT_AVERAGE	0x00000001	Choose the biggest value
MIN_LIMIT_AVERAGE	0x00000002	Choose the lowest value
MEDIAN_AVERAGE	0x00000003	Choose arithmetic median
(SAE J2534-2 reserved)	0x00000004 - 0x7FFFFFFF	Reserved
(Vendor Reserved)	0x80000000 - 0xFFFFFFFF	Specific to the vendor

10.3.3.2.5.1 SIMPLE_AVERAGE

The SIMPLE_AVERAGE is the arithmetic average of SAMPLES_PER_READINGS samples. In other words:

Reading = $(\text{Sample}_1 + \text{Sample}_2 + \ldots + \text{Sample}_{\text{SAMPLES\_PER\_READING}})/\text{SAMPLES\_PER\_READING}$

10.3.3.2.5.2 MAX_LIMIT_AVERAGE

The MAX_LIMIT_AVERAGE simply chooses the maximum value.

Reading = Max $(\text{Sample}_1 + \text{Sample}_2 + \ldots + \text{Sample}_{\text{SAMPLES\_PER\_READING}})$

10.3.3.2.5.3 MIN_LIMIT_AVERAGE

The MIN_LIMIT_AVERAGE simply chooses the minimum value.

Reading = Min $(\text{Sample}_1 + \text{Sample}_2 + \ldots + \text{Sample}_{\text{SAMPLES\_PER\_READING}})$

10.3.3.2.5.4 MEDIAN_AVERAGE

The MEDIAN_AVERAGE chooses the median value. Sort the samples, then compute:

Reading = $\text{Sample}_{(\text{SAMPLES\_PER\_READING}+1)/2}$

Or if SAMPLES_PER_READING is even:

Reading = $(\text{Sample}_{\text{SAMPLES\_PER\_READING}/2} + \text{Sample}_{(\text{SAMPLES\_PER\_READING}/2)+1})/2$

10.3.3.2.5.5 Vendor-specific Averaging Methods

Vendors are free to add their own averaging methods. They should use the range provided so they do not conflict with extensions to this standard.

10.3.3.2.6 SAMPLE_RESOLUTION

This read-only parameter indicates the number of bits of resolution that the A/D channels have in this subsystem. For example, a 12-bit A/D would return 12. Attempting to set this parameter shall return the value ERR_INVALID_IOCTL_PARAM_ID.

10.3.3.2.7 INPUT_RANGE_LOW

This signed, read-only parameter indicates the lower limit of the A/D subsystem. For example, an A/D subsystem that can measure voltages from -20.0V to +36V would return -20000 (0xFFFFB1E0). Attempting to set this parameter shall return the value ERR_INVALID_IOCTL_PARAM_ID.

10.3.3.2.8 INPUT_RANGE_HIGH

This signed, read-only parameter indicates the upper limit of the A/D subsystem. For example, an A/D subsystem that can measure voltages from -20.0V to +36V would return 36000 (0x00008CA0). Attempting to set this parameter shall return the value ERR_INVALID_IOCTL_PARAM_ID.

10.3.3.3 CLEAR_RX_BUFFER

The device shall remove any queued messages for the subsystem.

10.4 Message Structure

When a set of readings is ready, the device will queue a PASSTHRU_MSG structure for the application to read. This section specifies how to fill in that structure:

- ProtocolID contains the analog protocol that was connected (i.e., ANALOG_IN_1).
- RxStatus contains the overflow flags (see 10.4.2).
- TxFlags shall be zero and should be ignored by the application.
- Timestamp contains the time stamp of the first set of readings in the message. The application can calculate the timestamp of the remaining readings. The timestamp must correlate with the timestamp of normal message traffic.
- DataSize contains the number of bytes that the readings take up in the message. DataSize must be a multiple of 4 between 4 (a single reading) and 4128 (a full message), inclusive. The value of DataSize can be computed as READINGS_PER_MSG * (# bits set in ACTIVE_CHANNELS).
- Data[] contains the actual readings. The formatting of the data depends on the various parameters:
  - Each reading will take 4 bytes (32 bits), signed little endian format in millivolts.
  - All active channels in the subsystem (and only channels marked active) are represented. Each active channel is appended in order (starting from lowest active channel to highest active channel).
  - This format will repeat READINGS_PER_MSG times.

10.4.1 Examples

The SAE J2534 device assumed for this example has two analog input subsystems that it supports via protocols ANALOG_IN_1 and ANALOG_IN_2. The ANALOG_IN_1 subsystem provides four 16-bit A/D converters. The ANALOG_IN_2 subsystem provides two 24-bit A/D converters.

Tables 18 to 23 provide examples of different parameters and the resulting structures.

Table 18 - Sample A/D parameter configuration

Parameter	Value for ANALOG_IN_1	Value for ANALOG_IN_2
ACTIVE_CHANNELS	0xF	3
SAMPLE_RATE	2	0x80000005
SAMPLES_PER_READING	1	1
READINGS_PER_MSG	1	1
AVERAGING_METHOD	1	1
SAMPLE_RESOLUTION	16	24

This example uses the default values, except that a SAMPLE_RATE has been set for both subsystems. The first subsystem generates the following data twice per second:

NOTE: Only the PASSTHRU_MSG Data[] array is shown. Each box represents a 4-byte sample.

Table 19 - Data from ANALOG_IN_1 subsystem with READINGS_PER_MSG = 1

Channel 1 Sample 1	Channel 2 Sample 1	Channel 3 Sample 1	Channel 4 Sample 1

The second subsystem generates the following data once every 5 seconds:

Table 20 - Data from ANALOG_IN_2 subsystem with READINGS_PER_MSG = 1

Channel 1 Sample 1	Channel 2 Sample 1

Changing the READINGS_PER_MSG parameter on each channel from 1 to 2 changes the format and the rate of messages. The first subsystem now generates this message once per second:

Table 21 - Data from ANALOG_IN_1 subsystem with READINGS_PER_MSG = 2

Channel 1 Sample 1	Channel 2 Sample 1	Channel 3 Sample 1	Channel 4 Sample 1	Channel 1 Sample 2	Channel 2 Sample 2	Channel 3 Sample 2	Channel 4 Sample 2

The second subsystem now generates this message every 10 seconds:

Table 22 - Data from ANALOG_IN_2 subsystem with READINGS_PER_MSG = 2

Channel 1 Sample 1	Channel 2 Sample 1	Channel 1 Sample 2	Channel 2 Sample 2

Changing the ACTIVE_CHANNELS on subsystem 1 to 0xB (11 decimal, 1011 binary) disables channel 3. In this case, the structure would now be:

Table 23 - Data from ANALOG_IN_1 subsystem with ACTIVE_CHANNELS = 0xB

Channel 1 Sample 1	Channel 2 Sample 1	Channel 4 Sample 1	Channel 1 Sample 2	Channel 2 Sample 2	Channel 4 Sample 2

10.4.2 Message Flag and Status Definitions

Definitions for RxStatus bits are shown in Table 24.

Table 24 - RxStatus bit definitions

Definition	**RxStatus Bit(s)**	**Description**	**Value**
OVERFLOW	16	Indicates that the input range of the A/D has been exceeded on one or more samples in the received message	0 = All samples good 1 = Some samples clipped

10.4.3 DLL Installation and Registry

A registry key ANALOG_IN_x indicates the support for an analog sub-system where <x> indicates the sub-system number.

Example: ANALOG_IN_3 indicates support for Analog Sub-System #3, which can have up to 32 analog input channels.

In the case of devices which contain dynamic hardware architecture, an application will be able to determine the lack of runtime ANALOG_IN_x support when ERR_NOT_SUPPORTED is returned from a PassThruConnect().

The registry entries are retained for support of legacy applications. New applications shall use the discovery mechanism defined Section 25.

10.5 Discovery Support

The device shall report support for Analog channels and associated parameters through the discovery mechanism defined in Section 25.

11. GM UART (SAE J2740)

11.1 Scope of the GM UART Optional Feature

Information contained in this section will define extensions to a compliant SAE J2534-1 interface. This section specifically defines the common method of supporting GM's UART protocol as defined in SAE J2740.

11.2 Pass-Thru System Requirements

11.2.1 Pin Usage

All GM vehicles built since the 1996 model year, and a few built during the 1995 model year, have been equipped with an SAE J1962 connector. GM UART uses either pin 9 or pin 1 of this connector. Typically, SAE J1962 pin 9 (primary) is used while SAE J1962 pin 1 (secondary) is occasionally used. As with all SAE J2534-2 optional protocols, no default pin is identified; therefore, the application developer will be required to set the pin to be used. Refer to SAE J1962 for discussion of pin usage.

Most GM vehicles with serial data links built prior to the 1996 model year are equipped with a 12-pin connector as shown in Figure 3. The mating tool connector is shown in Figure 4. For programming these older vehicles using an SAE J2534 interface, a 12-pin connector must be available instead of an SAE J1962 connector to interface to the vehicle. The signal ground (pin 5 on an SAE J1962 connector) must be connected to pin A of the 12-pin connector. The serial data line (pin 9 on an SAE J1962 connector) must be connected to pin M of the 12-pin connector. The 12-pin connector does not contain battery power, so the SAE J2534 interface cannot be powered from the 12-pin connector.

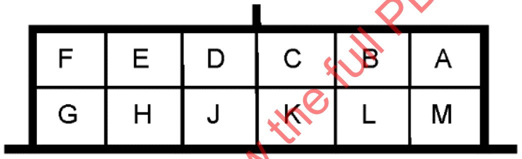


Figure 3 - 12-pin vehicle connector

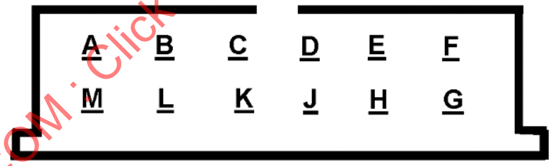


Figure 4 - 12-pin tool connector

11.3 Win32 Application Programming Interface

11.3.1 API Functions - Overview

Information contained in this section is intended to define the API resources required to incorporate an optional protocol channel. This protocol, identified as the GM UART protocol, will require software API support and hardware to fully implement this feature.

At a high level, the ProtocolID GM_UART_PS and GM_UART_CHx have been defined to indicate the physical layer. The protocol defines a master/slave relationship between the tester and the electronic control unit (ECU). The tester must request and be granted mastership over the vehicle bus before communications can begin. This section details the API resources required to enable use of the SAE J2534 API as applied to GM UART protocol. If the feature is not supported, an error code, ERR_NOT_SUPPORTED, will be returned by the call to PassThruConnect.

Generally, vehicle bus mastership is accomplished by calling the PassThruIoctl function BECOME_MASTER which monitors the vehicle bus for a poll message and when the poll message is received, a poll response message is returned. Upon receiving the poll response message, the current master will relinquish mastership to sender. However, in some vehicles, there is no poll message, so the SAE J2534 device will be instructed by the application to send the poll response message immediately.

Prior to calling the PassThruIoctl function BECOME_MASTER, the application will first listen to the communication link (using PassThruReadMsgs function) to determine if a tester polling message exists, the type of the polling message (3 byte or 4 byte), and the Device ID of the polling device (4-byte poll only). Using this data, the application will call the PassThruIoctl function SET_POLL_RESPONSE to define the poll response message. The application then calls the BECOME_MASTER function to direct the interface to either wait for a poll message with the same Message ID Byte (MIB) as specified in the poll_ID parameter passed to the BECOME_MASTER command, or to send the poll response message immediately (depending upon the poll ID specified).

Table 25 summarizes the changes to the SAE J2534-1 API functions.

Table 25 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.
PassThruIoctl	Added new PassThruIoctl sub-functions - SET_POLL_RESPONSE and BECOME_MASTER.

11.3.2 API Functions - Detailed Information

11.3.2.1 PassThruConnect

When PassThruConnect is called with the GM_UART_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

11.3.2.1.1 ProtocolID Values (GM UART)

One additional ProtocolID Value has been defined. Only the definition and description of the ProtocolID value is defined in Table 26. The actual value is defined in Section 24.

Table 26 - Protocol ID values

Definition	Description
GM_UART_PS	GM UART protocol with pin selection
GM_UART_CHx	Additional channels of GM UART protocol

11.3.3 IOCTL Section

Table 27 provides the details on the IOCTLs available through the PassThruIoctl function.

Table 27 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr Represents	Purpose
SET_POLL_RESPONSE	Points to SBYTE_ARRAY which contains the poll response message	NULL pointer	Defines poll response message.
BECOME_MASTER	Poll ID: 0 - don't wait for poll message or Non-zero - wait for poll ID value	NULL pointer	Waits for valid poll message and upon receipt transmits poll response message if POLL_ID does not equal zero. If Poll ID is zero, poll response message is sent immediately (no wait).

11.3.3.1 SET_POLL_RESPONSE

Details of the SET_POLL_RESPONSE are shown in Table 28. The IoctlID parameter of SET_POLL_RESPONSE is used to define the poll response message. Typically, the poll response message is a "Disable Normal Communications" (Mode 8) message, but this will be set by the application.

Table 28 - SET_POLL_RESPONSE details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect.
IoctlID	Is set to SET_POLL_RESPONSE
InputPtr	Points to unsigned char PollResponseMsg[100]
OutputPtr	NULL pointer

See Table 29 for a typical 4-byte poll response message example:

Table 29 - Poll response message

Byte	Value	Description
1	F4	Message ID (MIB)
2	56	Message length (not including message ID OR CHECKSUM) + 0X55
3	08	ALDL Mode ID
4	CS	CHECKSUM

The poll response message shall be sent by the interface upon the receipt of a tester poll message. There are two types of tester poll messages based on length. The 4-byte type includes a device ID while the 3-byte type does not. The two types are described are described in Tables 30 and 31.

Table 30 - Typical 3-byte type poll message

Byte	Value	Description
1	F0	Message ID (MIB)
2	55	Message length (not including message ID or CHECKSUM) + 0X55
3	CS	CHECKSUM

Table 31 - Typical 4-byte type poll message

Byte	Value	Description
1	F0	Message ID (MIB)
2	56	Message length (not including message ID or CHECKSUM) + 0X55
3	F4	Device ID
4	CS	CHECKSUM

11.3.3.2 BECOME_MASTER

The IoctlID value of BECOME_MASTER will transmit the Poll Response Message as identified in the SET_POLL_RESPONSE function depending upon which of the two modes it is functioning under. The two functional modes are:

1. Polling mode, where the poll response message begins within 1.42 ms after receipt of a valid poll message (start of transmission is in 1.42 ms). Refer to SAE J2740 for more details.

2. Non-polling mode, where the poll response message is sent out immediately.

Specifying a Poll_ID parameter value of zero puts BECOME_MASTER into non-polling mode, while a Poll_ID parameter of non-zero puts BECOME_MASTER into polling mode.

If in polling mode, BECOME_MASTER will scan incoming messages for one with a Message ID equal to the Poll_ID parameter value specified in the BECOME_MASTER command. Upon receipt of a valid poll message, BECOME_MASTER will respond by transmitting the poll response message, or, if no poll message is received within 2 seconds, the function will return to the caller with an ERR_FAILED error. In the event that the poll response message was successfully transmitted, the BECOME_MASTER function will return STATUS_NOERROR.

See Table 32 for BECOME_MASTER detail.

When a poll message is received, it may be either a 4-byte type (containing device ID) or a 3-byte type (no device ID).

Table 32 - BECOME_MASTER detail

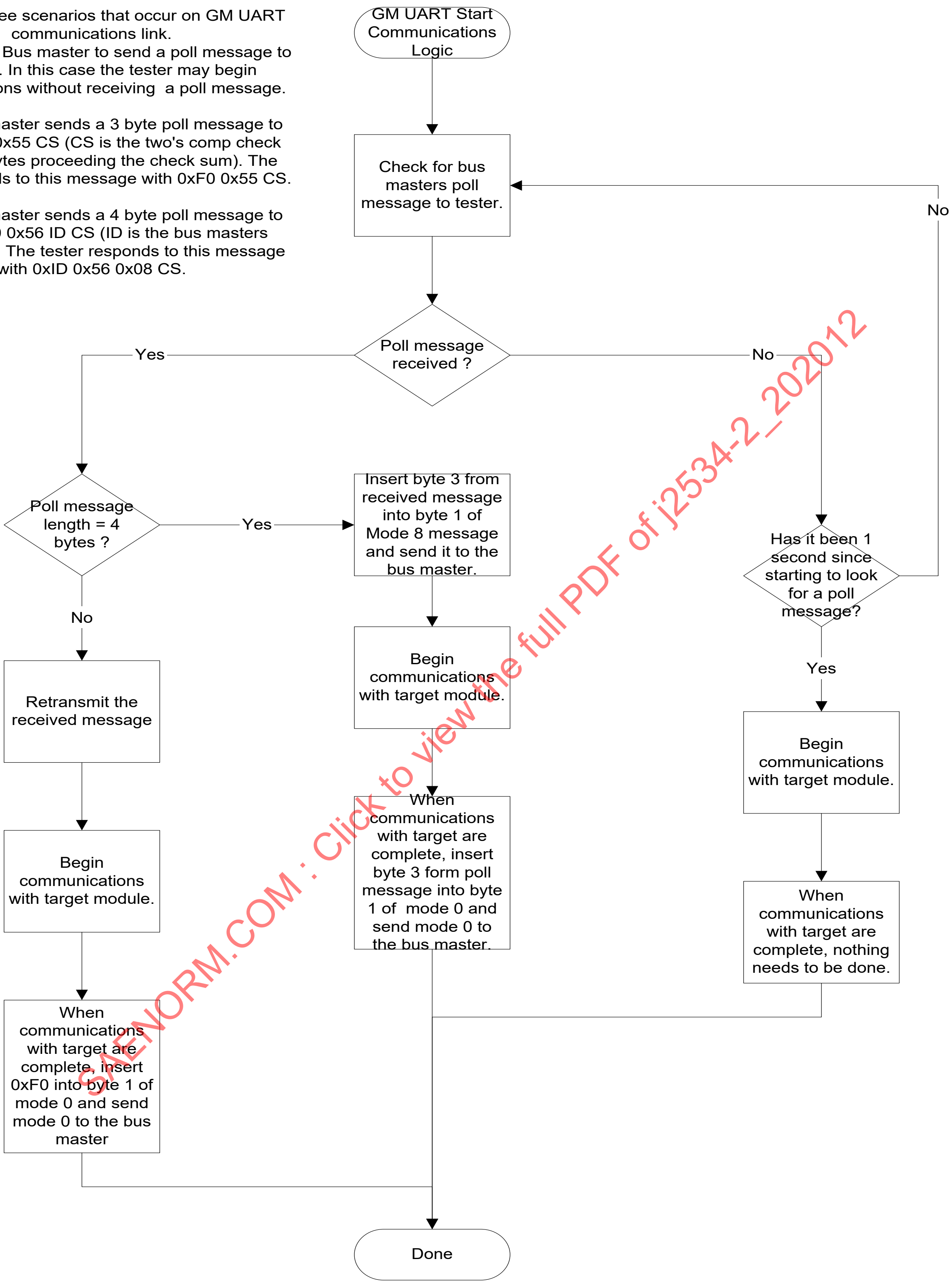
Parameter	Description
ChannelID	Channel ID assigned by DLL during PASSTHRUCONNECT
IoctlID	Is set to BECOME_MASTER
InputPtr	Points to unsigned char Poll_ID (Value to be compared to Poll Message MIB)
OutputPtr	NULL pointer

Figure 5 defines the generalized process flow of obtaining bus mastership and is included for information only. Most of this logic will be included in the reprogramming application and does not need to be incorporated into an SAE J2534-2 device and driver.

There are three scenarios that occur on GM UART communications link.
1. There is no Bus master to send a poll message to the tester. In this case the tester may begin communications without receiving a poll message.

2. The bus master sends a 3 byte poll message to tester 0xF0 0x55 CS (CS is the two's comp check sum of all bytes proceeding the check sum). The tester responds to this message with 0xF0 0x55 CS.

3. The bus master sends a 4 byte poll message to tester 0xF0 0x56 ID CS (ID is the bus masters message ID). The tester responds to this message with 0xID 0x56 0x08 CS.



GM UART Start Communications Logic
Check for bus masters poll message to tester.
No
Poll message received ?
Yes
No
Poll message length = 4 bytes ?
Yes
Insert byte 3 from received message into byte 1 of Mode 8 message and send it to the bus master.
Has it been 1 second since starting to look for a poll message?
No
Yes
Retransmit the received message
Begin communications with target module.
Begin communications with target module.
Begin communications with target module.
When communications with target are complete, insert byte 3 form poll message into byte 1 of mode 0 and send mode 0 to the bus master.
When communications with target are complete, nothing needs to be done.
When communications with target are complete, insert 0xF0 into byte 1 of mode 0 and send mode 0 to the bus master
Done

Figure 5 - Process flow for obtaining bus mastership

11.4 Message Structure

11.4.1 Elements

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID.

11.4.2 Message Data Formats

One additional protocol has been added and is defined in Table 33.

Table 33 - Allowed message sizes per protocol

Protocol ID	Min Tx	Max Tx	Min Rx	Max Rx	Notes
GM_UART_PS or GM_UART_CHx	3	170	3	170	3 header bytes containing destination ID, source ID and length. Length will be the actual length plus 0x55. Messages can contain 0 data bytes.

11.5 DLL Installation and Registration

Upon device installation, a key will be set in the appropriate registry folder to indicate the support for the GM UART protocol. In the case of devices which contain dynamic hardware architecture, an application will be able to determine the lack of runtime support when an error is returned from a PassThruConnect().

The registry entries are retained for support of legacy applications. New applications shall use the discovery mechanism defined in Section 25.

11.6 Discovery Support

The device shall report support for the GM UART protocol and associated parameters through the discovery mechanism defined in Section 25.

12. UART ECHO BYTE PROTOCOL

12.1 Scope of the UART Echo Byte Protocol Optional Feature

This section details the extensions to an SAE J2534-1 interface that will allow support of the UART Echo Byte protocol. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed from SAE J2534-1

12.2 Pass-Thru System Requirements

12.2.1 Simultaneous Communication on Multiple Protocols

Support for simultaneous communication of UART Echo Byte protocol with other protocols shall be equivalent to the definition for ISO9141 in SAE J2534-1.

12.2.2 Pin Usage

VW and Audi use of UART Echo Byte is connected to pin 7 of the SAE J1962 connector.

As with all SAE J2534-2 optional protocols, no default pin is identified. Therefore, the application developer will be required to set the pin to be used.

See Section 6 for details of the method used to switch SAE J1962 pins.

12.3 Win32 Application Programming Interface

12.3.1 API Functions - Overview

Feature Implementation Summary:

This feature supports a UART based protocol that uses an ISO9141 K line physical layer interface.

Refer to SAE J2809 Honda ABS/VSA or SAE J2818 KWP 1281 for the detailed protocol information.

ECU wakeup is done by a variation of the ISO9141-2 5 baud wakeup sequence, where the transmission of the inverse of the wakeup address by the ECU is replaced by the transmission of an ECU ID message.

A key feature of the protocol is the byte echo scheme employed for every tester and ECU transmission. Following the transmission of each byte from the source device, the destination device responds with the complement of the received byte. This interleaved sequence continues until the complete message has been transmitted by the source device.

The guidelines to implement this communication capability to SAE J2534-2 API are shown below:

- The interface shall receive and check the echo bytes returned from ECU during transmitting request message.
- Echoed bytes received from the ECU during a transmission by the interface shall not be returned to the application.
- Echo bytes generated by the interface when receiving a message from the ECU shall not be returned to the application.
- When any error is detected in transmitting a request message as defined in the "Error Handling During Normal Communication" sections of SAE J2809 or SAE J2818, the interface shall retransmit the failed message. If two subsequent retransmissions fail, the error shall be indicated to the application (i.e., ERR_TIMEOUT is returned in case of a blocking write). To account for this, the application should set the PassThruWriteMsgs timeout to a value that accounts for the possibility of the three transmission attempts by the interface.
- The interface shall automatically transmit the echo bytes when receiving a response message.
- For response messages, if the interface detects ETX (the last byte) in response message lost, the interface shall wait for two times UEB_T7_MAX to receive possible repeat transmission of the response message from the ECU. Following two times the value of UEB_T7_MAX expiry, the response message with absent ETX shall be queued (in this case, the application will detect loss of ETX and request the interface to transmit a No Acknowledge message).
- For a message to be valid, bytes at least up to ETX (excluding ETX) must be received. Otherwise, the message will be discarded.
- The message counter (the second byte of messages) shall be generated and controlled by the application.
- The application is responsible for handling received No Acknowledge messages from the ECU and repeating transmission of the requested message.
- The interface shall discard any bytes received unexpectedly, i.e., outside the bounds of an in-progress receive message.

The interface is not required to transmit Acknowledge messages to the ECU, except in the case of the ECU ID receive mechanism described in 12.3.3.2.

Messages passed to PassThruWriteMsgs by the application will include all message data bytes, including ETX. All received message bytes including ETX shall be queued by the interface for retrieval by calls to PassThruReadMsgs.

Support for the UART Echo Byte protocol is achieved by the addition of a new ProtocolID, modified behavior of FIVE_BAUD_INIT and a new low-level implementation for transmission and reception of messages. This protocol identified as UART_ECHO_BYTE_PS or UART_ECHO_BYTE_CHx requires additional API and device software support, but no additional hardware compared with a standard SAE J2534-1 device.

If UART Echo Byte protocol feature is not supported, ERR_NOT_SUPPORTED will be returned by the call PassThruConnect. The calling application will be required to notify the user that this optional feature may not be supported by the interface.

The SAE J2534-2 Repeat Messaging feature shall not be supported on this protocol as it is not feasible for the interface to correctly maintain MessageCounter values.

12.3.2 API Change List

Table 35 summarizes the changes to the SAE J2534-1 API functions.

Table 34 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.
PassThruWriteMsgs	This function only accepts one message at a time, i.e., pNumMsgs must always be 1.
PassThruStartPeriodicMsg	This function is not supported.
PassThruIoctl	FIVE_BAUD_INIT is enhanced. New configuration parameters are added.

NOTE: There is no change to PassThruStartMsgFilter function. However, the interface shall participate in the echoed byte receive process for all received messages, regardless of any filter settings. Standard SAE J2534-1 filtering shall be applied to all completed messages received by the interface.

12.3.3 API Functions - Detailed Information

12.3.3.1 PassThruConnect

When PassThruConnect is called with the UART_ECHO_BYTE_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

12.3.3.1.1 C/C++ Prototype

There are no changes defined for the function prototype.

12.3.3.1.2 Parameters

There are no changes to the parameters.

12.3.3.1.3 Connect Flag Values

There are no changes to the connect flags.

12.3.3.1.4 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 35. The actual value is defined in Section 24.

Table 35 - Protocol ID values

Definition	Description
UART_ECHO_BYTE_PS	Honda ABS/VSA described in SAE J2809, or KWP1281 described in SAE J2818, with pin selection
UART_ECHO_BYTE_CHx	Additional channels of Honda ABS/VSA described in SAE J2809 or KWP1281 described in SAE J2818

12.3.3.2 PassThruWriteMsgs

12.3.3.2.1 C/C++ Prototype

There are no changes defined for the function prototype.

12.3.3.2.2 Parameters

Due to the MessageCounter being incremented alternately by tester and ECU, multiple messages cannot be transmitted by a single call to PassThruWriteMsgs. Therefore, the only acceptable value for *pNumMsgs is one.

If value pointed to by pNumMsgs is greater than one, ERR_EXCEEDED_LIMIT shall be returned.

12.3.3.3 PassThruStartPeriodicMsg

This function is not supported on this protocol as it is not feasible for the interface to correctly maintain MessageCounter values. If this function is called, ERR_NOT_SUPPORTED shall be returned.

12.3.4 IOCTL Section

12.3.4.1 SET_CONFIG and GET_CONFIG Additional Parameters

Ten new configuration parameters will be added to support UART Echo Byte protocol.

Table 36 - SET_CONFIG and GET_CONFIG additional parameters

SAE J2534-2 Parameter	Valid Values	Default Values	Description
UEB_T0_MIN	0X0 - 0XFFFF (1 ms per bit)	10	Minimum idle time before the start of transmission of the address byte.
UEB_T1_MAX	0X0 - 0XFFFF (1 ms per bit)	400	Maximum time between correct stimulation and start of the synchronization byte.
UEB_T2_MAX	0X0 - 0XFFFF (1 ms per bit)	200	Maximum time between synchronization byte and key-byte 1.
UEB_T3_MAX	0X0 - 0XFFFF (1 ms per bit)	200	Maximum time between key-byte 1 and key-byte 2.
UEB_T4_MIN	0X0 - 0XFFFF (1 ms per bit)	1	Minimum time between key-byte 2 and complement key-byte 2.
UEB_T5_MAX	0X0 - 0XFFFF (1 ms per bit)	1000	Maximum time between key-byte 2 and the renewed output of the synchronization byte. (If complement key-byte 2 incorrectly received by control unit.)
UEB_T6_MAX	0X0 - 0XFFFF (1 ms per bit)	200	Maximum time between key-byte 2 and start of ECU identification.
UEB_T7_MIN	0X0 - 0XFFFF (1 ms per bit)	1	Minimum time between bytes within a message (i.e., between a byte from the ECU and its complement from the interface).
UEB_T7_MAX	0X0 - 0XFFFF (1 ms per bit)	40	Maximum time between bytes within a message (i.e., between a byte from the interface and its complement from the ECU).
UEB_T9_MIN	0X0 - 0XFFFF (1 ms per bit)	1	Minimum time between end of a message and start of the next message

The DATA_RATE parameter shall also be supported, and the default value shall be 9600 bps.

No other SAE J2534-1 defined configuration parameters are supported with the exception of SAE J1962_PINS.

Table 37 translates the UART Echo Byte protocol configuration parameters to the SAE J2809 and SAE J2818 parameters.

Table 37 - Parameter translation table

SAE J2534-2 Parameter	SAE J2809 Parameter	SAE J2818 Parameter	Description
UEB_T0_MIN	t_0	T_R0	Minimum idle time before the start of transmission of the address byte.
UEB_T1_MAX	t_1	t_r1	Maximum time between correct stimulation and start of the synchronization byte.
UEB_T2_MAX	t_2	t_r2	Maximum time between synchronization byte and key-byte 1.
UEB_T3_MAX	t_3	t_r3	Maximum time between key-byte 1 and key-byte 2.
UEB_T4_MIN	t_4	t_r4	Minimum time between key-byte 2 and complement key-byte 2.
UEB_T5_MAX	t_5	t_rkmax	Maximum time between key-byte 2 and the renewed output of the synchronization byte. (If complement key-byte 2 incorrectly received by control unit.)
UEB_T6_MAX	t_6	t_r5	Maximum time between key-byte 2 and start of ECU identification.
UEB_T7_MIN	t_7 minimum	t_r6	Minimum time between bytes within a message (i.e., between a byte from the ECU and its complement from the interface).
UEB_T7_MAX	t_7 maximum	t_r8 max	Maximum time between bytes within a message (i.e., between a byte from the interface and its complement from the ECU).
UEB_T9_MIN	t_9	t_rb	Minimum time between end of a message and start of the next message.

12.3.4.2 FIVE_BAUD_INIT

The interface to the IOCTL function is identical to the definition in SAE J2534-1. However, as described in the Telegram-Specific Overview section of SAE J2809, or Stimulation and Initialization section of SAE J2818, the transmission of the key-byte 2 is followed by the transmission of a two-part ECU ID message. The ECU ID message following 5-baud initialization shall be received and checked for correct timing by the interface, before indicating success of the initialization process. Then the ECU ID message shall be discarded by the interface.

Note that that interface shall not use the synchronization byte to set the baud rate. The baud rate shall be as specified by the DATA_RATE parameter. The interface shall not support the initialization retry strategy initiated by the control unit's retransmission of the synchronization byte, described in the Communication Set-Up section of SAE J2809 or SAE J2818. The sequence of the ECU ID message output is shown below. Note that the echoed complement bytes are omitted in the example data streams below:

- First part of the identification message is transmitted by the ECU.

  Example: 0x0F 0x01 0xF6 0x30 0x32 0x36 0x35 0x39 0x35 0x30 0x30 0x31 0x34 0x20 0x20 0x03

- The interface sends an Acknowledge message.

  Example: 0x03 0x02 0x09 0x03

- The ECU sends the second part of the identification message.

  Example: 0x0B 0x03 0xF6 0x42 0x42 0x20 0x32 0x39 0x30 0x36 0x30 0x03

- The interface sends a second Acknowledge message.

  Example: 0x03 0x04 0x09 0x03

- The ECU then sends an Acknowledge message in response.

  Example: 0x03 0x05 0x09 0x03

- This signals the end of the sequence.

The interface is only required to generate Acknowledge messages in the specific situation described above. In all other situations, the application generates the Acknowledge messages. Note that the interface must generate the Message Counter field in both Acknowledge messages described above, and the values shall be 2 and 4.

12.4 Message Structure

12.4.1 C/C++ Definition

There is no change to the C/C++ definition.

12.4.2 Message Data Formats

When using UART Echo Byte protocol, all message bytes including ETX are defined by the application.

All received messages that comply with the timing and message structure requirements shall be queued to the application.

UART Echo Byte Protocol IDs and message limitations are defined in Table 38.

Table 38 - Allowed message sizes per protocol

Protocol ID	Min Tx	Max Tx	Min Rx	Max Rx	Notes
UART_ECHO_BYTE_PS or UART_ECHO_BYTE_CHx	4	256	3	256	Data[0] = byte message length Data[1] = message counter Data[2] = message title Data[3 - (n-1)] = optional message-specific data Data[n] = ETX all application defined

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID.

12.4.3 Format Checks for Messages Passed to the API

The vendor DLL shall validate all PASSTHRU_MSG structures, and return an error, ERR_INVALID_MSG if DataSize violates Min Tx or Max Tx columns in Table 38.

12.4.4 Message Flag and Status Definitions

No TxFlags, RxStatus, or Indications are supported for this protocol. These flags are undefined for this protocol.

12.5 Return Value Error Codes

ERR_EXCEEDED_LIMIT will be returned when more than one message is written at a time.

12.6 Discovery Support

The device shall report support for UART Echo Byte protocol and associated parameters through the discovery mechanism defined in Section 25.

13. HONDA DIAG-H PROTOCOL

13.1 Scope of the Honda DIAG-H Optional Feature

This section details the extensions to SAE J2534-1 that will allow support of the Honda 92 Hm/2 protocol and DIAG-H protocol interface. DIAG-H is a bi-directional serial communication line that provides diagnostic communication capability using the “92 Hm/2” communication protocol which is Honda proprietary. DIAG-H line is a single wire communication bus based on a UART interface.

13.2 Pass-Thru System Requirements

13.2.1 Simultaneous Communication on Multiple Protocols

The interface shall be capable of supporting one of the protocols from data link set DATA LINK SET 1, one of the protocols from DATA LINK SET 2 and one of the protocols from DATA LINK SET 3.

An attempt to open an unsupported protocol channel shall result in the ERR_NOT SUPPORTED return code from PassThruConnect.

Table 39 - Simultaneous communication options

Data Link Set 1	Data Link Set 2	Data Link Set 3
HONDA_DIAGH_PS or HONDA_DIAGH_CHx	ISO9141 ISO14230	CAN ISO15765

13.2.2 Programmable Power Supply

There are no changes to this section.

13.2.3 Pin Usage

When the vehicle is equipped with an SAE J1962 connector, DIAG-H may be absent, or may be present on SAE J1962 pin 14 or pin 1. On vehicles that have the CAN_L line present on pin 14, normal mode communication between ECUs on the CAN bus may be corrupted by connection to a tester DIAG-H circuit. Therefore, the tester’s DIAG-H circuit shall be disconnected (500 kΩ minimum impedance) from pin 14 until commanded by the diagnostic application.

As with all SAE J2534-2 optional protocols, no default pin is identified. Therefore, the application developer will be required to set the pin to be used. Refer to SAE J1962 or SAE J2534-2 for details of the method used to switch SAE J1962 pins.

13.2.4 Honda Diagnostic Connectors

Figures below show the diagnostic communication line configuration of OBD regulation compliant and non-OBD regulation compliant Honda vehicles.

The DIAG-H protocol shall be supported on SAE J1962 pin 1 and on SAE J1962 pin 14.

Most Honda non-OBD (1992 to 2000 model year) vehicles are equipped with 3 pin or 5 pin DLC. The “DIAG-H” bi-directional serial communication line is connected to the DLC and provides serial data stream to the tester in 92 Hm/2 protocol.

The 3-pin and 5-pin DLC vehicles are supported by existing SAE J1962 adaptor cables that route SAE J1962 pin 14 to pin 1 of the 3-pin or 5-pin DLC. (The Honda diagnostic application always assigns DIAG-H to SAE J1962 pin 14 when communicating with these vehicles.)

On the other hand, Honda OBD compliant vehicles are equipped with 16-pin DLC conformed to SAE J1962/ISO15031-3 for the off-board diagnostic communication. Mainly, 92 Hm/2 protocol is used for the Chassis/Body System ECUs.

Note that N.C. in Figure 6 indicates that some 3-pin DLC vehicles do not include battery positive supply on the DLC. In this case, the SAE J2534 interface device must be powered by a separate cable, from the cigar lighter, for example.

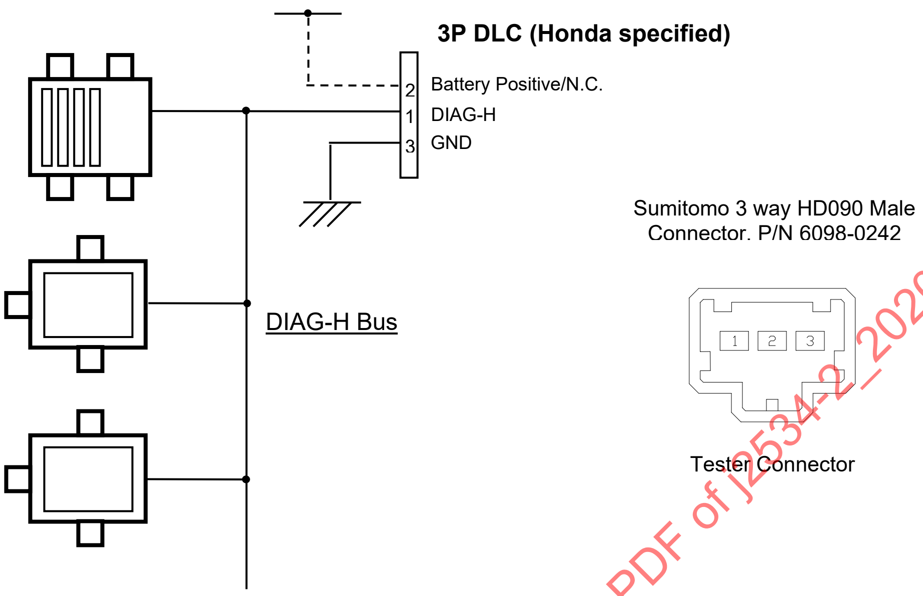


Figure 6 - Diagnostic communication line configuration (3-pin DLC model)

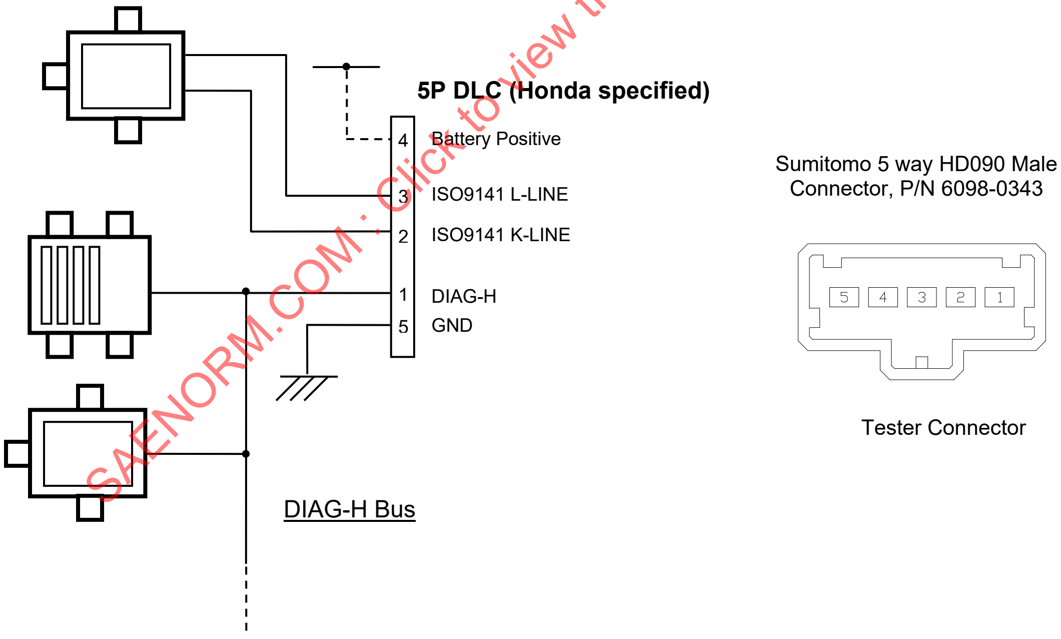


Figure 7 - Diagnostic communication line configuration (5-pin DLC model)

The maximum current on a ground pin of Honda 3-pin and 5-pin DLC is 1.5 A. It is equivalent to the specification for signal ground (pin 5) defined in SAE J1962.

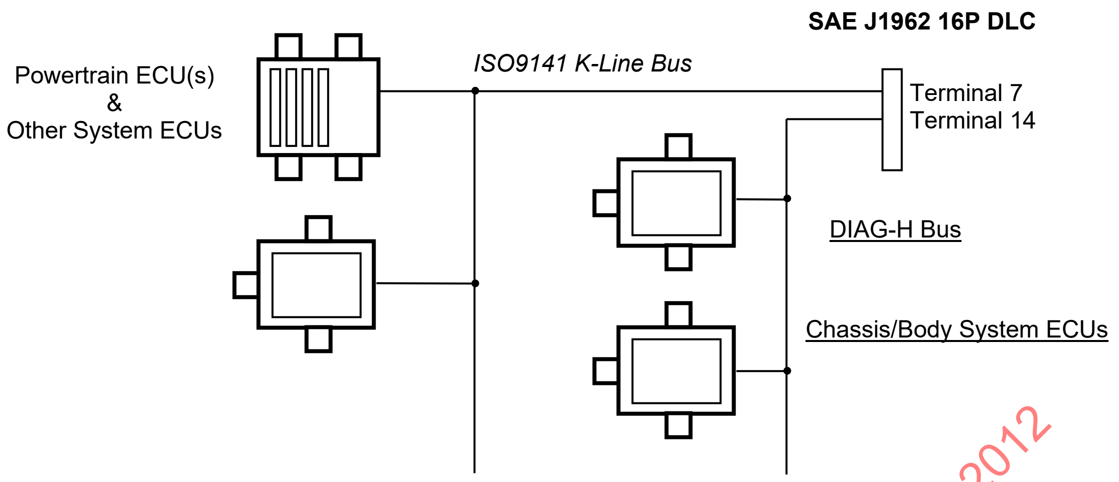


Figure 8 - Diagnostic communication line configuration (16-pin DLC model without CAN)

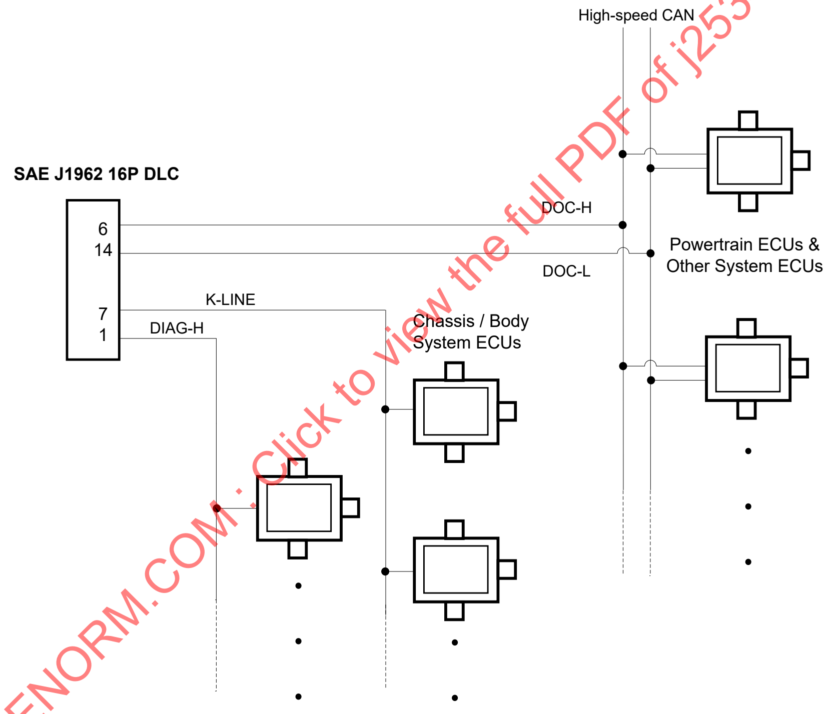

High-speed CAN
SAE J1962 16P DLC
6
14
7
1
DOC-H
DOC-L
K-LINE
DIAG-H
Powertrain ECUs & Other System ECUs
Chassis / Body System ECUs

Figure 9 - Diagnostic communication line configuration (16-pin DLC model with CAN)

13.2.5 Serial Communication Interface/Protocol

Table 40 describes the serial interface of Honda DIAG-H communication.

Table 40 - Serial interface parameters

Signal Line	"DIAG-H"; Bidirectional, Half-Duplex, Voltage Drive (0 to 5 V)
Communication Speed	9600 bps
Bit Encoding	NRZ Asynchronous
Byte Frame Format	1 start bit, 8 data bits, 1 stop bit No parity bit
Network Access	Master/slave Single request/single response (Any initialization process such as ISO9141/-2 is NOT required)
Message Format	Honda 92 message format; defined by HONDA

13.2.6 Electrical Characteristics of Interface

This section states the requirements of the serial communication interface of the off-board diagnostic tester to communicate with the ECU on the Honda vehicles through DIAG-H bus line.

13.2.6.1 Electrical Characteristics

Table 41 - Electrical characteristics

Item	Requirements	Description
Power Supply Voltage (Vcc)	5.0 V ± 5%	Voltage of power supply to DIAG-H line
Capacitance (C_T)	Less than 2000 pF	Total capacitance of tester and all its cabling (see Figure 10)
Pull-Up Resistance	4.7 kΩ ± 5%	Resistance pull-up to Vcc

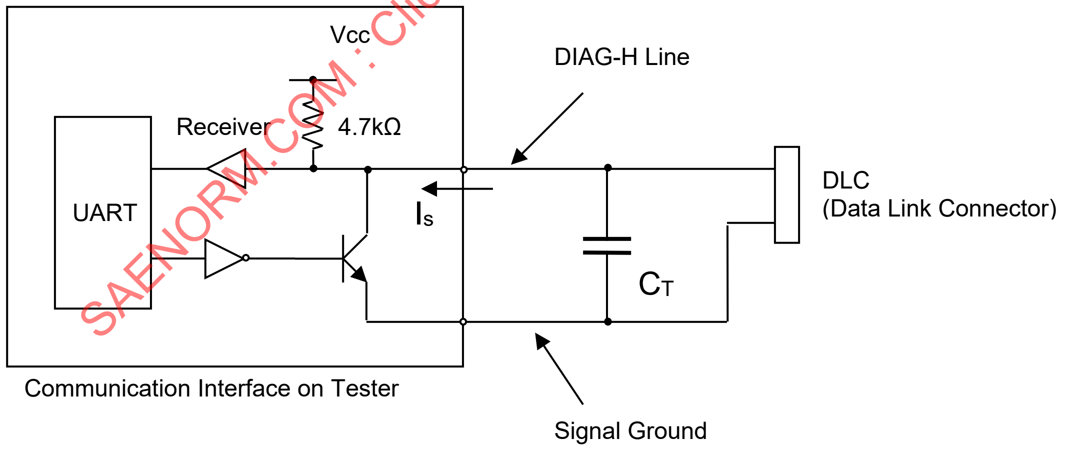


Figure 10 - Recommended interface circuit on tester

13.2.6.2 Requirements of Transmitter

Table 42 - Transmitter electrical requirements

Item	Requirements	Description
Logic 0 Voltage	Less than 0.5 V	Output voltage at logic 0
Sink Current (IS)	More than 50 mA	Sink Current at logic 0 (Figure 10)
Logic 1 Voltage	More than 4.5 V	Output voltage at logic 1 with the load shown in Figure 11

Table 43 - Transmitter timing requirements

Item	Requirements	Condition
Transmission Speed	9600 bps ± 0. 2%	At DLC terminal
t_{DST} (Figure 12)	5.0 seconds	Disconnected from vehicle

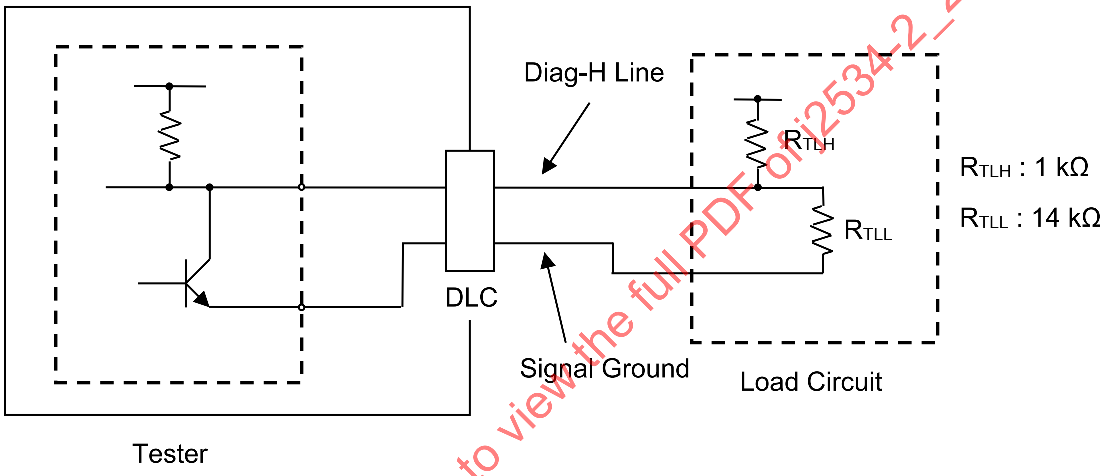
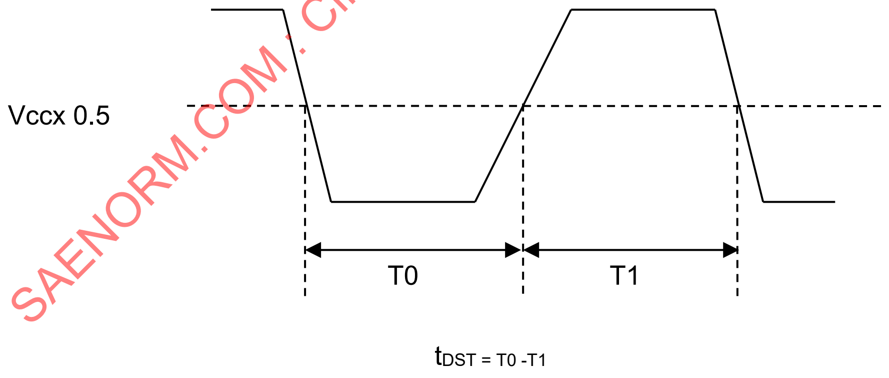


Figure 11 - Load circuit



"t_{DST}" is defined as time difference between duration of logic 0 and 1, when 0 and 1 bits are transmitted alternately.

Figure 12 - Definition of t_{DST}

13.2.6.3 Requirements of Receiver

Table 44 - Receiver electrical requirements

Item	Requirements	Description
Logic 0 Threshold	More than 1.9 V	Threshold voltage to determine logic 0
Logic 1 Threshold	Less than 3.1 V	Threshold voltage to determine logic 1
Internal Resistance	More than 100 k ± (to Signal Ground) More than 4.4 kΩ (to Vcc)	Internal resistance in receiving state
Hysteresis	More than 0.4 V	Difference of threshold voltage between 0 and 1

Table 45 - Receiver timing requirements

Item	Requirements	Condition
Acceptable Communication Speed	9600 bps ± 1.5%	With wiring and ECUs *1
t_{DST} (See Figure 12)	±17.0 seconds	

*1 - - - - Total Capacitance of Vehicle: Less than 2700 pF.
ECU pulled-up resistor to Vcc: 4.7 k ± 5% (maximum 7 units).

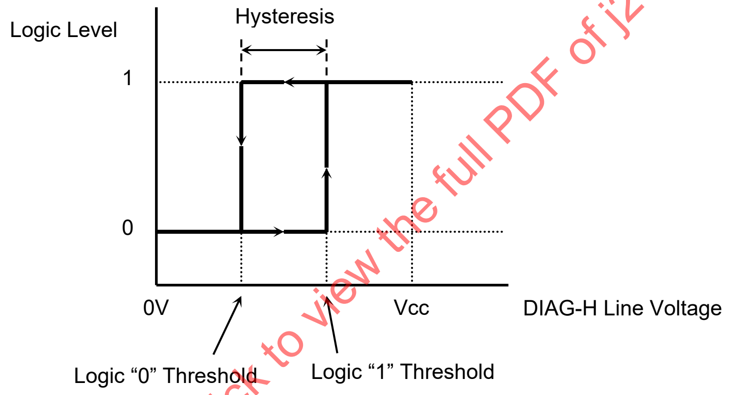


Figure 13 - Threshold voltage

13.3 Win32 Application Programming Interface

13.3.1 API Functions - Overview

Feature Implementation Summary:

The DIAG-H physical interface supports the 92 Hm/2 diagnostic communication protocol that is a UART-based protocol operating at 9600 baud. The SAE J2534-2 support of this protocol is based on the existing SAE J2534-1 definitions for ISO9141. ISO9141 timing parameters are used. No support for 5 baud or fast initialization is required.

The interface is not required to perform any message transmission retries. Retries shall be handled by the application.

Support for the DIAG-H interface and 92 Hm/2 protocol is achieved by the addition of a new physical layer interface in the interface hardware and a new ProtocolID.

API Change List:

Table 46 summarizes the changes to the SAE J2534-1 API functions.

Table 46 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values

13.3.2 API Functions - Detailed Information

13.3.2.1 PassThruConnect

When PassThruConnect is called with the HONDA_DIAGH_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

13.3.2.1.1 C/C++ Prototype

There are no changes defined for the function prototype.

13.3.2.1.2 Parameters

There are no changes to the parameters.

13.3.2.1.3 Connect Flag Values

There are no changes to the connect flags.

13.3.2.1.4 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 47. The actual value is defined in Section 24.

Table 47 - Protocol ID values

Definition	Description
HONDA_DIAGH_PS	Honda DIAG-H protocol with pin selection
HONDA_DIAGH_CHx	Additional channels of Honda DIAG-H protocol

13.3.2.1.5 Return Values

There are no changes defined for the Return Values.

13.3.3 IOCTL Section

13.3.3.1 SET_CONFIG and GET_CONFIG Supported Parameters

The LOOPBACK, P1_MAX, P3_MIN, P4_MIN, J1962_PINS configuration parameters shall be supported for the protocol HONDA DIAG-H protocol. Attempts to access any other parameters shall result in return of the ERR_INVALID_IOCTL_PARAM_ID error code.

P1_MAX expiry shall be used to detect the end of an ECU response message.

The interface shall not check for P2_MAX violations. All received messages shall be queued to the application regardless of the time since the end of the last transmission by the interface. Transmissions from the interface shall not be allowed within P3_MIN since the end of a received message.

13.4 Message Structure

13.4.1 C/C++ Definition

There is no change to the C/C++ definition.

13.4.2 Elements

There is no change to any of the elements.

13.4.3 Message Data Formats

When using HONDA_DIAGH_PS, all message bytes are defined by the application.

Honda DIAG-H Protocol IDs and message limitations have been defined in Table 48.

Table 48 - Allowed message sizes per protocol

Protocol ID	Min Tx	Max Tx	Min Rx	Max Rx	Notes
HONDA_DIAGH_PS or HONDA_DIAGH_CHx	3	255	1	255	Data[0 - n] = Data bytes

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID.

13.4.4 Format Checks for Messages Passed to the API

The vendor DLL shall validate all PASSTHRU_MSG structures, and return an error, ERR_INVALID_MSG if DataSize violates Min Tx or Max Tx columns in “Message Data Formats” section.

13.4.5 Message Flag and Status Definitions

Only the TX_MSG_TYPE RxStatus bit shall be used by this protocol.

No TxFlags or Indications are supported for this protocol. These flags are undefined for this protocol.

13.5 Discovery Support

The device shall report support for HONDA DiagH Protocol and associated parameters through the discovery mechanism defined in Section 25.

14. REPEAT MESSAGING

14.1 Scope of the Repeat Messaging Protocol Optional Feature

This section details the extensions to an SAE J2534-1 interface that will allow support of the Repeat Messaging feature. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed from SAE J2534-1.

14.2 Win32 Application Programming Interface

14.2.1 API Functions - Overview

Feature Implementation Summary:

Some operations on ECUs require a message to be transmitted repeatedly at a rapid and predictable rate until the operation is completed. Completion of the operation is indicated by a change in a byte value in the response message to the repeated request.

This optional feature defines an extension to the API that allows the interface to perform autonomous transmissions based on the content of messages received from a vehicle bus. The configuration of this behavior is performed by the application using new IOCTLs. In this way, the feature provides a general purpose, protocol-independent method of achieving high performance repeated transmissions while certain conditions persist on a vehicle.

Support of this feature is achieved by three additional PassThruIoctl IoctlID values. The first is used to set-up and start the repeat transmission, the second is used to query the status of a transmission, and the third is used to stop the transmission and release resources. The device shall support a minimum of ten repeat messages per channel. Additionally, this feature shall be supported on all protocols unless explicitly excluded by the description of the protocol.

The operation of the feature is in three stages:

1. Specifying the message to transmit, transmission rate and stop criteria via PassThruIoctl START_REPEAT_MESSAGE.

2. Querying to see if the transmission has been stopped automatically via PassThruIoctl QUERY_REPEAT_MESSAGE.

3. Stopping an in-progress transmission and releasing resources via PassThruIoctl STOP_REPEAT_MESSAGE.

In general, repeat messages are almost identical to periodic messages. Repeat messages shall have the same DataSize and TxFlags limitations as periodic messages. Repeat messages shall generate loopback messages, if enabled on the channel. Additionally, repeat messages shall have the same priority as periodic messages. The queueing mechanism used by periodic messages shall be extended to include repeat messages. Finally, repeat messages shall not violate bus idle timing parameters (like P3_MIN) or other protocol specific requirements.

Responses to the repeat messages, including the response that caused termination of the repeat transmission, shall be treated in the same way as other received data and shall be queued to the application, subject to any filters configured on the channel. Incoming messages will be evaluated against the repeat mask/pattern before being processed by the standard Pass or Block filtering.

API Change List:

Table 49 summarizes the changes to the SAE J2534-1 API functions.

Table 49 - SAE J2534 API functions

Function	Description of Change
PassThruIoctl	New IOCTL IDs are added: START_REPEAT_MESSAGE QUERY_REPEAT_MESSAGE STOP_REPEAT_MESSAGE

14.2.2 IOCTL Section

Table 50 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr Represents	Purpose
START_REPEAT_MESSAGE	Pointer to REPEAT_MSG_SETUP	Pointer to unsigned long	Sets up and starts a repeat message transmission.
QUERY_REPEAT_MESSAGE	Pointer to unsigned long	Pointer to unsigned long	Reports the status of a repeat message transmission.
STOP_REPEAT_MESSAGE	Pointer to unsigned long	NULL pointer	Terminates an existing repeat message transmission.

14.2.2.1 START_REPEAT_MESSAGE

This function is used to specify and initiate a repeat message. The parameters are specified in Table 51.

Table 51 - START_REPEAT_MESSAGE details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect.
IoctlID	Is set to the define START_REPEAT_MESSAGE.
InputPtr	Is a pointer to the structure REPEAT_MSG_SETUP which is defined as follows: typedef struct { unsigned long TimeInterval; /* Repeat transmission rate */ unsigned long Condition; PASSTHRU_MSG RepeatMsgData[3]; } REPEAT_MSG_SETUP; where: TimeInterval is the time interval between the end of a message transmission and the start of the next time this message is sent, in milliseconds. The valid range is 5 to 65535 ms. Condition specifies whether the repeat transmission should: 0 = continue until a message is received that matches the pattern 1 = continue while every received message matches the pattern RepeatMsgData[0] Is a PASSTHRU_MSG structure that specifies the message to be repeatedly transmitted. The Data, DataSize, and TxFlags elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. RepeatMsgData[1] is a PASSTHRU_MSG structure that specifies the mask message that will be ANDed with each incoming message to mask any unimportant bits. The Data, DataSize, and TxFlags elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. RepeatMsgData[2] is a PASSTHRU_MSG structure that specifies the pattern message that will be compared to the incoming message after the mask message has been applied. If the result matches this pattern message, the designated action shall be taken. The Data, DataSize, and TxFlags elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. Additionally, the DataSize and TxFlags elements shall match that of the mask message.
OutputPtr	Is a pointer to an unsigned long, MsgId, the message ID that is assigned by the DLL.

If this function call is successful, the return value shall be STATUS_NOERROR and the associated repeat message shall have been created (even if it duplicates an existing repeat message) and one copy shall be placed in the repeat message transmit queue.

If Condition = 0 (REPEAT_MESSAGE_UNTIL_MATCH), the repeat message transmissions shall continue until a message is received that, after ANDing with the mask message, matches the pattern message exactly. Additionally, repeat message transmissions shall continue if the TimeInterval expires without receiving any message.

If Condition = 1 (REPEAT_MESSAGE_WHILE_MATCH), the repeat message transmissions shall continue while every received message that, after ANDing with the mask message, matches the pattern message exactly. The repeat message transmissions shall cease when a message is received that, after ANDing the with the mask message, doesn't match the pattern exactly. Additionally, repeat message transmissions shall terminate if the TimeInterval expires without receiving any message.

ERR_EXCEEDED_LIMIT shall be returned if more than the maximum number of repeat messages supported by the interface is requested. ERR_INVALID_MSG shall be returned if the data size of an individual message exceeds that defined in SAE J2534-1 for periodic messages. ERR_INVALID_MSG shall also be returned if the TxFlags do not match those specified in SAE J2534-1 for periodic messages.

If this IOCTL is not supported by the interface an ERR_NOT_SUPPORTED shall be returned.

NOTE: Only the first RepeatMsgData[1].DataSize bytes of each incoming message shall be evaluated (i.e., additional bytes in a message longer than DataSize shall be ignored and treated as "don't care"). Incoming messages whose DataSize is less than that of the pattern message shall be considered not to match (even if the mask bytes indicated "don't care").

Examples:

Application sets up a repeat message:
0x64 0x28 0xF5 0x22 0x02 0x00

Normal ECU response is:
0x74 0xF5 0x28 0x62 0x02 0x00 0x00

The application requires the transmission to terminate when the following is received:
0x74 0xF5 0x28 0x62 0x02 0x00 0x01

Example #1:

The filtering arrays could be set up as:

Condition	0 (REPEAT_MESSAGE_UNTIL_MATCH)
Mask	0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
Pattern	0x74 0xF5 0x28 0x62 0x02 0x00 0x01

This would require all bytes in the received message to match the pattern specified above for transmission to stop.

If there were a sequence number that changed in the first byte every time, the following could be used to ignore the value of the changing sequence number byte:

Condition	0 (REPEAT_MESSAGE_UNTIL_MATCH)
Mask	0x00 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
Pattern	0x00 0xF5 0x28 0x62 0x02 0x00 0x01

Example #2:

If there are several responses that are required to cause the transmission to terminate, for example:

0x74 0xF5 0x28 0x62 0x02 0x00 0x01

0x74 0xF5 0x28 0x62 0x02 0x00 0x52

0x74 0xF5 0x28 0x62 0x02 0x00 0x81

Then the following setup could be used to continue transmission while the Normal ECU response alone continues to be received:

Condition	1 (REPEAT_MESSAGE_WHILE_MATCH)
Mask	0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF
Pattern	0x74 0xF5 0x28 0x62 0x02 0x00 0x00

14.2.2.2 QUERY_REPEAT_MESSAGE

This function is used to check the status of the designated repeat message. The parameters are specified in Table 52.

Table 52 - QUERY_REPEAT_MESSAGE details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect
IoctlID	Is set to the define QUERY_REPEAT_MESSAGE
InputPtr	Is a pointer to an unsigned long, MsgId, the message ID of the repeat message transmission to be queried
OutputPtr	Is a pointer to an unsigned long status, the current status of the specified repeat message transmission

If this function call is successful, the return value shall be STATUS_NOERROR and the value of the status parameter shall be set as shown in Table 53.

Table 53 - Status parameter values

Condition	Status Value
Repeat message transmission in progress, filter criteria not yet met	TRUE (1)
Filter criteria met and repeat transmission ceased	FALSE (0)

If this function is called with an invalid MsgID, ERR_INVALID_MSG_ID shall be returned.

14.2.2.3 STOP_REPEAT_MESSAGE

This function is used to terminate a defined repeat transmission and release the specified MsgID. The parameters are specified in Table 54.

Table 54 - STOP_REPEAT_MESSAGE details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect
IoctlID	Is set to the define STOP_REPEAT_MESSAGE
InputPtr	Is a pointer to an unsigned long, MsgId, the message ID of the repeat message transmission to be stopped
OutputPtr	Is a NULL pointer, as this parameter is not used

If this function call is successful the return value shall be STATUS_NOERROR, the MsgID shall be invalid, the interval timers shall be stopped, and the associated repeat message shall be removed from the transmit queue (if one is present). This function will not terminate the associated repeat message if it is currently being sent.

It should be noted that the MsgID shall not be released automatically on termination of the repeat transmission due to receipt of a message matching the filter conditions. Therefore, a call to STOP_REPEAT_MESSAGE shall always be required.

If this function is called with an invalid MsgID, ERR_INVALID_MSG_ID shall be returned.

14.3 Discovery Support

The device shall report support for Repeat Messaging and associated parameters through the discovery mechanism defined in Section 25.

15. EXTENDED PROGRAMMING VOLTAGE SUPPORT

15.1 Scope of the Extended Programming Voltage Feature

This section details the additional programming voltage features supported by SAE J2534-2.

15.2 Pass-Thru System Requirements

15.3 Win32 Application Programming Interface

15.3.1 API Functions - Overview

Table 55 summarizes the changes to the SAE J2534-1 API function.

Table 55 - SAE J2534 API functions

Function	Description
PassThruSetProgrammingVoltage	Support Short-to-Ground feature on pin 9

15.3.2 API Functions - Detailed Information

15.3.2.1 PassThruSetProgrammingVoltage

In addition to pin 15 supported in SAE J2534-1, Short-to-Ground will also be allowed on pin 9. Pin 9 Short-to-Ground feature is not mutually exclusive with the other programming pins and can be used while supplying programming voltage through other supported pins.

Short-to-Ground feature shall be supported on only one pin at a time.

ERR_PIN_IN_USE shall be returned if:

- An attempt is made to set a programming voltage on pin 9 while it is at ground.
- An attempt is made to ground pin 9 while it has a programming voltage applied.

ERR_VOLTAGE_IN_USE shall be returned if:

- An attempt is made to ground pin 9 while pin 15 is grounded.
- An attempt is made to ground pin 15 while pin 9 is grounded.

ERR_NOT_SUPPORTED shall be returned if:

- The device does not support pin 9 Short-to-Ground feature.

15.4 Discovery Support

The device shall report Extended Programming Voltage Support and associated parameters through the discovery mechanism defined in Section 25.

16. SAE J1939 PROTOCOL

16.1 Scope of the SAE J1939 Protocol Optional Feature

This section details the extensions to SAE J2534-1 that will allow support of the SAE J1939 protocol. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed.

This section specifically defines the common method of supporting the SAE J1939 heavy-duty vehicle protocol as defined in SAE J1939.

The SAE J2534-2 implementation will implement single frame messages and multiple frame messages using both the SAE J1939 BAM process and the Connection Management message process. Refer to SAE J1939-21 for details on multi-frame messages. In addition, the SAE J2534-2 device will support the Claim SAE J1939 Address Process outlined in SAE J1939-81.

The BAM message process will be used when the SAE J2534-2 device has a claimed address and a message to be transmitted or received has more than 8-data bytes and is being sent to or received from the Global Address (0xFF). To send a message using the BAM process, fill in the Destination Address field with the Global Address.

The Connection Management message process will be used when the SAE J2534-2 device has a claimed address and a message with more than 8-data bytes is being sent to a specific destination address or a message is being sent from the vehicle to the SAE J2534-2 device's claimed address. To send a message using the Connection Management process, fill in the Destination Address field with a specific address.

16.2 Pass-Thru System Requirements

16.2.1 Connection to Vehicle

Vendors will have to provide a custom cable to connect to an SAE J1939 vehicle.

NOTE: Some vehicles that use SAE J1939 operate with 24-V electrical systems. This SAE J2534-2 optional feature does not specify changes to the SAE J2534-1 device, thus it is up to the customer and SAE J2534-2 device supplier to insure no damage is done to the SAE J2534-2 device when used with 24-V systems.

16.2.2 Communication Protocol

The following features of SAE J1939 must be supported by the PassThru device:

a. 250 kbps

b. 29-bit identifiers

c. BAM and Connection Management message transfers

d. Protection of ten source addresses and the name associated with each source address

16.2.3 Simultaneous Communication on Multiple Protocols

The SAE J1939 implementation will have the same simultaneous communication on multiple protocol requirements as the CAN protocol (SAE J1850 and ISO9141 must be able to run simultaneously).

16.3 Win32 Application Programming Interface

16.3.1 API Functions - Overview

Feature Implementation Summary:

At a high level, a new ProtocolID has been defined to indicate the use of the SAE J1939 physical layer. The protocol defines a communication link between the tester and the electronic control unit (ECU). If the feature is not supported, an error code, ERR_NOT_SUPPORTED, will be returned by the call to PassThruConnect. The calling application will be required to notify the user that this optional feature may not be supported by interface.

When the SAE J1939 protocol is opened, the application must use the CAN_29BIT_ID connect flag and set the baud rate to 250K.

API Change List:

Table 56 summarizes the changes to the SAE J2534-1 API functions.

Table 56 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.
PassThruWriteMsgs	Added a new return value.
PassThruIoctl	New IO control sub-function and 5 configuration parameters.

16.3.2 API Functions - Detailed Information

16.3.2.1 PassThruConnect

When PassThruConnect is called with the J1939_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

16.3.2.1.1 Connect Flag Values

There are no changes to the connect flag values.

Only CAN_29BIT_ID shall be supported.

16.3.2.1.2 ProtocolID Values

Only the definition and description of the ProtocolID value is defined Table 57. The actual value is defined in Section 24.

Table 57 - Protocol ID values

Definition	Description
J1939_PS	SAE J1939 protocol with pin selection
J1939_CHx	Additional channels of SAE J1939 protocol

16.3.2.2 PassThruWriteMsgs

Added a new return value.

16.3.2.2.1 Return Values

Table 58 shows the new return value for PassThruWriteMsgs.

Table 58 - Return values

Definition	Description
ERR_ADDRESS_NOT_CLAIMED	Returned when a write message has a data payload larger than 8-data bytes and the source address in the message does not match a current claimed address

16.3.2.3 PassThruIoctl

16.3.2.3.1 IOCTL ID Values

The new IOCTL value PROTECT_J1939_ADDR is added to support the claiming of an SAE J1939 CAN address for the tester (SAE J2534 device).

16.3.3 IOCTL Section

The details on the new IOCTLs available through the PassThruIoctl function are shown in Table 59.

Table 59 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr Represents	Purpose
PROTECT_J1939_ADDR	Points to SBYTE_ARRAY	NULL pointer	To direct the pass-thru device to claim and defend the specified address

16.3.3.1 Configuration Parameters

16.3.3.1.1 SET_CONFIG and GET_CONFIG Supported Parameters

The DATA_RATE, LOOPBACK, BIT_SAMPLE_POINT, and SYNC_JUMP_WIDTH configuration parameters will be supported for the SAE J1939 protocol. The default values for SAE J1939 are LOOPBACK = 0, BIT_SAMPLE_POINT = 87.5%, and SYNC_JUMP_WIDTH = 6.25% (1Tq).

16.3.3.1.2 Additional SET_CONFIG and GET_CONFIG Parameters

Table 60 shows the five new configuration parameters will be added to support the SAE J1939 protocol.

Table 60 - IOCTL GET_CONFIG/SET_CONFIG parameters details

Parameter	Valid Values for Parameter	Default Value (decimal)	Description
J1939_T1	0X0 - 0X0000FFFF (1 ms per bit)	750	For protocol SAE J1939, this sets the maximum time the device waits for the next data frame in a multi-frame transfer.
J1939_T2	0X0 - 0X0000FFFF (1 ms per bit)	1250	For protocol SAE J1939, this sets the maximum time a receiver will wait for the next data frame after sending a CTS in a multi-frame transfer.
J1939_T3	0X0 - 0X0000FFFF (1 ms per bit)	1250	For protocol SAE J1939, this sets the maximum time the device waits for CTS in a multi-frame transfer.
J1939_T4	0X0 - 0X0000FFFF (1 ms per bit)	1050	For protocol SAE J1939, this sets the maximum time the device waits for the next CTS after receiving a CTS with a request for zero frames in a multi-frame transfer.
J1939_BRDCST_MIN_DELAY	0X0 - 0X0000FFFF (1 ms per bit)	50	For protocol SAE J1939, this sets the minimum time between data frames in a multi-frame broadcast transmission.

16.3.3.2 PROTECT_J1939_ADDR

The IoctlID parameter of PROTECT_J1939_ADDR is used to set the SAE J1939 source address to defend and to assign the PassThru device's SAE J1939 NAME as defined in SAE J1939-81. The PassThru device must be able to protect ten source addresses.

Table 61 - Protect_J1939_addr details

Parameter	**Description**
ChannelID	Channel ID assigned by DLL during PassThruConnect
IoctlID	Is set to PROTECT_J1939_ADDR

InputPtr	Points to the structure SBYTE_ARRAY, which is defined as follows:
	
	Typedef struct
	{
	 unsigned long NumOfBytes; /* number of bytes in array */
	 unsigned char *BytePtr; /* array of bytes */
	}
	
	where:
	
	NumOfBytes is an input that indicates the number of bytes in the array BytePtr. It should always be 9 for PROTECT_J1939_ADDR.
	
	BytePtr[0] is the source address to claim (0 - 253), 254 is NOT allowed, 255 = is the default and means no address is claimed.
	
	BytePtr[1] - BytePtr[8] is the SAE J1939 defined name. BytePtr[1] is the LSB as defined in SAE J1939-81 (the LSB of the Identity Number).

OutputPtr	NULL pointer

This call is non-blocking. When an address claim is successful a J1939_ADDRESS_CLAIMED indication will be received. A J1939_ADDRESS_LOST indication will be received if the address claim was unsuccessful.

If BytePrt[0] equals 254 or 255, the function will return ERR_INVALID_IOCTL_VALUE.

To cancel a protected address, BytePrt[0] equals the source address you want canceled and all the name values (BytePrt1 through BytePtr[8]) shall all be zero.

16.4 Message Structure

16.4.1 C/C++ Definition

There is no change to the C/C++ definition.

16.4.2 Elements

There is no change to any of the elements.

16.4.3 Message Data Formats

When using the SAE J1939 protocol, the first 5 bytes of data contain the 29-bit CAN ID used in SAE J1939 and the destination address. Data[0] contains CAN ID bits 28-24 (the three most significant bits will be zero), Data[1] contains bits 23-16, Data[2] contains bits 15-8, Data[3] contains bits 7-0, and Data[4] contains the destination address. It is up the application to pack this data correctly (as specified in SAE J1939-21). Additional Protocol IDs and message limitations have been defined in Table 62.

Table 62 - Allowed message sizes per protocol

Protocol ID	Min Tx	Max Tx	Min Rx	Max Rx	Notes
J1939_PS or J1939_CHx	5	1790	5	1790	4 bytes of CAN ID, 1-byte destination address followed by up to 1785 data bytes

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID.

16.4.4 Format Checks for Messages Passed to the API

For messages with less than 9-data bytes, the destination address (Data[4]) is a “don’t care” and is not used in sending the message.

For messages with more than 8-data bytes, if the SAE J2534-2 device has a claimed address and the destination address (Data[4]) is the global address (0xFF), the BAM message process will be used to send the message.

For messages with more than 8-data bytes, if the SAE J2534-2 device has a claimed address and the message is being sent to a specific destination address (Data[4]), the Connection Management message process will be used to send the message.

16.4.5 Conventions for Returning Messages from the API

Received messages with less than 9-data bytes, the destination address (Data[4]) will be filled in with the specific destination address of the message or the global address.

Received messages greater than 8 bytes will be returned to the API when the SAE J2534-2 device has a claimed address and a BAM message is received. Data[4] will contain the global address.

Received messages greater than 8 bytes will be returned to the API when the SAE J2534-2 device has a claimed address and a Connection Managed message was sent to the SAE J2534-2 device’s claimed address. Data[4] will contain the claimed address that the message was received on.

16.4.6 Message Flag and Status Definitions

Supported Message Status (RxStatus) Definitions.

The TX_MSG_TYPE and CAN_29BIT_ID RxStatus bits will be supported. For SAE J1939, loopback messages returned by the API will have TX_MSG_TYPE set in RxStatus. Messages with a 29-bit ID field will have CAN_29BIT_ID set.

There are two additions to the Message Status (RxStatus) definitions to be used as indications.

Table 63 - RxStatus bit definitions

Definition	RxStatus Bit(s)	Description	Value
J1939_ADDRESS_LOST	17	Address lost status indication will be sent when a claimed address and name has been lost or when an attempt to claim an address and name fails. The source address lost will be in Data[0] of the message. ExtraDataIndex = 0, DataSize = 1. SAE J1939 only.	0 = no address lost indication 1 = address lost or claim attempt failed
J1939_ADDRESS_CLAIMED	16	Address claimed status indication will be sent when an address and name is successfully claimed. The source address claimed will be in Data[0] of the message. ExtraDataIndex = 0, DataSize = 1. SAE J1939 only.	0 = no address claimed indication 1 = address claimed was successful

The CAN_29BIT_ID TxFlags bit is the only TxFlags bit supported. When sending a message with a 29-bit ID field, set the TxFlags bit CAN_29BIT_ID.

16.5 Return Value Error Codes

ERR_ADDRESS_NOT_CLAIMED will be returned from PassThruWriteMsgs when a message has a data payload larger than 8-data bytes and the source address in the message does not match a current claimed address.

16.6 Discovery Support

The device shall report support for the SAE J1939 protocol and associated parameters through the discovery mechanism defined in Section 25.

17. SAE J1708 PROTOCOL

17.1 Scope of the SAE J1708 Protocol Optional Feature

This section details the extensions to SAE J2534-1 that will allow support of the SAE J1708 protocol. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed.

17.2 Pass-Thru System Requirements

17.2.1 Connection to Vehicle

Vendors will have to provide a custom cable to connect to an SAE J1708 vehicle.

NOTE: Some vehicles that use SAE J1708 operate with 24-V electrical systems. This SAE J2534-2 optional feature does not specify changes to the SAE J2534-1 device, thus it is up to the customer and the SAE J2534-2 device supplier to ensure no damage is done to the SAE J2534-2 device when used with 24-V systems.

17.2.2 Communication Protocol

The following features of SAE J1708 must be supported by the PassThru device:

- 9600 bps
- Priority message delay

17.2.3 Simultaneous Communication on Multiple Protocols

The SAE J1708 protocol implementation will have the same simultaneous communication on multiple protocol requirements as the ISO9141 protocol (SAE J1850 and CAN must be able to run simultaneously with SAE J1708).

17.3 Win32 Application Programming Interface

17.3.1 API Functions - Overview

Feature Implementation Summary:

Support for the SAE J1708 protocol is achieved by the addition of a new physical layer interface and a new ProtocolID. In addition, the SAE J2534-2 device vendor may provide a heavy truck cable or adapter.

The protocol feature of calculating and appending the message checksum is required.

The protocol feature of priority message delay for transmitted messages will be implemented by passing the priority message delay to the PassThru device in the PassThru message TxFlags.

Refer to SAE J1708 for details of communication interface byte encoding and the electrical interface design and characteristics.

API Change List:

Table 64 summarizes the changes to the SAE J2534-1 API functions.

Table 64 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values

17.3.2 API Functions - Detailed Information

17.3.2.1 PassThruConnect

When PassThruConnect is called with the J1708_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

17.3.2.1.1 Connect Flag Values

The SAE J1708 protocol uses the SAE J2534-1 connect flag shown in Table 65.

Table 65 - Connect flag values

Definition	Description
CHECKSUM_DISABLED	0 = The interface will generate and append the checksum as defined in SAE J1708 for transmitted messages. The interface will verify the checksum for received messages and place it in the data section pointed to by ExtraDataIndex. 1 = The interface will not generate and verify the checksum. The entire message will be treated as data by the interface.

17.3.2.1.2 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 66. The actual value is defined in Section 24.

Table 66 - Protocol ID values

Definition	Description
J1708_PS	SAE J1708 protocol with pin selection
J1708_CHx	Additional channels of SAE J1708 protocol

17.3.2.2 Configuration Parameters

17.3.2.2.1 SET_CONFIG and GET_CONFIG Supported Parameters

The DATA_RATE and LOOPBACK configuration parameters will be supported for the SAE J1708 protocol. The default value for LOOPBACK will be zero.

17.4 Message Structure

17.4.1 C/C++ Definition

There is no change to the C/C++ definition.

17.4.2 Elements

There are no changes to any of the elements.

17.4.3 Message Data Formats

When transmitting messages, the Transmit Message Priority will be encoded in four TxFlags bits. The SAE J2534-2 device will calculate the check sum and send it after the last data byte of the message unless the CHECKSUM_DISABLED connect flag was set on the PassThruConnect, in which case the application is responsible for calculating the check sum and placing it as the last data byte of the message. (Refer to SAE J1708 for instructions on how to calculate the check sum.)

For received messages, when the CHECKSUM_DISABLED flag is not set, the check sum will be calculated as per SAE J1708 and compared to the last byte in the data stream. If the check sum is correct, the message will be place in the receive queue with ExtraDataIndex pointing to the check sum. If the check sum does not verify, the message will be discarded. If CHECKSUM_DISABLED was set on the PassThruConnect, all the data received on the bus will be treated as message data.

Table 67 - Allowed message sizes per protocol

Protocol	Min Tx	Max Tx	Min Rx	Max Rx	Notes
J1708_PS or J1708_CHx	1	4095	1	4095	

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID.

17.4.4 Conventions for Returning Messages from the API

For the SAE J1708 protocol, loopback messages returned by the API will have TX_MSG_TYPE set in RxStatus.

17.4.5 Message Flag and Status Definitions

Changes to TxFlags bits:

Table 68 - TxFlags bit definitions

Definition	TxFlags Bit(s)	Description	Value
MSG_PRIORITY_VALUE	16 -19	Message priority	Valid values 1 thru 8 Value of 0 or greater than 8 will be treated as priority 8

If the TX message is looped back to the application MSG_PRIORITY_VALUE must be preserved in the looped back message. If MSG_PRIORITY_VALUE is not valid the device will use the lowest priority (a value of 8) for that message.

The TX_MSG_TYPE is the only RxStatus flag supported in SAE J1708. It will be set on all loop back TX messages.

17.5 Discovery Support

The device shall report support for the SAE J1708 protocol and associated parameters through the discovery mechanism defined in Section 25.

18. EXTENDED PASSTHRUIOCTL FOR DEVICE CONFIGURATION PARAMETERS

18.1 Scope of the Extended PassThruIoctl Optional Feature

This section details the extensions to an SAE J2534-1 interface that will allow setting and retrieval of device-wide configuration parameters. Currently, this feature is limited to support of ten non-volatile and re-writable data storage parameters. However, the feature may be extended in future to support additional non-protocol-specific parameters. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed from SAE J2534-1.

18.2 Pass-Thru Concept

This optional feature defines an extension to the pass-thru concept that allows device-wide parameters to be specified. More specifically, the current definition is for ten data items to be stored on the PassThru interface in a non-volatile storage area (i.e., an area in which data survives disconnection from a power source). Each data item will be 4-bytes long.

18.3 Win32 Application Programming Interface

18.3.1 API Functions - Overview

The PassThruIoctl called with GET_DEVICE_CONFIG and SET_DEVICE_CONFIG IoctlIDs are used to set device parameters and require the DeviceID to be passed as the first PassThruIoctl parameter.

Table 69 summarizes the changes to the SAE J2534-1 API function.

Table 69 - SAE J2534 API functions

Function	Description
PassThruIoctl	GET_DEVICE_CONFIG and SET_DEVICE_CONFIG are added. Parameters NON_VOLATILE_STORE_1 to NON_VOLATILE_STORE_10 are added.

18.4 IOCTL Section

Table 70 provides details of the additional IoctlIDs that support this feature.

Table 70 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr Represents	Purpose
GET_DEVICE_CONFIG	Pointer to SCONFIG_LIST	NULL pointer	To get the configuration or status of the pass-thru device.
SET_DEVICE_CONFIG	Pointer to SCONFIG_LIST	NULL pointer	To set the configuration or status of the pass-thru device.

18.4.1 GET_DEVICE_CONFIG

The IoctlID value of GET_DEVICE_CONFIG is used to obtain the configuration or status of the pass-thru device. The calling application is responsible for allocating and initializing the associated parameters described in Table 71. When the function is successfully completed, the corresponding parameter value(s) indicated in Table 73 will be placed in each value.

Table 71 - GET_DEVICE_CONFIG details

Parameter	Description
DeviceIID	Device ID assigned by DLL during PassThruOpen.
IoctlID	Is set to the define GET_DEVICE_CONFIG.
InputPtr	Points to the structure SCONFIG_LIST, which is defined as follows: typedef struct { unsigned long NumOfParams; /* number of SCONFIG elements */ SCONFIG *ConfigPtr; /* array of SCONFIG */ } SCONFIG_LIST where: NumOfParms is an INPUT, which contains the number of SCONFIG elements in the array pointed to by ConfigPtr. ConfigPtr is a pointer to an array of SCONFIG structures. The structure SCONFIG is defined as follows: typedef struct { unsigned long Parameter; /* name of parameter */ unsigned long Value; /* value of the parameter */ } SCONFIG where: Parameter is an INPUT that represents the parameter to be obtained (see Table 73 for a list of valid parameters). Value is an OUTPUT that represents the value of that parameter (see Table 73 for a list of valid values).
OutputPtr	Is a NULL pointer, as this parameter is not used.

Requesting other parameters shall result in the Return Value ERR_INVALID_IOCTL_PARAM_ID.

18.4.2 SET_DEVICE_CONFIG

The IoctlID value of SET_DEVICE_CONFIG is used to set the configuration or status of the pass-thru device. The calling application is responsible for allocating and initializing the associated parameters described in Table 72. When the function is successfully completed the corresponding parameter(s) and value(s) indicated in Table 73 will be in effect.

Table 72 - SET_DEVICE_CONFIG details

Parameter	Description
DeviceIID	Device ID assigned by DLL during PassThruOpen.
IoctlID	Is set to the define SET_DEVICE_CONFIG.
InputPtr	Points to the structure SCONFIG_LIST, which is defined as follows: typedef struct { unsigned long NumOfParams;/* number of SCONFIG elements */ SCONFIG *ConfigPtr; /* array of SCONFIG */ } SCONFIG_LIST where: NumOfParms is an INPUT, which contains the number of SCONFIG elements in the array pointed to by ConfigPtr. ConfigPtr is a pointer to an array of SCONFIG structures. The structure SCONFIG is defined as follows: typedef struct { unsigned long Parameter; /* name of parameter */ unsigned long Value; /* value of the parameter */ } SCONFIG where: Parameter is an INPUT that represents the parameter to be set (see Table 73 for a list of valid parameters). Value is an INPUT that represents the value of that parameter (see Table 73 for a list of valid values).
OutputPtr	Is a NULL pointer, as this parameter is not used.

Setting other parameters shall result in the Return Value ERR_INVALID_IOCTL_PARAM_ID.

Table 73 - IOCTL GET_DEVICE_CONFIG/SET_DEVICE_CONFIG parameter details

Parameter	Valid Values for Parameter	Default Value (decimal)	Description
NON_VOLATILE_STORE_1	0x00000000 - 0xFFFFFFFF	0	First 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_2	0x00000000 - 0xFFFFFFFF	0	Second 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_#	0x00000000 - 0xFFFFFFFF	0	Third 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_4	0x00000000 - 0xFFFFFFFF	0	Fourth 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_5	0x00000000 - 0xFFFFFFFF	0	Fifth 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_6	0x00000000 - 0xFFFFFFFF	0	Sixth 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_7	0x00000000 - 0xFFFFFFFF	0	Seventh 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_8	0x00000000 - 0xFFFFFFFF	0	Eighth 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_9	0x00000000 - 0xFFFFFFFF	0	Ninth 4-byte non-volatile data storage location.
NON_VOLATILE_STORE_10	0x00000000 - 0xFFFFFFFF	0	Tenth 4-byte non-volatile data storage location.

Notes for NON_VOLATILE_STORE parameters:

1. The SET_DEVICE_CONFIG operation shall complete, and the DLL shall return from the PassThruIoctl call within 10 seconds.

2. Parameter values shall survive disconnection from all power sources.

3. Parameter values shall not be affected by an interface device firmware programming operation.

4. Parameter values shall persist for a minimum of 6 months after each write operation.

Usage Guideline

A typical SAE J2534 interface may have a limit on the number of writes to non-volatile memory. These parameters are intended to be SET infrequently. The number of write cycles can be minimized by writing as many parameters as possible in one SET_DEVICE_CONFIG call.

18.5 Discovery Support

The device shall report support for Extended PassThruIoctl for Device Configuration through the discovery mechanism defined in Section 25.

19. TP2.0 PROTOCOL

19.1 Scope of the TP2.0 Protocol Optional Feature

This section details the extensions to an SAE J2534-1 interface that will allow support of the TP2.0 protocol. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed from SAE J2534-1.

19.2 Pass-Thru System Requirements

19.2.1 Simultaneous Communication on Multiple Protocols

Support for simultaneous communication of TP2.0 with other protocols shall be equivalent to the definition for CAN in SAE J2534-1.

19.2.2 Pin Usage

TP2.0 uses SAE J1962 pins 6 and pin 14. As with all SAE J2534-2 optional protocols, no default pin is identified. Therefore, the application developer will be required to set the pin to be used.

Refer to the SAE J1962 pin selection section of SAE J2534-2 for details of the method used to switch SAE J1962 pins.

19.3 Win32 Application Programming Interface

19.3.1 API Functions - Overview

Feature Implementation Summary:

Some ECUs support the TP2.0 protocol that uses the CAN physical layer interface ISO11898-1:2015 (Classic CAN only).

Refer to SAE J2819 for the detailed protocol information.

Key features of the TP2.0 protocol are the establishment and maintenance of a channel between two nodes. The maintenance of a connection on an established channel and the ability to send broadcast messages that repeat five times at a configurable time spacing.

The guidelines to implement this communication capability to SAE J2534-2 API are shown below:

- The interface shall be able to connect a 500K baud CAN physical layer interface on pins 6 and 14 of the SAE J1962 connector.
- Send a broadcast message repeated five times with the last 2 bytes alternating between 0x55 and 0xAA, with a configurable time spacing between the messages.
- Send a re-trigger broadcast message which starts out as stated above with additional messages sent at a slower configured rate.
- The ability to stop the re-triggered broadcast message.
- The ability to establish and maintain four channels with connections simultaneously.
- Send indication messages to the application when a connection is established or disconnected.
- The interface must fragment and reassemble messages.
- The interface must be able to delay the transmission of the transport packets as specified by T1 and T3 in the TP2.0 protocol. The interface must support a minimum time resolution of 500 μs.

NOTE: Even though the TP2.0 specification can specify a time delay of 100 μs, the PassThru device need only support a minimum delay of 500 μs. The PassThru device should always extend a requested delay time to the nearest 500 μs time interval.

- Buffer one outgoing and one incoming messages as they are being fragmented or reassembled on each channel.
- Handle the Acknowledge with retry and break events.

- The interface shall automatically transmit the Connection Test telegram to keep a connection active.
- The device shall maintain the sequence number as part of the data telegram and data acknowledge telegram for each connection.
- The interface shall pass up to the application any messages received unexpectedly that passes any filter criteria, i.e., outside the bounds of an established connection.
- The interface shall support one connection as a passive device (a module on the network requests a connection with the interface). This passive connection is configurable meaning it can be disabled. A passive connection counts as one of the four connections supported by the interface.

Messages passed to PassThruWriteMsgs by the application will include all message address and data bytes, including the proper CAN address for the device or connection. All received messages shall be queued by the interface for retrieval by calls to PassThruReadMsgs. Data telegrams received on an established connection will be reassembled into a PASSTHRU_MSG that is stored in the receive message queue. The application will be able to retrieve this message with a call to PassThruReadMsgs.

Support for the TP2.0 protocol is achieved by the addition of a new ProtocolID, modified behavior of PassThruStartPeriodicMsg, new PassThruIoctl IoctlIDs, and a new low-level implementation for transmission and reception of sequenced messages. This protocol requires additional API and device software support, but no additional hardware compared with a standard SAE J2534-1 device.

If the TP2.0 protocol is not supported, ERR_NOT_SUPPORTED will be returned by the call PassThruConnect. The calling application will be required to notify the user that this optional feature may not be supported by the interface.

API Change List:

Table 74 summarizes the changes to the SAE J2534-1 API functions.

Table 74 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.
PassThruWriteMsgs	Added new functionality to this function.
PassThruStartPeriodicMsg	Added new functionality to this function.
PassThruIoctl	New configuration parameters are added.

19.3.2 API Functions - Detailed Information

19.3.2.1 PassThruConnect

When PassThruConnect is called with the TP2_0_PS Protocol ID, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

19.3.2.1.1 C/C++ Prototype

There are no changes defined for the function prototype.

19.3.2.1.2 Parameters

There are no changes to the parameters.

19.3.2.1.3 Connect Flag Values

There are no changes to the connect flags.

19.3.2.1.4 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 75. The actual value is defined in Section 24.

Table 75 - Protocol ID values

Definition	Description
TP2_0_PS	TP2.0 protocol with pin selection.
TP2_0_CHx	Additional channels of TP2.0 protocol.

19.3.2.2 PassThruWriteMsgs

When used with the TP2.0 protocol, the PassThruWriteMsgs function will perform the following additional functions.

- If the write message's address matches an established connection address, the message will be sent using the TP2.0 protocol.
- If the write message's address does not match an established connection address and the message size fits in a single CAN message, the message will be sent on the bus.
- If the write message's address does not match an established connection address and the message size is larger than a single CAN message, the error ERR_NO_CONNECTION_ESTABLISHED will be returned.
- If the TxFlag "TP2_0_BROADCAST_MSG" is set in the message and the message contains a valid broadcast address in Data[0] (Data[0] value between 0xF0 and 0xFF), the message will be sent five times using TP2_0_ T_BR_INT as the rate, with the last 2 bytes of the message alternating between 0xAA and 0x55. If Data[0] does not contain a valid broadcast address, the function will return ERR_INVALID_MSG.

19.3.2.3 PassThruStartPeriodicMsg

This function is used to send the re-triggered broadcast messages as well as normal periodic messages. To send the TP2.0 re-triggered broadcast message, populate the message data with the broadcast message, set the TxFlag "TP2_0_BROADCAST_MSG" bit and set the periodic rate to the re-trigger rate desired. This function will immediately send out the message five times using TP2_0_T_BR_INT as the rate, and then send the message at the periodic rate (alternating the last two bytes of the message between 0xAA and 0x55).

19.3.3 IOCTL Section

Table 76 provides the details on the new IOCTL IDs available through the PassThruIoctl function.

Table 76 - IOCTL details

Value of IoctlID	InputPtr Represents	OutputPtr	Purpose
REQUEST_CONNECTION	Points to SBYTE_ARRAY	NULL pointer	To establish a channel and connection.
TEARDOWN_CONNECTION	Points to SBYTE_ARRAY	NULL pointer	To tear down an established connection.

19.3.3.1 SET_CONFIG and GET_CONFIG Additional Parameters

The DATA_RATE, LOOPBACK, BIT_SAMPLE_POINT, and SYNC_JUMP_WIDTH configuration parameters will be supported for the TP2.0 protocol. The default values for TP2.0 are LOOPBACK = 0, BIT_SAMPLE_POINT = 80%, and SYNC_JUMP_WIDTH = 15%.

Ten new configuration parameters will be added to support TP2.0 protocol.

Table 77 - TP2.0 configuration parameters

Parameter	Valid Values	Default Values	Description
TP2_0_T_BR_INT	0X0 - 0XFFFF (1 ms per bit)	20	Broadcast interval: time interval between the five messages of a single broadcast message.
TP2_0_T_E	0X0 - 0XFFFF (1 ms per bit)	100	Maximum time-out waiting for a Channel Acknowledge or Connection Acknowledge telegram.
TP2_0_MNTC	0X0 - 0XFFFF (1 count per bit)	10	Retry count for connection management messages.
TP2_0_T_CTA	0X0 - 0XFFFF (1 ms per bit)	1000	Connection test time out for the active device.
TP2_0_MNCT	0X0 - 0XFFFF (1 count per bit)	5	Retry count for sending connection test messages.
TP2_0_MNTB	0X0 - 0XFFFF (1 count per bit)	5	Maximum number of Not Ready Acknowledge received in on block.
TP2_0_MNT	0X0 - 0XFFFF (1 count per bit)	2	Maximum repeats of acknowledge requests.
TP2_0_T_WAIT	0X0 - 0XFFFF (1 ms per bit)	100	Time to wait before the next transmission when a Not Ready Acknowledge is received.
TP2_0_T1	0X0 - 0XFFFF (1 ms per bit)	100	Time out used when waiting for a response.
TP2_0_T3	0X0 - 0XFFFF (1 ms per bit)	0	Minimum time the SAE J2534 device needs between telegrams sent to it.
TP2_0_IDENTIFER	0X0 or 0X200 - 0X2EF	0	The passive identifier CAN ID is used by the interface to receive requests for a connection. The value of zero disables this feature.
TP2_0_RXIDPASSIVE	0X0 or 0X300 - 0X7FF	0	The passive RX CAN ID used in a passive connection. The value of zero disables this feature.

No other SAE J2534-1 defined configuration parameters are supported with the exception of J1962_PINS.

The TP2_0_IDENTIFIER is used by application to give the interface a CAN ID which will be used to compare with the destination field in the TP2.0 messages. An implicit PASS filter will be started when this value is set between the range of 0x200 and 0x2EF. The implicit PASS filter will be cleared when the address is changed to zero. If a value that is not zero or in the range of 0x200 through 0x2EF is requested, the PassThruIoctl will return ERR_INVALID_IOCTL_VALUE.

If the TP2_0_IDENTIFIER has been set to a valid value and the request for a connection is received from the network, it will be rejected with a 0xD8 "Temporarily no resources are free" response if all four connections are in use, or the TP2_0_RXIDPASSIVE is zero. If the request for connection is received and a connection is available and TP2_0_RXIDPASSIVE is set to a valid value, the connection will be established and the CONNECTION_ESTABLISHED Indication message will be sent to the application. The TP2_0_RXIDPASSIVE will be the first four data bytes Data[0] through Data[3] of the indication message and the TX-ID-P address from the network device will be the next four data bytes Data[4] through Data[7], Data[4] is MSB.

19.3.3.2 REQUEST_CONNECTION

The IoctlID parameter of REQUEST_CONNECTION is used to request the establishment of a channel and connection between the SAE J2534 device and an ECU.

Table 78 - REQUEST_CONNECTION details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PASSTHRUCONNECT
IoctlID	Is set to REQUEST_CONNECTION

InputPtr	Points to the structure SBYTE_ARRAY, which is defined as follows:
	
	Typedef struct
	{
	 unsigned long NumOfBytes; /* number of bytes in array */
	 unsigned char *BytePtr; /* array of bytes */
	}
	
	where:
	
	NumOfBytes is an input that indicates the number of bytes in the array BytePtr. It should always be 11 for REQUEST_CONNECTION.
	
	BytePtr[0] through BytePtr[3] is the CAN address (identifier).
	
	BytePtr[0] is MSB.
	
	BytePtr[4] is the destination.
	
	BytePtr[5] is the opcode (0xC0).
	
	BytePtr[6] through BytePtr[7] is the TX-ID-A information.
	
	BytePtr[8] through BytePtr[9] is the RX-ID-A information.
	
	BytePtr[10] is the application type.

OutputPtr	NULL pointer

The data contained in the BytePtr array is the equivalent of the CAN message sent to Request the Channel. The BytePtr data will be sent on the network to try and start a connection. If the RX-ID-A is already in use by another established channel, the error ERR_NOT_UNIQUE will be returned. When a connection is successfully established, an implicit PASS filter will be created such that all messages received on the channel from CAN ID RX-ID-A will be available to the application. The implicit PASS filter will consume one of the minimum of ten filters required by SAE J2534-1.

NOTE: With four open channels, a total of four filters will be consumed as implicit PASS filters.

The REQUEST_CONNECTION IOCTL call is non-blocking which means the API will not wait for the connection to be made. A CONNECTION_ESTABLISHED indication will be placed in the RX queue indicating a successful Connection Request. A CONNECTION_LOST indication will be placed in the RX queue to indicate a failed Connection Request. The RX-ID-A specified in the REQUEST_CONNECTION will be the CAN address of the indication message.

If the Data in the BytePtr array is ill-formatted, or the NumOfBytes is not 11, the function will return with the error ERR_INVALID_IOCTL_VALUE.

19.3.3.3 TEARDOWN_CONNECTION

The IoctlID parameter of TEARDOWN_CONNECTION is used to tear down an established channel and connection between the SAE J2534 device and an ECU.

Table 79 - TEARDOWN_CONNECTION details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect
IoctlID	Is set to TEARDOWN_CONNECTION

InputPtr	Points to the structure SBYTE_ARRAY, which is defined as follows:
	
	Typedef struct
	{
	 unsigned long NumOfBytes; /* number of bytes in array */
	 unsigned char *BytePtr; /* array of bytes */
	}
	
	where:
	
	NumOfBytes is an input that indicates the number of bytes in the array BytePtr. It must always be 4 for TEARDOWN_CONNECTION.
	
	BytePtr[0] through BytePtr[3] is the receiving CAN address, BytePtr[0] is MSB.

OutputPtr	NULL pointer

The data contained in the BytePtr array is the CAN address that the SAE J2534-2 device is receiving CAN messages for this connection. This is the same address as the RX-ID-A in the REQUEST_CONNECTION call. The implicit pass filter for this connection will be removed once the connection has been torn down.

NOTE: All filters set up with PassThruStartMsgFilter will remain even if they match the ID used in the connection.

The TEARDOWN_CONNECTION IOCTL call is non-blocking which means the API will not wait for the connection to be torn down. A CONNECTION_LOST indication will be placed in the RX queue indicating the success of the connection tear down and the implicit PASS filter will be removed.

If the data in the BytePtr array does not match an established connection or the NumOfBytes is not four, then the function will return with the error ERR_INVALID_IOCTL_VALUE.

19.4 Message Structure

19.4.1 C/C++ Definition

There is no change to the C/C++ definition.

19.4.2 Message Data Formats

All received messages that comply with the timing and message structure requirements shall be queued to the application.

Additional Protocol IDs and message limitations have been defined in Table 80.

Table 80 - Allowed message sizes per protocol

Protocol ID	Min Tx	Max Tx	Min Rx	Max Rx	Notes
TP2_0_PS or TP2_0_CHx	4	4096	4	4096	4 bytes of CAN ID followed by 4092 bytes of data

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_ MSG_PROTOCOL_ID.

19.4.3 Format Checks for Messages Passed to the API

If DataSize violates Min Tx or Max Tx columns the vendor DLL shall return, ERR_INVALID_MSG.

19.4.4 Message Flag and Status Definitions

Supported Message Status (RxStatus) Definitions:

The TX_MSG_TYPE and CAN_29BIT_ID RxStatus bits will be supported. For the TP2.0 protocol, loopback messages returned by the API will have TX_MSG_TYPE set in RxStatus. Messages with a 29-bit ID field will have CAN_29BIT_ID set.

There are two additions to the Message Status (RxStatus) definitions to be used as indications.

Table 81 - RxStatus bit definitions

Definition	RxStatus Bit(s)	Description	Value
CONNECTION_LOST	17	Connection lost status indication will be sent when an established connection has been lost, or when an attempt to establish a connection fails. The receiving CAN address RX-ID-A or RX-ID-P will be in Data[0] through Data[3] of the message, Data[0] is MSB. Data[4] will indicated why the connection failed: 0 = teardown, 1= timeout, 0xD6 = not supported, 0xD7 = temporarily not supported, 0xD8 = no resources free, ExtraDataIndex = 0, and DataSize = 5. TP2.0 only. The implicit PASS filter is removed.	0 = no connection lost indication 1 = connection lost or establish attempt failed
CONNECTION_ESTABLISHED	16	Connection established status indication will be sent when a connection is successfully established. The receiving CAN address RX-ID-A or RX-ID-P will be in Data[0] through Data[3] of the message Data[0] is MSB. The sending CAN address TX-ID-A or TX-ID-P will be in Data[4] through Data[7], Data[4] is MSB. ExtraDataIndex = 0, DataSize = 8. TP2.0 only	0 = no connection indication 1 = connection was successful

Supported TxFlags Definitions:

The TxFlag CAN_29BIT_ID will be supported if the TP2.0 connection was established with CAN_ID_BOTH and a 29-bit CAN message needs to be sent.

The One Additional TxFlags Definition:

Table 82 - TxFlag bit definition

Definition	TxFlags Bit(s)	Description	Value
TP2_0_BROADCAST_MSG	16	Send this message as a broadcast, which means it will be sent five times with an interval of TP2_0_ T_BR_INT. The last two data bytes will alternate between 0x55 and 0xAA.	0 = normal message 1 = TP2.0 broadcast message

19.5 Return Values

A new Return Value has been defined and shown in Table 83.

Table 83 - Additional return values for the TP2.0 protocol

Definition	Description
ERR_NO_CONNECTION_ESTABLISHED	TP2.0 connection has been lost or was never established

19.6 Discovery Support

The device shall report support for the TP2.0 protocol and associated parameters through the discovery mechanism defined in Section 25.

20. FAULT-TOLERANT CAN

20.1 Scope of the Fault-Tolerant CAN Optional Feature

Information contained in this section will define extensions to a compliant SAE J2534-1 interface to support Fault-Tolerant CAN.

20.2 Pass-Thru System Requirements

20.2.1 Pin Usage

Fault-Tolerant CAN (FTCAN) may be connected to either of these sets of pins on the SAE J1962 connector:

- Pin 1 - CAN-high, pin 9 - CAN-low
- Pin 3 - CAN-high, pin 11 - CAN-low

As with all SAE J2534-2 optional protocols, no default pin is identified, therefore, the application developer will be required to set the pin to be used. See the SAE J1962 pin selection section for discussion of pin usage.

20.3 Win32 Application Programming Interface

20.3.1 API Functions - Overview

Information contained in this section is intended to define the API resources required to incorporate an optional protocol channel. This channel, identified as Fault-Tolerant CAN (FTCAN), will require hardware and software API support to fully implement this feature.

From the SAE J2534 API perspective, FT_CAN_PS and FT_CAN_CHx are equivalent to CAN_PS and CAN_CHx except as specifically mentioned in this section. Likewise, FT_ISO15765_PS and FT_ISO15765_CHx are equivalent to ISO15765_PS and ISO15765_CHx except as specifically mentioned in this section.

The details on the physical implementation of FTCAN are defined in ISO11898-3.

If this feature is not supported an error code, ERR_NOT_SUPPORTED, will be returned by the call PassThruConnect.

Table 84 summarizes the changes to the SAE J2534-1 API functions.

Table 84 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new ProtocolID values.

20.3.2 API Functions - Detailed Information

20.3.2.1 PassThruConnect

When PassThruConnect is called with either the FT_CAN_PS or FT_ISO15765_PS Protocol IDs, the physical layer remains disconnected until a call to PassThruIoctl, SET_CONFIG is made to select the desired cable and pins.

20.3.2.2 ProtocolID Values

Only the definition and description of the ProtocolID value is defined in Table 85. The actual value is defined Section 24.

Table 85 - ProtocolID descriptions

Definition	Description
FT_CAN_PS	Raw fault-tolerant CAN messages with pin selection
FT_ISO15765_PS	Fault-tolerant CAN adhering to ISO15765-2 flow control with pin selection
FT_CAN_CHx	Additional channels of raw fault-tolerant CAN messages
FT_ISO15765_CHx	Additional channels of fault-tolerant CAN adhering to ISO15765-2 flow control

20.4 Message Structure

20.4.1 Elements

The ProtocolID field of the pass-thru message structure shall always contain the ProtocolID that was passed into PassThruConnect function. All other ProtocolIDs will result in ERR_MSG_PROTOCOL_ID. (The only exception to this is the mixed format frames feature on a CAN network.)

SAE J2534-1 definitions of requirements, restrictions, and error conditions for CAN and ISO15765 protocols will apply to single wire CAN channels discussed above.

20.4.2 Message Flag and Status Definitions

There is one addition to the Message Status (RxStatus) definitions.

Table 86 - New RxStatus definitions

Definition	RxStatus Bit(s)	Description	Value
LINK_FAULT	17	Status bit in a received message that is set when the transceiver has detected a network fault, but the device was still able to correctly receive the message.	0 = no fault 1 = fault detected

20.5 Discovery Support

The device shall report support for Fault Tolerant CAN and associated parameters through the discovery mechanism defined in Section 25.

21. CAN FD PROTOCOL

21.1 Scope of the CAN FD Optional Feature

Information contained in this section will define extensions to a compliant SAE J2534-1 interface to support CAN FD.

21.2 Pass-Thru System Requirements

21.2.1 Pin Usage

As with all SAE J2534-2 optional protocols, no default pin is identified

See Section 6 for details of the method used to switch cable and pins.

CAN FD is typically connected to the following pins on the SAE J1962 connector:

Option #	CAN High	CAN Low
1	6	14
2	3	11
3	12	13
4	1	9

Note that when PassThruConnect is called, the physical layer remains disconnected until the data phase data rate has been set and a call to PassThruIoctl, SET_CONFIG, J1962_PINS is made.

The cable used to connect the pass-thru device to the vehicle's SAE J1962 connector must meet the wiring requirements for CAN FD.

21.2.2 Communication Protocol

The following features of CAN FD must be supported by the pass-thru device:

a. 125, 250, 500, and 1000 kbps arbitration data rates.

b. 125, 250, 500, 1000, 2000, 4000, and 5000 kbps data phase data rates.

c. 11- and 29-bit identifiers for both CAN 2.0 and CAN FD messages.

d. Transmitting and receiving both CAN 2.0 or CAN FD messages.

e. Support all the frame sizes as defined by CAN FD.

f. Shall support a switchable 120-Ω termination resistor between CAN high and CAN low. Refer to ISO11898-1:2015 for resistor specifications.

g. Filters shall ignore the CAN message format; the message will be filtered on the address and data regardless of whether the message was received as a CAN 2.0 or CAN FD message.

h. Periodic messages are limited to a maximum <DataSize> of 12 bytes, which includes the 4-byte CAN ID.

21.2.3 Simultaneous Communication on Multiple Protocols

The CAN FD implementation will have the same simultaneous communication on multiple protocol requirements as the CAN ProtocolID (either J1850VPW or J1850PWM, and either ISO9141 or ISO14230 must be able to run simultaneously).

21.3 Win32 Application Programming Interface

21.3.1 API Functions - Overview

CAN FD will require hardware and software API support to fully implement this feature. The details on the physical implementation of CAN FD are defined in ISO11898-1:2015.

A new <ProtocolID> has been defined for the CAN FD physical layer. This section details the API resources required to enable use of the SAE J2534 API as applied to CAN FD.

NOTE: There are no remote frames in CAN FD format. CAN FD controllers are able to handle remote frames in CAN 2.0 format.

If this feature is not supported an error code, ERR_NOT_SUPPORTED, will be returned by the call PassThruConnect. The calling application will be required to notify the user that this optional feature may not be supported by the interface.

Table 87 summarizes the changes to the SAE J2534-1 API functions.

Table 87 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new <ProtocolID> values.
PassThruReadMsgs	Added <RxStatus> bits to indicate how a CAN FD message was received.
PassThruWriteMsgs	Added <TxFlags> to control how a CAN FD message is transmitted.
PassThruIoctl	Added GET_CONFIG and SET_CONFIG parameters. Added new IOCTL IDs to support CAN FD capability.

21.3.2 API Functions - Detailed Information

21.3.2.1 PassThruConnect

PassThruConnect sets up the pass-thru resources required for CAN FD communication. After a successful call to PassThruConnect, the physical layer shall remain disconnected, and no communication shall be possible until the data phase baud rate has been set, and subsequent to that the communications pins have been selected. It is critical that the data phase baud rate is set before the pins have been configured in order to ensure that the network is not corrupted.

21.3.2.1.1 C/C++ Prototype

There are no changes defined for the function prototype.

21.3.2.1.2 Parameters

There are no changes to the parameters.

21.3.2.1.3 Connect Flag Values

There are two connect flags defined for CAN FD:

Table 88 - Connect flag values

Definition	Description
CAN_29BIT_ID	0 = receive standard CAN ID (11 bit) 1 = receive extended CAN ID (29 bit)
CAN_ID_BOTH	0 = either standard or extended CAN ID used (defined by CAN_29BIT_ID) 1 = both standard and extended CAN ID used - if the CAN controller allows prioritizing either standard (11 bit) or extended (29 bit) CAN ID, then CAN_29BIT_ID will determine the higher priority ID

21.3.2.1.4 <ProtocolID> Values

The definition, description, and value of the <ProtocolID> value is defined in Table 89.

Table 89 - <ProtocolID> descriptions

Definition	Description
FD_CAN_PS	CAN FD protocol with pin selection
FD_CAN_CHx	Additional channels of CAN FD

21.3.2.1.5 Return Values

There are no changes defined for the return values.

21.3.2.2 PassThruReadMsgs

21.3.2.2.1 RxStatus

Table 92 defines the <RxStatus> bits for CAN FD messages.

21.3.2.3 PassThruWriteMsgs

21.3.2.3.1 TxFlags

Table 93 defines the <TxFlags> bits for CAN FD messages.

21.3.2.4 PassThruIoctl

21.3.2.4.1 IOCTL ID Values

There are no new IOCTL ID values for CAN FD.

21.3.2.5 Configuration Parameters

21.3.2.5.1 SET_CONFIG and GET_CONFIG Supported Parameters

The DATA_RATE and LOOPBACK configuration parameters defined for CAN in SAE 2534-1 will be supported.

The BIT_SAMPLE_POINT and SYNC_JUMP_WIDTH configuration parameters are not changeable for CAN FD. The device must set these parameters to the values defined in SAE J2284. If SET_CONFIG is called with these parameters, the error ERR_NOT_SUPPORTED will be returned. GET_CONFIG can be used to read the arbitration phase BIT_SAMPLE_POINT and SYNC_JUMP_WIDTH set by the device.

In addition, Table 90 shows the two new configuration parameters to support CAN FD.

Table 90 - IOCTL GET_CONFIG/SET_CONFIG parameters details

Parameter	Valid Values for Parameter	Default Value (decimal)	Description
FD_CAN_DATA_PHASE_RATE	125000, 250000, 500000, 1000000, 2000000, 4000000, and 5000000	N/A	The data rate used for the data section of a CAN FD message. The data phase rate cannot be less than the arbitration data rate.
HS_CAN_TERMINATION	0, 3	0	0 = no termination 3 = 120-Ω termination

For the Protocol ID FD_CAN_PS, the IoctlID SET_CONFIG configuration parameter FD_CAN_DATA_PHASE_RATE must be set before the pins are selected, otherwise the return value shall be ERR_FAILED. If setting of the data phase rate and pin selection are done in a single call, the application shall place the data phase rate configuration ahead of the pin selection in the configuration array, otherwise the return value shall be ERR_FAILED.

For the Protocol IDs FD_CAN_CHx, the channel must remain at high-impedance until the IoctlID SET_CONFIG configuration parameter FD_CAN_DATA_PHASE_RATE is set, at which time the channel will be connected to the associated pins. ERR_PIN_INVALID shall be returned from PassThruReadMsgs, PassThruWriteMsgs, PassThruStartPeriodicMsg, or PassThruStartMsgFilter if the FD_CAN_DATA_PHASE_RATE has not been successfully set.

21.4 Message Structure

21.4.1 C/C++ Definition

There is no change to the C/C++ definition.

21.4.2 Elements

There is no change to any of the elements.

21.4.3 Message Data Formats

When using FD_CAN_PS or FD_CAN_CHx the first four bytes of <Data> contain the 11-bit CAN ID or the 29-bit CAN ID:

Table 91 - Allowed message sizes per protocol

Protocol	Min Tx	Max Tx	Min Rx	Max Rx	Notes
FD_CAN_PS or FD_CAN_CHx	4	68	4	68	4 bytes of CAN ID plus 0 to 64 bytes of CAN data. Note: The CAN data is limited to 0 to 8, 12, 16, 20, 24, 32, 48, or 64 bytes.

The <ProtocolID> field of the pass-thru message structure shall always contain the <ProtocolID> that was passed into PassThruConnect function. All other <ProtocolID>s will result in ERR_MSG_PROTOCOL_ID.

21.4.4 Format Checks for Messages Passed to the API

The <ProtocolID> field of the pass-thru message structure shall always contain the <ProtocolID> that was passed into PassThruConnect function.

If the <TxFlag> FD_CAN_FORMAT is set to 0 then ERR_INVALID_MSG shall be returned when the value of <DataSize> in the PASSTHRU_MSG structure is not between 4 bytes and 12 bytes.

If the <TxFlag> FD_CAN_FORMAT is set to one, then ERR_INVALID_MSG shall be returned when the value of <DataSize> in the PASSTHRU_MSG structure is not as defined in Table 91.

If CAN_ID_BOTH bit was not set during PassThruConnect then the <TxFlag> CAN_29_BIT flag of the message must match CAN_29_BIT flag passed to the PassThruConnect, otherwise ERR_INVALID_MSG is returned.

21.4.5 Conventions for Returning Messages from the API

There are no changes to this section.

21.4.6 Conventions for Returning Indications from the API

There are no changes to this section.

21.4.7 Message Flag and Status Definitions

21.4.7.1 RxStatus

Supported <RxStatus> bits:

Table 92 - <RxStatus> bit definitions

Definition	Description	Value
TX_MSG_TYPE	Receive indication/transmit loopback	0 = received, i.e., this message was transmitted on the bus by another node 1 = transmitted, i.e., this is the echo of the message transmitted by the PassThru device
CAN_29BIT_ID	Indicates the CAN ID size for the CAN FD and CAN message	0 = 11-bit identifier 1 = 29-bit identifier
FD_CAN_BRS	For CAN FD frames this flag reflects the value of the BRS bit in the received frame	0 = the CAN FD controller did not switch baud rate during the data phase (all bits at the arbitration speed) 1 = the CAN FD controller switched to the alternate baud rate during the data phase NOTE: The value shall be zero for CAN 2.0 frames
FD_CAN_FORMAT	Indicates the frame was received in the CAN 2.0 format	0 = the message was received using the CAN 2.0 format 1 = the message was received in the CAN FD format
FD_CAN_ESI	Indicates the ERROR STATE of the node that sent the frame	0 = error active node 1 = error passive node NOTE: This flag is not used for CAN 2.0 frames and shall be set to 0

21.4.7.2 TxFlags

Supported <TxFlags> bits:

Table 93 - <TxFlags> bit definitions

Definition	Description	Value
CAN_29BIT_ID	Indicates the CAN ID size for the CAN FD and CAN message	0 = 11 bit 1 = 29 bit
FD_CAN_BRS	For CAN FD frames, this flag reflects the value of the BRS bit when sending a frame	0 = the CAN FD controller will not switch baud rate during the data phase (all bits at the arbitration speed) 1 = the CAN FD controller will switch to the alternate baud rate during the data phase NOTE: The value shall be zero for CAN 2.0 frames
FD_CAN_FORMAT	Indicates the format in which the frame will be transmitted	0 = the message will be transmitted using the CAN 2.0 format 1 = the message will be transmitted in the CAN FD format

21.5 Discovery Support

The device shall report support for CAN FD Network and associated parameters through the discovery mechanism defined in Section 25.

22. ISO15765 ON CAN FD

22.1 Scope of the ISO15765 on CAN FD Optional Feature

Information contained in this section will define extensions to a compliant SAE J2534-1 interface to support ISO15765 on CAN FD.

22.2 Pass-Thru System Requirements

22.2.1 Pin Usage

Refer to pin usage section for CAN FD.

22.2.2 Communication Protocol

Refer to CAN FD section for CAN FD related communication protocol requirements.

In addition, the following shall be supported by the pass-thru device:

a. Shall be capable of sending and receiving ISO15765-2:2016 messages in both CAN 2.0 and CAN FD formats.

b. Shall be capable of sending ISO15765-2 messages with data lengths up to 4124 bytes, or 4123 bytes for extended addressed frames.

c. Shall be capable of receiving ISO15765-2 messages with data lengths up to 4124 bytes, or 4123 bytes for extended addressed frames.

d. Shall support the CAN_MIXED_FORMAT as described in Section 8. Additionally, filters shall ignore the CAN message format (messages will be filtered on the address and data regardless of whether the message was received as a CAN 2.0 or CAN FD message).

e. For ISO15765 in CAN 2.0 format, single frames and the last consecutive frames are padded when data size is less than eight and padding is enabled.

f. For ISO15765 in CAN FD format, single frames and the last consecutive frames are padded when data sizes greater than eight but less than the CAN FD frame size when padding is enabled as defined in ISO15765-2.

g. The pass-thru device will send all flow control frames using the same CAN message format as the first frame. Thus, a first frame received in CAN 2.0 format will generate a flow control frame using the CAN 2.0 format, a first frame received in CAN FD format will generate a flow control frame using the CAN FD format.

h. Periodic messages for ISO15765 on CAN FD are limited to a maximum <DataSize> of 11 bytes, which includes 4 bytes of CAN ID.

i. When the pass-thru device is transmitting a segmented ISO15765 on CAN FD message, the ISO15765 state machine shall accept flow control messages from the receiver in CAN FD format or CAN 2.0 format.

j. If the sender (pass-thru device is receiving) sends a first frame with a message size larger than 4124 bytes, 4123 bytes with extended addressing, the pass-thru device shall NACK the first frame with a flow control frame containing the flow status overflow value.

22.2.3 Simultaneous Communication on Multiple Protocols

See “Simultaneous Communication on Multiple Protocols” section for CAN FD.

22.3 Win32 Application Programming Interface

22.3.1 API Functions - Overview

ISO15765 on CAN FD will require hardware and software to support CAN FD. The details on the physical implementation of CAN FD are defined in ISO11898-1:2015.

A new <ProtocolID> has been defined for ISO15765 on a CAN FD network. This section details the API resources required to enable use of the SAE J2534 API as applied to ISO15765 on CAN FD.

If this feature is not supported an error code, ERR_NOT_SUPPORTED, will be returned by the call PassThruConnect. The calling application will be required to notify the user that this optional feature may not be supported by the interface.

Table 94 summarizes the changes to the SAE J2534-1 API functions.

Table 94 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new <ProtocolID> values.
PassThruReadMsgs	Added <RxStatus> bits to indicate how an ISO15765 on CAN FD message was received.
PassThruWriteMsgs	Added <TxFlags> to control how an ISO15765 on CAN FD message is transmitted.
PassThruIoctl	Added GET_CONFIG and SET_CONFIG parameters. Added new IOCTL IDs to support ISO15765 on CAN FD capability.

22.3.2 API Functions - Detailed Information

22.3.2.1 PassThruConnect

PassThruConnect sets up the pass-thru resources required for CAN FD communication. After a successful call to PassThruConnect, the physical layer shall remain disconnected, and no communication shall be possible until the data phase baud rate has been set, and subsequent to that the communications pins have been selected. It is critical that the data phase baud rate is set before the pins have been configured in order to ensure that the network is not corrupted.

22.3.2.2 Connect Flag Values

There are two connect flags defined for ISO15765 on CAN FD.

Table 95 - Connect flag values

Definition	Description	Bit
CAN_29BIT_ID	0 = receive standard CAN ID (11 bit) 1 = receive extended CAN ID (29 bit)	Use the SAE J2534-1 bit
CAN_ID_BOTH	0 = either standard or extended CAN ID types used - CAN ID type defined by bit 8 1 = both standard and extended CAN ID types used - if the CAN controller allows prioritizing either standard (11 bit) or extended (29 bit) CAN ID's then bit 8 will determine the higher priority ID type	Use the SAE J2534-1 bit

22.3.2.2.1 <ProtocolID> Values

The definition, description and value of the <ProtocolID> value is defined in Table 96.

Table 96 - <ProtocolID> descriptions

Definition	Description
FD_ISO15765_PS	CAN FD adhering to ISO15765-2 flow control
FD_ISO15765_CHx	Additional channels of ISO15765 on CAN FD

22.3.2.2.2 Return Values

There are no changes defined for the return values.

22.3.2.3 PassThruReadMsgs

22.3.2.3.1 RxStatus

<RxStatus> identified in Table 99 will apply to all ISO15765 on CAN FD messages.

22.3.2.4 PassThruWriteMsgs

22.3.2.4.1 TxFlags

<TxFlags> identified in Table 100 will apply to all ISO15765 on CAN FD messages.

22.3.2.5 PassThruIoctl

22.3.2.5.1 IOCTL ID Values

There are no new IOCTL ID values for ISO15765 on CAN FD.

22.3.2.6 Configuration Parameters

22.3.2.6.1 SET_CONFIG and GET_CONFIG supported parameters

All configuration parameters defined for ISO15765 in SAE J2534-1 (except the parameter excluded below) shall be supported.

The BIT_SAMPLE_POINT and SYNC_JUMP_WIDTH configuration parameters are not changeable for CAN FD. The device must set these parameters to the values defined in SAE J2284. If SET_CONFIG is called with these parameters, the error ERR_NOT_SUPPORTED will be returned. GET_CONFIG can be used to read the arbitration phase BIT_SAMPLE_POINT and SYNC_JUMP_WIDTH set by the device.

In addition, Table 97 shows the five new configuration parameters to support CAN FD.

Table 97 - IOCTL GET_CONFIG/SET_CONFIG parameters details

Parameter	Valid Values for Parameter	Default Value (decimal)	Description
FD_CAN_DATA_PHASE_RATE	250000, 500000, 1000000, 2000000, 4000000, and 5000000	N/A	The data rate used for the data section of a CAN FD message. The data phase rate cannot be less than the arbitration data rate.
FD_ISO15765_TX_DATA_LENGTH	8, 12, 16, 20, 24, 32, 48, 64	64	The data phase data size used for ISO15765 single frames, first frames, and consecutive frames. Note: The last frame shall always use the smallest data length possible.
HS_CAN_TERMINATION	0, 3	0	0 = no termination 3 = 120-Ωtermination
N_CR_MAX	$0001 -$FFFF (1 ms per bit)	1000	Consecutive frame timeout.
ISO15765_PAD_VALUE	$00 - $FF	0	Pad byte used to pad frames.

For the Protocol ID FD_ISO15765_PS, the IoctlID SET_CONFIG configuration parameter FD_CAN_DATA_PHASE_RATE must be set before the pins are selected, otherwise the return value shall be ERR_FAILED. If setting of the data phase rate and pin selection are done in a single call, the application shall place the data phase rate configuration ahead of the pin selection in the configuration array, otherwise the return value shall be ERR_FAILED.

For the Protocol IDs FD_ISO15765_CHx, the channel must remain at high-impedance until the IoctlID SET_CONFIG configuration parameter FD_CAN_DATA_PHASE_RATE is set, at which time the channel will be connected to the associated pins. ERR_PIN_INVALID shall be returned from PassThruReadMsgs, PassThruWriteMsgs, PassThruStartPeriodicMsg, or PassThruStartMsgFilter if the FD_CAN_DATA_PHASE_RATE has not been successfully set.

22.4 Message Structure

22.4.1 C/C++ Definition

There is no change to the C/C++ definition.

22.4.2 Elements

There is no change to any of the elements.

22.4.3 Message Data Formats

When using FD_ISO15765_PS or FD_ISO15765_Chx the first 4 bytes of data contain the 11-bit or 29-bit CAN ID:

Table 98 - Allowed message sizes per protocol

Protocol	Min Tx	Max Tx	Min Rx	Max Rx	Notes
FD_ISO15765_PS or FD_ISO15765_CHx	4	4128	4	4128	4 bytes of CAN ID, 0 to 4124 bytes of ISO15765 data.
FD_ISO15765_PS or FD_ISO15765_CHx using Extended addressing	5	4128	5	4128	4 bytes of CAN ID, 1 byte extended address, 0 to 4123 bytes of ISO15765 data.

22.4.4 Format Checks for Messages Passed to the API

The <ProtocolID> field of the pass-thru message structure shall always contain the <ProtocolID> that was passed into PassThruConnect function or a <ProtocolID> allowed by mixed format.

If the <DataSize> in the PASSTHRU_MSG structure is not as defined in Table 98, then ERR_INVALID_MSG shall be returned.

If CAN_ID_BOTH bit was not set during PassThruConnect, then the <TxFlag> CAN_29_BIT flag of the message must match CAN_29_BIT flag passed to the PassThruConnect, otherwise ERR_INVALID_MSG is returned.

22.4.5 Conventions for Returning Messages from the API

There are no changes to this section.

22.4.6 Conventions for Returning Indications from the API

There are no changes to this section.

22.4.7 Message Flag and Status Definitions

22.4.7.1 RxStatus

Supported <RxStatus> bits:

Table 99 - <RxStatus> bit definitions

Definition	Description	Value
TX_MSG_TYPE	Receive indication/transmit loopback	0 = received, i.e., this message was transmitted on the bus by another node 1 = transmitted, i.e., this is the echo of the message transmitted by the PassThru device
START_OF_MESSAGE	Indicates the reception of the first frame of an ISO15765 multi-frame message - CAN ID and extended address, if present, shall be included in the message structure	0 = not a start of message indication 1 = first frame received
TX_INDICATION	ISO15765 TxDone indication - CAN ID and extended address, if present, shall be included in the message structure	0 = no TxDone 1 = TxDone
ISO15765_ PADDING_ERROR	For <ProtocolID> ISO15765 a CAN frame was received with less than 8-data bytes	0 = no error 1 = padding error
ISO15765_ADDR_TYPE	ISO15765-2 addressing method	0 = no extended address 1 = extended address is first byte after the CAN ID
CAN_29BIT_ID	Indicates the CAN ID type for the ISO15765 on CAN FD message	0 = 11-bit identifier 1 = 29-bit identifier
FD_CAN_BRS	For ISO15765 on CAN FD frames, this flag reflects the value of the BRS bit in the received frame	0 = the CAN FD controller did not switch baud rate during the data phase (all bits at the arbitration speed) 1 = the CAN FD controller switched to the alternate baud rate during the data phase NOTE: The value shall be zero for CAN 2.0 frames
FD_CAN_FORMAT	Indicates the format in which the frame was received	0 = the message was received using the CAN 2.0 format 1 = the message was received in the CAN FD format
FD_CAN_ESI	Indicates the ERROR STATE of the node that sent the frame	0 = error active node 1 = error passive node NOTE: This flag is not used for CAN 2.0 frames and shall be set to zero

22.4.7.2 TxFlags

Supported <TxFlags> bits:

Table 100 - <TxFlags> bit definitions

Definition	Description	Value
ISO15765_FRAME_PAD	ISO15765-2 frame padding	0 = no padding 1 = pad all flow-controlled messages to a full frame size using the PAD value
ISO15765_ADDR_TYPE	ISO15765-2 addressing method	0 = no extended address 1 = extended address is first byte after the CAN ID
CAN_29BIT_ID	CAN ID type for CAN and ISO15765	0 = 11 bit 1 = 29 bit
FD_CAN_BRS	For CAN FD frames, this flag reflects the value of the BRS bit when sending a frame	0 = the CAN FD controller will not switch baud rate during the data phase (all bits at the arbitration speed) 1 = the CAN FD controller will switch to the alternate baud rate during the data phase NOTE: The value shall be zero for CAN 2.0 frames
FD_CAN_FORMAT	Indicates the format in which the frame will be transmitted	0 = the message will be transmitted using the CAN 2.0 format 1 = the message will be transmitted in the CAN FD format

22.5 Discovery Support

The device shall report support for ISO15765 on CAN FD Network and associated parameters through the discovery mechanism defined Section 25.

23. READING THE VOLTAGE ON AN SAE J1962 PIN

23.1 Scope of the Pin Voltage Read Optional Feature

This section details the extensions to SAE J2534-1 that allow reading the voltage on a specified J1962 pin. A new IOCTL ID is defined for this purpose.

23.2 Win32 Application Programming Interface

The PassThruIoctl function is used with the following IOCTL ID: READ_J1962PIN_VOLTAGE.

The <IoctlID> value of READ_J1962PIN_VOLTAGE is used to obtain the voltage on the specified SAE J1962 pin measured at the time of the function call. Table 101 describes the parameters for this function call. When the function is successfully completed, voltage read on the designated pin shall be placed in the variable pointed to by <OutputPtr>. The measurable voltage range shall be minimum of 0 to 24 VDC with an accuracy of at least ±500 mV and a resolution of at least 200 mV. A larger range is allowed; typical regulated voltage range on 12 VDC systems is 0 to 16 V, while regulated voltage on 24 VDC systems is 0 to 32 V. When a voltage greater than the maximum range is sampled, the device should report its maximum range value. The units shall be in millivolts. The ground reference shall be signal ground, defined as pin 5 on the SAE J1962 connector. Attempting to reading voltages on pins 4 or 5 shall not be supported and shall result in the return value ERR_PIN_INVALID.

Device shall return STATUS_NOERROR for pins it supports and ERR_PIN_INVALID for pins it does not support. A voltage reading on pin 16 should report the same voltage as a call to READ_VBATT.

NOTE: The pass-thru device is not required to provide high sample rate voltage measurements. The pass-thru device must update the voltage measurement for the specified pin at a minimum 10 Hz rate. If an application is calling the READ_J1962PIN_VOLTAGE PassThruIoctl at a rate faster than the specified minimum rate, the pass-thru device may return the same reading for multiple function calls.

Differences Between API Function Calls that Read Voltage:

The voltage on pin 16 can be read using READ_VBATT or, if supported, READ_J1962PIN_VOLTAGE pin 16. An application should use the READ_VBATT function call to read the voltage on pin 16 since this method is the standard SAE J2534-1 way to read the voltage and must be supported on all devices. READ_J1962PIN_VOLTAGE shall be used when there is an application feature need to read voltage from any one of the supported connector pins to accomplish a diagnostic process supported by the diagnostic application.

Table 101 - READ_J1962PIN_VOLTAGE details

Parameter	Description
ChannelID	Device ID assigned by DLL during PassThruOpen.
IoctlID	Is set to READ_J1962PIN_VOLTAGE.
InputPtr	Is a pointer to an unsigned long containing the pin number (1 to 16).
OutputPtr	Is a pointer to an unsigned long where the pin voltage is returned.

The following is a code example to read the voltage on a pin:

```
unsigned long ulPin = 1;
unsigned long ulPinVoltage;
long RetVal;

RetVal = PassThruIoctl(DeviceID, READ_J1962PIN_VOLTAGE, &ulPin, &ulPinVoltage);
if(RetVal == STATUS_NOERROR)
{
        if(ulPinVoltage > 6000) printf("Key is ON\n");
        else printf("Key is OFF");
}
```

23.3 Discovery Support

The device shall report support for reading the SAE J1962 pin voltage through the discovery mechanism defined in Section 25.

24. ETHERNET_NDIS

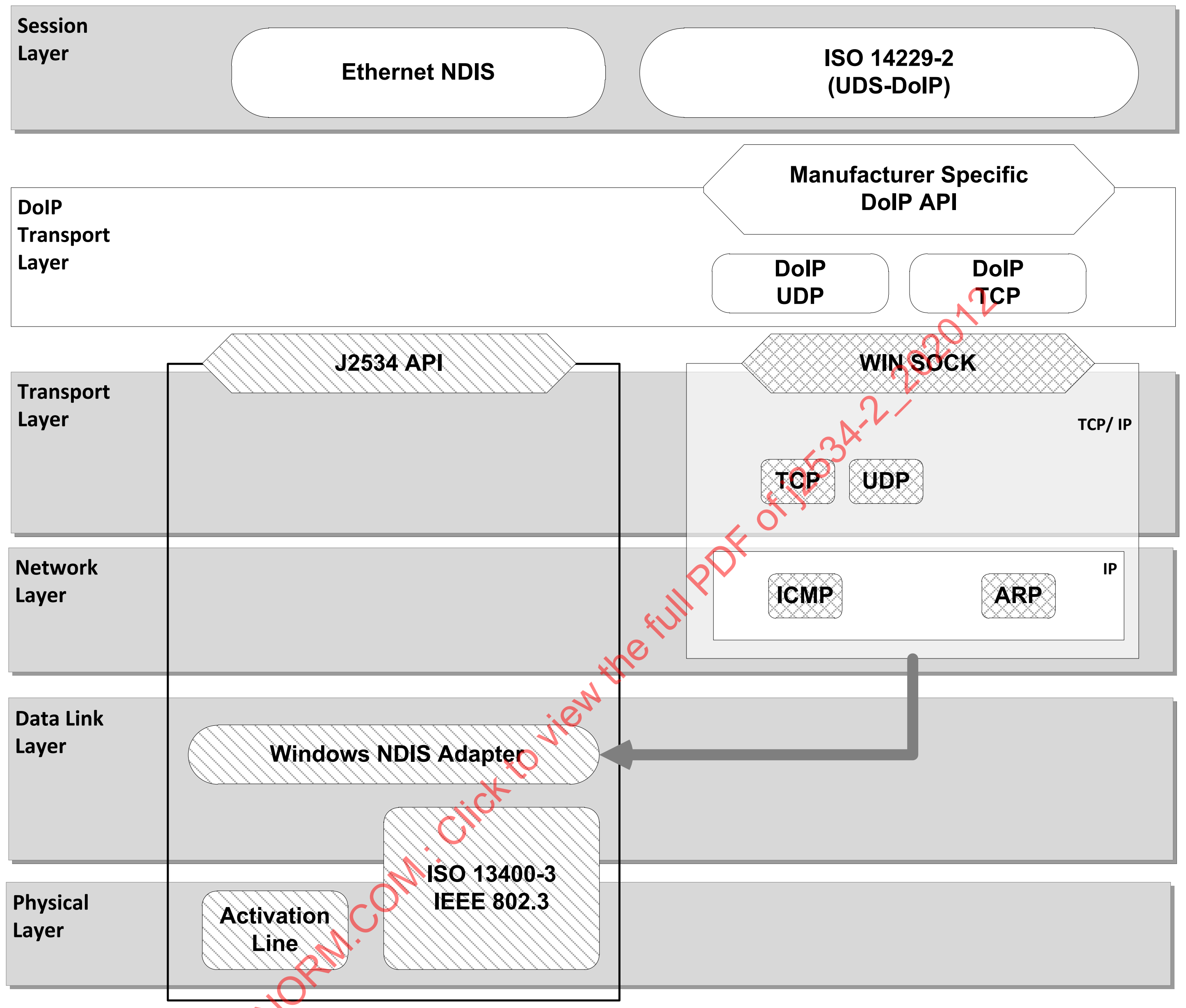
24.1 Scope of the Ethernet Optional Feature

This implementation of Ethernet is different from all other protocols because all message traffic shall be routed through an application interface for Ethernet as defined by the Microsoft's Network Driver Interface Specification (NDIS). The actual implementation may use the PC's internal network adapters or a remote NDIS miniport driver (RNDIS).

The SAE J2534 API's primary responsibility shall be to assert the activation signal, and configure routing for physical Ethernet connection to the DLC. The NDIS driver may be pre-loaded on the PC or loaded when PassThruConnect() is called. The installed NDIS driver shall be compatible with current and future versions of the Windows operating system.

Figure 14 shows how the ETHERNET_NDIS implementation is divided between the pass-thru device and the Windows Ethernet system.

Figure 14 - ETHERNET_NDIS implementation



24.2 Pass-Thru System Requirements

Application developers should consult ISO13400-2 and Windows Socket Services examples when implementing communications with the vehicle that will take place after the Pass-Thru NDIS Connection is established.

24.2.1 Connection to Vehicle

The Pass-Thru Interface shall support the following features to meet the Ethernet communication protocol requirements:

a. Pass-thru Device attachment cable shall meet the wiring requirements for Ethernet pin specified for the vehicle DLC (reference ISO13400-4 and Table 104 of this document)

b. Meet the test equipment activation line requirements specified in ISO13400-3:2016.

c. Support the activation line process as defined in ISO13400-3:2016.

d. Support data rates of 10M bps or 100M bps as defined in IEEE 802.3.

e. Support the 100Base-TX and 10Base-T standards as defined in IEEE 802.3.

f. Support the auto-MDI(X) feature as defined in IEEE 802.3.

24.2.2 Protocol Details

24.2.2.1 Link Activation

The full requirements for the activation line shall be met. The activation line needs to be pulled high to a threshold greater than 5V and kept there for the Ethernet connection to be activated. The activation line shall be asserted whenever an SAE J2534 ETHERNET_NDIS protocol connection is created. Refer to ISO13400-3:2016 for guidance on Ethernet activation.

24.2.3 Simultaneous Communication on Multiple Protocols

The ETHERNET_NDIS implementation will operate simultaneous with CAN or CAN-FD. Table 102 indicates which combinations of protocols shall be supported simultaneously.

Table 102 - Simultaneous communication options

Data Link Set 1	Data Link Set 2
CAN CAN FD ISO15765 ISO15765 FD	Ethernet NDIS

24.2.4 Pin Usage

ETHERNET_NDIS is connected to the following pins on the SAE J1962 connector:

Table 103 - ETHERNET_NDIS pin usage details

SAE J1962 PIN	Option 1 Configuration	Option 2 Configuration
1		Ethernet Tx (+)
3	Ethernet Tx (+)	
8	Activation Line	Activation Line
9		Ethernet Tx (-)
11	Ethernet Tx (-)	
12	Ethernet Rx (+)	Ethernet Rx (+)
13	Ethernet Rx (-)	Ethernet Rx (-)

24.2.5 API Functions - Overview

Information contained in this Section defines the API resources required to support an optional protocol channel. This channel, identified as ETHERNET_NDIS, will require hardware and software API support to fully implement this feature.

If this feature is not supported, an error code—ERR_NOT_SUPPORTED—will be returned by the call PassThruConnect.

Table 104 summarizes the changes to the SAE J2534-1 API functions.

Table 104 - SAE J2534 API functions

Function	Description of Change
PassThruConnect	Added new <ProtocolID> values.
PassThruReadMsgs	Always returns an error because this feature is not supported for ETHERNET_NDIS.
PassThruWriteMsgs	Always returns an error because this feature is not supported for ETHERNET_NDIS.
PassThruStartPeriodicMsg	Always returns an error because this feature is not supported for ETHERNET_NDIS.
PassThruStartMsgFilter	Always returns an error because this feature is not supported for ETHERNET_NDIS.
PassThruIoctl	Added new IOCTL ID.

24.2.5.1 PassThruConnect

Calling PassThruConnect with the <ProtocolID> set to ETHERNET_NDIS shall command the pass-thru interface to do one of the following:

a. If a specific J1962 pin configuration is selected by the application, the Pass-Thru Interface shall connect to the designated pin configuration and then enable the activation pull-up that wakes the vehicle's Ethernet network.

b. Automatically determine Option 1 or Option 2 pin configuration by performing the activation process (per ISO13400), followed by enabling the activation pull-up that wakes the vehicle's Ethernet network.

PassThruConnect shall return ERR_NO_CONNECTION_ESTABLISHED if the activation line process fails.

24.2.5.1.1 <Flags> Values

Table 105 outlines how the various <Flags> values will impact the Ethernet pins used.

Table 105 - <Flags> descriptions

Flags		Description
NDIS_PINS_OPTION2 State	**NDIS_PINS_OPTION1 State**	
0	0	Automatically determine if the pin configuration is Option 1 or Option 2 by reading activation line per ISO13400
0	1	Force Option 1 pin configuration
1	0	Force Option 2 pin configuration
1	1	Automatically determine if the pin configuration is Option 1 or Option 2 by reading activation line per ISO13400

24.2.5.1.2 <BaudRate> Values

The baud rate shall be auto negotiated by the NDIS device. The value of <BaudRate> is not evaluated by the API.

24.2.5.1.3 <ProtocolID> Values

The definition, description, and value of the <ProtocolID> value is defined in Table 106.

Table 106 - <ProtocolID> descriptions

Definition	Description
ETHERNET_NDIS	Raw Ethernet messages.

24.2.5.2 PassThruReadMsgs

The ETHERNET_NDIS protocol does not support receiving messages with the pass-thru device. Thus, a call to PassThruReadMsgs will return the error ERR_NOT_SUPPORTED.

24.2.5.3 PassThruWriteMsgs

The ETHERNET_NDIS protocol does not support transmitting messages with the pass-thru device. Thus, a call to PassThruWriteMsgs will return the error ERR_NOT_SUPPORTED.

24.2.5.4 PassThruStartPeriodicMsg

The ETHERNET_NDIS protocol does not support periodic messages. Thus, a call to PassThruStartPeriodicMsg will return the error ERR_NOT_SUPPORTED.

24.2.5.5 PassThruStartMsgFilter

The ETHERNET_NDIS protocol does not support message filters. Thus, a call to PassThruStartMsgFilter will return the error ERR_NOT_SUPPORTED.

24.2.5.6 PassThruIoctl

24.2.5.6.1 IOCTL ID Values

24.2.5.6.1.1 GET_NDIS_ADAPTER_INFO

The IOCTL ID, GET_NDIS_ADAPTER_INFO shall return information pertaining to the PC's internal network adapter as defined by NDIS_ADAPTER_INFORMATION structure. This IOCTL ID assists an application in binding to a specific network interface adapter. When an application binds to a network interface adapter, it can filter which inbound and outbound messages to the configured adapter IP address will be forwarded to the application.

Details of the GET_NDIS_ADAPTER_INFO are shown in Table 107. This information is read-only and cannot be set by an application.

Table 107 - GET_NDIS_ADAPTER_INFO details

Parameter	Description
ChannelID	Channel ID assigned by DLL during PassThruConnect.
IoctlID	Is set to the define GET_NDIS_ADAPTER_INFO.
InputPtr	Is a NULL pointer, as this parameter is not used.
OutputPtr	Points to the structure, NDIS_ADAPTER_INFORMATION, which is defined as follows: typedef struct { char AdapterUniqueID[128]; char AdapterName[64]; unsigned long Status; unsigned char MAC_Address[6] unsigned char IPV6_Address[16]; unsigned char IPV4_Address[4]; unsigned long EthernetPinConfig; } NDIS_ADAPTER_INFORMATION; where: AdapterUniqueID is the Operating System unique registry key for the adapter, which is an ASCII string that is NULL terminated. (For example, “{269623C9-C702-499E-90A4-E371092FC76A}”.) AdapterName is the Adapter Display Name, which is an ASCII string that is NULL terminated. (For example, “Acme Pass-Thru NDIS Adapter”.) Status is the adapter status; 0=disabled, 1=enabled. MAC_Address is the Media Access Control (MAC) Address of the NDIS adapter that is connected to the vehicle. IPV6_Address is the IPv6 address assigned to the NDIS adapter. There shall be no compressed or short form notation. The bytes shall be in network order big endian format. If the address is not available, all bytes shall be set to zero. IPV4_Address is the IPv4 address assigned to the NDIS adapter. The bytes shall be in network order big endian format. If the address is not available, all bytes shall be set to zero. EthernetPinConfig is the current Ethernet pin configuration for the NDIS adapter; 1=Option 1, 2=Option 2 (see Table 104 for more details).

24.3 Discovery Support

The device shall report support for ETHERNET_NDIS protocol and associated parameters through the discovery mechanism defined in Section 25.

24.4 DLL Installation and Registration

Upon device installation, a key will be set in the appropriate registry folder to indicate the support for the Ethernet protocol. In the case of devices which contain dynamic hardware architecture, an application will be able to determine the lack of runtime support when an error is returned from a PassThruConnect().

The registry entries are retained for support of legacy applications. New applications shall use the discovery mechanism defined in Section 25.

25. DISCOVERY MECHANISM

25.1 Scope of the Discovery Mechanism Feature

This section details the extensions to SAE J2534-1 to support a mechanism to programmatically determine/confirm the capabilities of a specific SAE J2534 device. This section details only the changes from SAE J2534-1. Items not specifically detailed in this section are assumed not to have changed.

25.2 Pass-Thru System Requirements

The Discovery Mechanism imposes no additional hardware requirements.

25.3 Win32 Application Programming Interface

25.3.1 API Functions - Overview

The purpose of this feature is to provide a mechanism to programmatically determine/confirm the capabilities of a specific SAE J2534 device. These capabilities are static and shall not change based on the current state of the device. (For example, the identification of which pins ISO15765 can be switched to shall not be altered if some of those pins are currently in use by another protocol.)

To access the capabilities, an application must successfully call PassThruOpen. Then, the application must call PassThruIoctl with a list of parameters it is interested in. Upon return, the device will indicate if the parameter is supported and its associate value. Two new IoctlIDs have been defined to support this feature, they are:

GET_DEVICE_INFO - Used to acquire the general capabilities of the device.

GET_PROTOCOL_INFO - Used to acquire the protocol specific capabilities of the device.

To obtain the general capabilities of the device, the application shall call PassThruIoctl with the ChannelID set to the DeviceID returned from PassThruOpen, IoctlID set to GET_DEVICE_INFO, and OutputPtr pointing to a list of parameter/value pairs of interest to the user on the specified device.

To obtain the protocol specific capabilities of the device, the application shall call PassThruIoctl with the ChannelID set to the DeviceID returned from PassThruOpen, IoctlID set to GET_PROTOCOL_INFO, the InputPtr pointing to a Protocol ID, and OutputPtr pointing to a list of parameter/value pairs of interest to the user for the associated protocol on the specified device. It is expected that the application would repeat this step for each protocol it is interested in.

Table 108 summarizes the changes to the SAE J2534-1 API functions.

Table 108 - SAE J2534 API functions

Function	Description of Change
PassThruIoctl	Add new IoctlID values

25.3.2 API Functions - Detailed Information

25.3.2.1 IOCTL Section

There are two additional IoctlIDs, as follows:

Table 109 - IOCTL details

Value of IoctlID	InputPtr Represents	OututPtr Represents	Purpose
GET_DEVICE_INFO	NULL pointer	Pointer to SPARAM_LIST	To acquire the general capabilities of the device.
GET_PROTOCOL_INFO	Pointer to Protocol ID	Pointer to SPARAM_LIST	To acquire the protocol specific capabilities of the device.

25.3.2.2 GET_DEVICE_INFO

The IoctlID value of GET_DEVICE_INFO shall be used to request the general capabilities of an SAE J2534 device. The calling application is responsible for allocating and initializing the inputs described in Table 110. Upon successful completion (a return code of STATUS_NOERROR), the corresponding supported and/or value elements shall have been updated. Requests for parameters that are not supported by the device shall cause the corresponding supported element to be set to NOT_SUPPORTED and the corresponding value to remain unaltered. In this case, the interface shall continue processing the rest of the parameters in the list and the return code shall be STATUS_NOERROR (as long as no other errors conditions exist). Valid device parameters and their associated descriptions are detailed in Table 111.

Table 110 - GET_DEVICE_INFO details

Parameter	Description
ChannelID	Device ID assigned by DLL during PassThruOpen.
IoctlID	Is set to the define GET_DEVICE_INFO.
InputPtr	Is set to NULL, as this parameter is not used.
OutputPtr	Points to the structure SPARAM_LIST, which is defined as follows: typedef struct { unsigned long NumOfParams;/* number of SPARAM elements */ SPARAM *ParamPtr;/* array of SPARAM */ } SPARAM_LIST where: NumOfParms is an INPUT, set by the application, which contains the number of SPARAM elements in the array pointed to by ParamPtr. ParamPtr is an INPUT, set by the application, which points to an array of SPARAM structures allocated by the application. The structure SPARAM is defined as follows: typedef struct { unsigned long Parameter;/* name of parameter */ unsigned long Value;/* value of the parameter */ unsigned long Supported;/* support for parameter */ } SPARAM where: Parameter is an INPUT, set by the application, which represents the ID of the parameter requested (see Table 111 for a list of valid parameters). Value is typically an OUTPUT, set by the interface, which represents the parameter's value. However, for certain parameters Value may also be an INPUT set by the application (see Table 111 for more details). Supported is an OUTPUT, set by the interface to 0 (indicating the associated parameter is NOT_SUPPORTED) or 1 (indicating the associated parameter is SUPPORTED). The contents of Value shall remain unchanged if the parameter is NOT_SUPPORTED.

Table 111 - GET_DEVICE_INFO parameter details

Parameter	Description	Associated Value
SERIAL_NUMBER	Represents the device serial number.	Value is an OUTPUT that, upon return, contains an unsigned long that is the device serial number.
PART_NUMBER	Represents the part number of the device.	Value is an OUTPUT that, upon return, contains an unsigned long that is the device part number.
J1850PWM_SUPPORTED	Represents support for SAE J1850PWM related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1850PWM_CHx Protocol IDs that are available. RR is the number of channels for the J1850PWM_PS Protocol ID that are available. SS is 1 if the J1850PWM Protocol ID is supported; otherwise it must be 0.
J1850VPW_SUPPORTED	Represents support for SAE J1850VPW related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1850VPW_CHx Protocol IDs that are available. RR is the number of channels for the J1850VPW_PS Protocol ID that are available. SS is 1 if the J1850VPW Protocol ID is supported; otherwise it must be 0.
ISO9141_SUPPORTED	Represents support for ISO9141 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO9141_CHx Protocol IDs that are available. RR is the number of channels for the ISO9141_PS Protocol ID that are available. SS is 1 if the ISO9141 Protocol ID is supported; otherwise it must be 0.
ISO14230_SUPPORTED	Represents support for ISO14230 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO14230_CHx Protocol IDs that are available. RR is the number of channels for the ISO14230_PS Protocol ID that are available. SS is 1 if the ISO14230 Protocol ID is supported; otherwise it must be 0.

Parameter	Description	Associated Value
CAN_SUPPORTED	Represents support for CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of CAN_CHx Protocol IDs that are available. RR is the number of channels for the CAN_PS Protocol ID that are available. SS is 1 if the CAN Protocol ID is supported; otherwise it must be 0.
ISO15765_SUPPORTED	Represents support for ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO15765_CHx Protocol IDs that are available. RR is the number of channels for the ISO15765_PS Protocol ID that are available. SS is 1 if ISO15765 is supported; otherwise it must be 0.
FT_CAN_SUPPORTED	Represents support for FT_CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FT_CAN_CHx Protocol IDs that are available. RR is the number of channels for the FT_CAN_PS Protocol ID that are available. SS must be 0.
FT_ISO15765_SUPPORTED	Represents support for FT_ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FT_ISO15765_CHx Protocol IDs that are available. RR is the number of channels for the FT_ISO15765_PS Protocol ID that are available. SS must be 0.
SCI_A_ENGINE_SUPPORTED	Represents support for the SCI_A_ENGINE Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_A_ENGINE Protocol ID is supported; otherwise it must be 0.

Parameter	Description	Associated Value
SCI_A_TRANS_SUPPORTED	Represents support for the SCI_A_TRANS Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_A_TRANS Protocol ID is supported; otherwise it must be 0.
SCI_B_ENGINE_SUPPORTED	Represents support for the SCI_B_ENGINE Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_B_ENGINE Protocol ID is supported; otherwise it must be 0.
SCI_B_TRANS_SUPPORTED	Represents support for the SCI_B_TRANS Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_B_TRANS Protocol ID is supported; otherwise it must be 0.
SW_ISO15765_SUPPORTED	Represents support for SW_ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of SW_ISO15765_CHx Protocol IDs that are available. RR is the number of channels for the SW_ISO15765_PS Protocol ID that are available. SS must be 0.
SW_CAN_SUPPORTED	Represents support for SW_CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of SW_CAN_CHx Protocol IDs that are available. RR is the number of channels for the SW_CAN_PS Protocol ID that are available. SS must be 0.

Parameter	Description	Associated Value
GM_UART_SUPPORTED	Represents support for GM_UART related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of GM_UART_CHx Protocol IDs that are available. RR is the number of channels for the GM_UART_PS Protocol ID that are available. SS must be 0.
UART_ECHO_BYTE_SUPPORTE D	Represents support for UART_ECHO_BYTE related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of UART_ECHO_BYTE_CHx Protocol IDs that are available. RR is the number of channels for the UART_ECHO_BYTE_PS Protocol ID that are available. SS must be 0.
HONDA_DIAGH_SUPPORTED	Represents support for HONDA_DIAGH related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of HONDA_DIAGH_CHx Protocol IDs that are available. RR is the number of channels for the HONDA_DIAGH_PS Protocol ID that are available. SS must be 0.
J1939_SUPPORTED	Represents support for SAE J1939 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1939_CHx Protocol IDs that are available. RR is the number of channels for the J1939_PS Protocol ID that are available. SS must be 0.
J1708_SUPPORTED	Represents support for SAE J1708 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1708_CHx Protocol IDs that are available. RR is the number of channels for the J1708M_PS Protocol ID that are available. SS must be 0.

Parameter	Description	Associated Value
TP2_0_SUPPORTED	Represents support for TP2_0 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of TP2_0_CHx Protocol IDs that are available. RR is the number of channels for the TP2_0_PS Protocol ID that are available. SS must be 0.
J2610_SUPPORTED	Represents support for SAE J2610 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J2610_CHx Protocol IDs that are available. RR is the number of channels for the J2610_PS Protocol ID that are available. SS must be 0.
ANALOG_IN_SUPPORTED	Represents support for ANALOG_IN_xxx related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ANALOG_IN_x Protocol IDs that are available. RR must be 0. SS must be 0.
FD_CAN_SUPPORTED	Represents support for CAN FD related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FD_CAN_CHx Protocol IDs that are available. RR is the number of channels for the FD_CAN_PS Protocol ID that are available. SS must be 0.
FD_ISO15765_SUPPORTED	Represents support for ISO15765 on CAN FD related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FD_ISO15765_CHx Protocol IDs that are available. RR is the number of channels for the FD_ISO15765_PS Protocol ID that are available. SS must be 0.

Parameter	Description	Associated Value
J1850PWM_SIMULTANEOUS	Represents support for SAE J1850PWM related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1850PWM_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the J1850PWM_PS Protocol ID that can operate simultaneously. SS is 1 if the J1850PWM Protocol ID is supported; otherwise it must be 0.
J1850VPW_SIMULTANEOUS	Represents support for SAE J1850VPW related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1850VPW_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the J1850VPW_PS Protocol ID that can operate simultaneously. SS is 1 if the J1850VPW Protocol ID is supported; otherwise it must be 0.
ISO9141_SIMULTANEOUS	Represents support for ISO9141 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO9141_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the ISO9141_PS Protocol ID that can operate simultaneously. SS is 1 if the ISO9141 Protocol ID is supported; otherwise it must be 0.
ISO14230_SIMULTANEOUS	Represents support for ISO14230 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO14230_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the ISO14230_PS Protocol ID that can operate simultaneously. SS is 1 if the ISO14230 Protocol ID is supported; otherwise it must be 0.

Parameter	Description	Associated Value
CAN_SIMULTANEOUS	Represents support for CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of CAN_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the CAN_PS Protocol ID that can operate simultaneously. SS is 1 if the CAN Protocol ID is supported; otherwise it must be 0.
ISO15765_SIMULTANEOUS	Represents support for ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ISO15765_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the ISO15765_PS Protocol ID that can operate simultaneously. SS is 1 if ISO15765 is supported; otherwise it must be 0.
FT_CAN_SIMULTANEOUS	Represents support for FT_CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FT_CAN_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the FT_CAN_PS Protocol ID that can operate simultaneously. SS must be 0.
FT_ISO15765_SIMULTANEOUS	Represents support for FT_ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FT_ISO15765_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the FT_ISO15765_PS Protocol ID that can operate simultaneously. SS must be 0.
SCI_A_ENGINE_SIMULTANEOUS	Represents support for the SCI_A_ENGINE Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_A_ENGINE Protocol ID is supported; otherwise it must be 0.

Parameter	Description	Associated Value
SCI_A_TRANS_SIMULTANEOUS	Represents support for the SCI_A_TRANS Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_A_TRANS Protocol ID is supported; otherwise it must be 0.
SCI_B_ENGINE_SIMULTANEOUS	Represents support for the SCI_B_ENGINE Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_B_ENGINE Protocol ID is supported; otherwise it must be 0.
SCI_B_TRANS_SIMULTANEOUS	Represents support for the SCI_B_TRANS Protocol ID.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ must be 0. RR must be 0. SS is 1 if the SCI_B_TRANS Protocol ID is supported; otherwise it must be 0.
SW_ISO15765_SIMULTANEOUS	Represents support for SW_ISO15765 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of SW_ISO15765_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the SW_ISO15765_PS Protocol ID that can operate simultaneously. SS must be 0.
SW_CAN_SIMULTANEOUS	Represents support for SW_CAN related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of SW_CAN_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the SW_CAN_PS Protocol ID that can operate simultaneously. SS must be 0.

Parameter	Description	Associated Value
GM_UART_SIMULTANEOUS	Represents support for GM_UART related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of GM_UART_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the GM_UART_PS Protocol ID that can operate simultaneously. SS must be 0.
UART_ECHO_BYTE_SIMULTANEOUS	Represents support for UART_ECHO_BYTE related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of UART_ECHO_BYTE_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the UART_ECHO_BYTE_PS Protocol ID that can operate simultaneously. SS must be 0.
HONDA_DIAGH_SIMULTANEOUS	Represents support for HONDA_DIAGH related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of HONDA_DIAGH_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the HONDA_DIAGH_PS Protocol ID that can operate simultaneously. SS must be 0.
J1939_SIMULTANEOUS	Represents support for SAE J1939 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1939_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the J1939_PS Protocol ID that can operate simultaneously. SS must be 0.
J1708_SIMULTANEOUS	Represents support for SAE J1708 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J1708_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the J1708M_PS Protocol ID that can operate simultaneously. SS must be 0.

Parameter	Description	Associated Value
TP2_0_SIMULTANEOUS	Represents support for TP2_0 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of TP2_0_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the TP2_0_PS Protocol ID that can operate simultaneously. SS must be 0.
J2610_SIMULTANEOUS	Represents support for SAE J2610 related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of J2610_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the J2610_PS Protocol ID that can operate simultaneously. SS must be 0.
ANALOG_IN_SIMULTANEOUS	Represents support for ANALOG_IN_xxx related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of ANALOG_IN_x Protocol IDs that can operate simultaneously. RR must be 0. SS must be 0.
FD_CAN_SIMULTANEOUS	Represents support for FD_CAN_xxx related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FD_CAN_CHx Protocol IDs that can operate simultaneously. RR is the number of channels for the FD_CAN_PS Protocol ID that can operate simultaneously. SS must be 0.
FD_ISO15765_SIMULTANEOUS	Represents support for FD_ISO765_xxx related Protocol IDs.	Value is an OUTPUT that, upon return, contains an unsigned long (with the format 0xPPQQRRSS) where each byte represents the following: PP must be 0. QQ is the number of FD_ISO15765_CHx Protocol IDs that can operate simultaneously. RR is the number of FD_ISO15765_PS Protocol IDs that can operate simultaneously. SS must be 0.

Parameter	Description	Associated Value
MAX_NON_VOLATILE_STORAGE	Represents the total number of Non-Volatile Memory Storage Locations supported.	Represents the total number of Non-Volatile Memory Storage Locations supported.
SHORT_TO_GND_J1962	Represents a query from the application to see if the pin identified in Value is valid for short to ground line on the SAE J1962 connector. Upon return, Supported indicates if the desired pin is SUPPORTED or NOT_SUPPORTED; the contents of Value shall remain un-altered. This query shall not change the state of the associated pin on the device. NOTE: The application may include this parameter one or more of times in the list.	Value is an INPUT that contains a bit mapped unsigned long (with the format 0xHHHHLLLL) where the bit that corresponds to the desired pin location is set to 1. All other bits shall be set to 0. HHHH must be 0. LLLL is the short to ground line. Bit 0 corresponds to pin 1, bit 1 corresponds to pin 2, etc. For this field, it is invalid to set more than 1 bit at a time.
PGM_VOLTAGE_J1962	Represents a query from the application to see if the pin identified in Value is valid for programming voltage on the SAE J1962 connector. Upon return, Supported indicates if the desired pin is SUPPORTED or NOT_SUPPORTED; the contents of Value shall remain un-altered. This query shall not change the state of the associated pin on the device. NOTE: The application may include this parameter one or more of times in the list.	Value is an INPUT that contains a bit mapped unsigned long (with the format 0xHHHHLLLL) where the bit that corresponds to the desired pin location is set to 1. All other bits shall be set to 0. HHHH is the programming voltage line. Bit 16 corresponds to pin 1, bit 17 corresponds to pin 2, etc. For this field, it is invalid to set more than 1 bit at a time. LLLL must be 0.

Parameter	Description	Associated Value
J1850PWM_PS_J1962	Represents a query from the application to see if the pins identified in Value are valid for the J1850PWM_PS Protocol ID on the SAE J1962 connector. Upon return, Supported indicates if the desired pins are SUPPORTED or NOT_SUPPORTED; the contents of Value shall remain un-altered. This query shall not change the state of the associated pins on the device. NOTE: The application may include this parameter one or more of times in the list.	Value is an INPUT that contains a bit mapped unsigned long (with the format 0xHHHHLLLL) where the bit that corresponds to the desired pin location is set to 1. All other bits shall be set to 0. HHHH is BUS + line. Bit 16 corresponds to pin 1, bit 17 corresponds to pin 2, etc. For this field, it is invalid to set more than 1 bit at a time. LLLL is BUS - line. Bit 0 corresponds to pin 1, bit 1 corresponds to pin 2, etc. For this field, it is invalid to set more than 1 bit at a time.
J1850VPW_PS_J1962	Represents a query from the application to see if the pin identified in Value is valid for the J1850VPW_PS Protocol ID on the SAE J1962 connector. Upon return, Supported indicates if the desired pin is SUPPORTED or NOT_SUPPORTED; the contents of Value shall remain un-altered. This query shall not change the state of the associated pin on the device. NOTE: The application may include this parameter one or more of times in the list.	Value is an INPUT that contains a bit mapped unsigned long (with the format 0xHHHHLLLL) where the bit that corresponds to the desired pin location is set to 1. All other bits shall be set to 0. HHHH is BUS + line. Bit 16 corresponds to pin 1, bit 17 corresponds to pin 2, etc. For this field, it is invalid to set more than 1 bit at a time. LLLL must be 0.