



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.....	10
1.1	Purpose.....	10
1.2	Documentation Convention.....	10
2.	REFERENCES.....	10
2.1	Applicable Documents.....	10
2.1.1	SAE Publications.....	10
2.1.2	IHS Markit.....	11
2.1.3	ISO Documents.....	11
2.1.4	IEEE Publications.....	11
3.	DEFINITIONS.....	12
4.	ACRONYMS.....	13
4.1	Acronyms for Single Wire CAN.....	13
4.2	Acronyms for GM UART.....	13
4.3	Acronyms for Analog Inputs.....	13
4.4	Acronyms for UART Echo Byte Protocol.....	13
4.5	Acronyms for SAE J1939.....	13
4.6	Acronyms for TP 2.0.....	13
4.7	Acronyms for Fault-Tolerant CAN.....	13
4.8	Acronyms for CAN FD and ISO15765 on CAN FD.....	13
4.9	Acronyms for ETHERNET_NDIS.....	14
5.	ENABLING SAE J2534-2 FEATURES AND ACCESSING MULTIPLE SAE J2534 DEVICES.....	14

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

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

Copyright © 2020 SAE International

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

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

SAE WEB ADDRESS:

Full document details: SAE J2534-2_202012

For more information on this standard, visit
https://www.sae.org/standards/content/J2534-2_202012

Documents
Document Sharing Hub

6.	PIN SELECTION.....	14
6.1	Scope of the Pin Selection Optional Feature.....	14
6.2	Pass-Thru System Requirements.....	14
6.2.1	Pin Usage.....	14
6.3	Win32 Application Programming Interface.....	14
6.3.1	API Functions - Overview.....	14
6.3.2	API Functions - Detailed Information.....	15
6.3.3	IOCTL.....	16
6.4	Discovery Support.....	19
7.	ACCESS TO ADDITIONAL CHANNELS.....	19
7.1	Discovery Support.....	19
8.	MIXED FORMAT FRAMES ON A CAN NETWORK.....	20
8.1	Scope of the Mixed Format Frames on a CAN Network Optional Feature.....	20
8.2	Win32 Application Programming Interface.....	20
8.2.1	API Functions - Overview.....	20
8.2.2	API Functions - Detailed Information.....	23
8.2.3	IOCTL Section.....	24
8.2.4	Message Structure.....	25
8.2.5	Elements.....	25
8.3	Discovery Support.....	25
9.	SINGLE WIRE CAN.....	25
9.1	Scope of the Single Wire CAN Optional Feature.....	25
9.2	Pass-Thru System Requirements.....	25
9.2.1	Pin Usage.....	25
9.3	Win32 Application Programming Interface.....	25
9.3.1	API Functions - Overview.....	25
9.3.2	API Functions - Detailed Information.....	27
9.3.3	IOCTL Section.....	27
9.4	Message Structure.....	30
9.4.1	Elements.....	30
9.5	Discovery Support.....	31
10.	ANALOG INPUTS.....	31
10.1	Scope of the Analog Inputs Optional Feature.....	31
10.2	Pass-Thru System Requirements.....	31
10.2.1	Analog Inputs.....	31
10.2.2	Simultaneous Communication on Multiple Protocols.....	31
10.3	Win32 Application Programming Interface.....	31
10.3.1	API Functions - Overview.....	31
10.3.2	API Functions - Detailed Information.....	31
10.3.3	IOCTL Section.....	32
10.4	Message Structure.....	36
10.4.1	Examples.....	36
10.4.2	Message Flag and Status Definitions.....	37
10.4.3	DLL Installation and Registry.....	37
10.5	Discovery Support.....	37
11.	GM UART (SAE J2740).....	38
11.1	Scope of the GM UART Optional Feature.....	38
11.2	Pass-Thru System Requirements.....	38
11.2.1	Pin Usage.....	38
11.3	Win32 Application Programming Interface.....	38
11.3.1	API Functions - Overview.....	38
11.3.2	API Functions - Detailed Information.....	39
11.3.3	IOCTL Section.....	39

11.4	Message Structure	43
11.4.1	Elements	43
11.4.2	Message Data Formats	43
11.5	DLL Installation and Registration	43
11.6	Discovery Support	43
12.	UART ECHO BYTE PROTOCOL	43
12.1	Scope of the UART Echo Byte Protocol Optional Feature	43
12.2	Pass-Thru System Requirements	43
12.2.1	Simultaneous Communication on Multiple Protocols	43
12.2.2	Pin Usage	43
12.3	Win32 Application Programming Interface	44
12.3.1	API Functions - Overview	44
12.3.2	API Change List	45
12.3.3	API Functions - Detailed Information	45
12.3.4	IOCTL Section	46
12.4	Message Structure	48
12.4.1	C/C++ Definition	48
12.4.2	Message Data Formats	48
12.4.3	Format Checks for Messages Passed to the API	48
12.4.4	Message Flag and Status Definitions	48
12.5	Return Value Error Codes	48
12.6	Discovery Support	49
13.	HONDA DIAG-H PROTOCOL	49
13.1	Scope of the Honda DIAG-H Optional Feature	49
13.2	Pass-Thru System Requirements	49
13.2.1	Simultaneous Communication on Multiple Protocols	49
13.2.2	Programmable Power Supply	49
13.2.3	Pin Usage	49
13.2.4	Honda Diagnostic Connectors	49
13.2.5	Serial Communication Interface/Protocol	51
13.2.6	Electrical Characteristics of Interface	52
13.3	Win32 Application Programming Interface	54
13.3.1	API Functions - Overview	54
13.3.2	API Functions - Detailed Information	55
13.3.3	IOCTL Section	55
13.4	Message Structure	56
13.4.1	C/C++ Definition	56
13.4.2	Elements	56
13.4.3	Message Data Formats	56
13.4.4	Format Checks for Messages Passed to the API	56
13.4.5	Message Flag and Status Definitions	56
13.5	Discovery Support	56
14.	REPEAT MESSAGING	56
14.1	Scope of the Repeat Messaging Protocol Optional Feature	56
14.2	Win32 Application Programming Interface	56
14.2.1	API Functions - Overview	56
14.2.2	IOCTL Section	57
14.3	Discovery Support	61
15.	EXTENDED PROGRAMMING VOLTAGE SUPPORT	61
15.1	Scope of the Extended Programming Voltage Feature	61
15.2	Pass-Thru System Requirements	61
15.3	Win32 Application Programming Interface	61
15.3.1	API Functions - Overview	61
15.3.2	API Functions - Detailed Information	61
15.4	Discovery Support	61

16.	SAE J1939 PROTOCOL	62
16.1	Scope of the SAE J1939 Protocol Optional Feature.....	62
16.2	Pass-Thru System Requirements	62
16.2.1	Connection to Vehicle	62
16.2.2	Communication Protocol	62
16.2.3	Simultaneous Communication on Multiple Protocols.....	62
16.3	Win32 Application Programming Interface.....	62
16.3.1	API Functions - Overview.....	62
16.3.2	API Functions - Detailed Information	63
16.3.3	IOCTL Section.....	64
16.4	Message Structure	65
16.4.1	C/C++ Definition.....	65
16.4.2	Elements	65
16.4.3	Message Data Formats.....	65
16.4.4	Format Checks for Messages Passed to the API	66
16.4.5	Conventions for Returning Messages from the API.....	66
16.4.6	Message Flag and Status Definitions	66
16.5	Return Value Error Codes.....	67
16.6	Discovery Support.....	67
17.	SAE J1708 PROTOCOL	67
17.1	Scope of the SAE J1708 Protocol Optional Feature.....	67
17.2	Pass-Thru System Requirements	67
17.2.1	Connection to Vehicle	67
17.2.2	Communication Protocol	68
17.2.3	Simultaneous Communication on Multiple Protocols.....	68
17.3	Win32 Application Programming Interface.....	68
17.3.1	API Functions - Overview.....	68
17.3.2	API Functions - Detailed Information	68
17.4	Message Structure	69
17.4.1	C/C++ Definition.....	69
17.4.2	Elements	69
17.4.3	Message Data Formats.....	69
17.4.4	Conventions for Returning Messages from the API.....	70
17.4.5	Message Flag and Status Definitions	70
17.5	Discovery Support.....	70
18.	EXTENDED PASSTHRUIOCTL FOR DEVICE CONFIGURATION PARAMETERS.....	70
18.1	Scope of the Extended PassThruIoctl Optional Feature.....	70
18.2	Pass-Thru Concept	70
18.3	Win32 Application Programming Interface.....	70
18.3.1	API Functions - Overview.....	70
18.4	IOCTL Section.....	71
18.4.1	GET_DEVICE_CONFIG	71
18.4.2	SET_DEVICE_CONFIG.....	72
18.5	Discovery Support.....	74
19.	TP2.0 PROTOCOL	74
19.1	Scope of the TP2.0 Protocol Optional Feature	74
19.2	Pass-Thru System Requirements	75
19.2.1	Simultaneous Communication on Multiple Protocols.....	75
19.2.2	Pin Usage.....	75
19.3	Win32 Application Programming Interface.....	75
19.3.1	API Functions - Overview.....	75
19.3.2	API Functions - Detailed Information	76
19.3.3	IOCTL Section.....	77

19.4	Message Structure	80
19.4.1	C/C++ Definition	80
19.4.2	Message Data Formats	80
19.4.3	Format Checks for Messages Passed to the API	81
19.4.4	Message Flag and Status Definitions	81
19.5	Return Values	82
19.6	Discovery Support	82
20.	FAULT-TOLERANT CAN	82
20.1	Scope of the Fault-Tolerant CAN Optional Feature	82
20.2	Pass-Thru System Requirements	82
20.2.1	Pin Usage	82
20.3	Win32 Application Programming Interface	82
20.3.1	API Functions - Overview	82
20.3.2	API Functions - Detailed Information	83
20.4	Message Structure	83
20.4.1	Elements	83
20.4.2	Message Flag and Status Definitions	83
20.5	Discovery Support	84
21.	CAN FD PROTOCOL	84
21.1	Scope of the CAN FD Optional Feature	84
21.2	Pass-Thru System Requirements	84
21.2.1	Pin Usage	84
21.2.2	Communication Protocol	84
21.2.3	Simultaneous Communication on Multiple Protocols	85
21.3	Win32 Application Programming Interface	85
21.3.1	API Functions - Overview	85
21.3.2	API Functions - Detailed Information	85
21.4	Message Structure	87
21.4.1	C/C++ Definition	87
21.4.2	Elements	87
21.4.3	Message Data Formats	87
21.4.4	Format Checks for Messages Passed to the API	87
21.4.5	Conventions for Returning Messages from the API	88
21.4.6	Conventions for Returning Indications from the API	88
21.4.7	Message Flag and Status Definitions	88
21.5	Discovery Support	89
22.	ISO15765 ON CAN FD	89
22.1	Scope of the ISO15765 on CAN FD Optional Feature	89
22.2	Pass-Thru System Requirements	89
22.2.1	Pin Usage	89
22.2.2	Communication Protocol	89
22.2.3	Simultaneous Communication on Multiple Protocols	90
22.3	Win32 Application Programming Interface	90
22.3.1	API Functions - Overview	90
22.3.2	API Functions - Detailed Information	91
22.4	Message Structure	93
22.4.1	C/C++ Definition	93
22.4.2	Elements	93
22.4.3	Message Data Formats	93
22.4.4	Format Checks for Messages Passed to the API	93
22.4.5	Conventions for Returning Messages from the API	93
22.4.6	Conventions for Returning Indications from the API	93
22.4.7	Message Flag and Status Definitions	94
22.5	Discovery Support	95

23.	READING THE VOLTAGE ON AN SAE J1962 PIN.....	95
23.1	Scope of the Pin Voltage Read Optional Feature.....	95
23.2	Win32 Application Programming Interface.....	95
23.3	Discovery Support.....	96
24.	ETHERNET_NDIS.....	96
24.1	Scope of the Ethernet Optional Feature.....	96
24.2	Pass-Thru System Requirements.....	97
24.2.1	Connection to Vehicle.....	97
24.2.2	Protocol Details.....	98
24.2.3	Simultaneous Communication on Multiple Protocols.....	98
24.2.4	Pin Usage.....	98
24.2.5	API Functions - Overview.....	98
24.3	Discovery Support.....	101
24.4	TDLL Installation and Registration.....	101
25.	DISCOVERY MECHANISM.....	102
25.1	Scope of the Discovery Mechanism Feature.....	102
25.2	Pass-Thru System Requirements.....	102
25.3	Win32 Application Programming Interface.....	102
25.3.1	API Functions - Overview.....	102
25.3.2	API Functions - Detailed Information.....	103
26.	SAE J2534-2 RESOURCE.....	136
26.1	Connect Flag Values.....	136
26.2	ProtocolID Values.....	138
26.3	Filter Type Values.....	140
26.4	IOCTL ID Values.....	140
26.5	IOCTL GET/SET CONFIG Parameter Details.....	141
26.6	IOCTL GET/SET DEVICE_CONFIG Parameter Details.....	142
26.7	GET_DEVICE_INFO Defines.....	142
26.8	GET_PROTOCOL_INFO Defines.....	144
26.9	Error Return Values.....	144
26.10	TxFlags Bits.....	145
26.11	RxStatus Bits.....	145
27.	NOTES.....	146
27.1	Revision Indicator.....	146
Figure 1	Processing of received messages when CAN_MIXED_FORMAT is CAN_MIXED_FORMAT_ON.....	21
Figure 2	Processing of received messages when CAN_MIXED_FORMAT is CAN_MIXED_FORMAT_ALL_FRAMES.....	22
Figure 3	12-pin vehicle connector.....	38
Figure 4	12-pin tool connector.....	38
Figure 5	Process flow for obtaining bus mastership.....	42
Figure 6	Diagnostic communication line configuration (3-pin DLC model).....	50
Figure 7	Diagnostic communication line configuration (5-pin DLC model).....	50
Figure 8	Diagnostic communication line configuration (16-pin DLC model without CAN).....	51
Figure 9	Diagnostic communication line configuration (16-pin DLC model with CAN).....	51
Figure 10	Recommended interface circuit on tester.....	52
Figure 11	Load circuit.....	53
Figure 12	Definition of t_{DST}	53
Figure 13	Threshold voltage.....	54
Figure 14	ETHERNET_NDIS implementation.....	97

Table 1	Protocol ID values (SAE J2534-1 defined)	15
Table 2	SAE J2534 API functions	15
Table 3	IOCTL GET_CONFIG/SET_CONFIG parameter details	17
Table 4	SAE J2534 API functions	22
Table 5	IOCTL GET_CONFIG/SET_CONFIG parameter details	24
Table 6	SAE J2534 API functions	27
Table 7	ProtocolID descriptions	27
Table 8	IOCTL details	28
Table 9	IOCTL GET_CONFIG/SET_CONFIG parameter details	29
Table 10	SW_CAN_HS details	29
Table 11	SW_CAN_NS details	29
Table 12	RxStatus bit definitions	30
Table 13	TxFlags bit definitions	30
Table 14	SAE J2534 API functions	31
Table 15	ProtocolID values	32
Table 16	IOCTL GET_CONFIG/SET_CONFIG parameter details	33
Table 17	Values for the AVERAGING_METHOD parameter	34
Table 18	Sample A/D parameter configuration	36
Table 19	Data from ANALOG_IN_1 subsystem with READINGS_PER_MSG = 1	36
Table 20	Data from ANALOG_IN_2 subsystem with READINGS_PER_MSG = 1	37
Table 21	Data from ANALOG_IN_1 subsystem with READINGS_PER_MSG = 2	37
Table 22	Data from ANALOG_IN_2 subsystem with READINGS_PER_MSG = 2	37
Table 23	Data from ANALOG_IN_1 subsystem with ACTIVE_CHANNELS = 0xB	37
Table 24	RxStatus bit definitions	37
Table 25	SAE J2534 API functions	39
Table 26	Protocol ID values	39
Table 27	IOCTL details	40
Table 28	SET_POLL_RESPONSE details	40
Table 29	Poll response message	40
Table 30	Typical 3-byte type poll message	40
Table 31	Typical 4-byte type poll message	41
Table 32	BECOME_MASTER detail	41
Table 33	Allowed message sizes per protocol	43
Table 34	SAE J2534 API functions	45
Table 35	Protocol ID values	45
Table 36	SET_CONFIG and GET_CONFIG additional parameters	46
Table 37	Parameter translation table	47
Table 38	Allowed message sizes per protocol	48
Table 39	Simultaneous communication options	49
Table 40	Serial interface parameters	52
Table 41	Electrical characteristics	52
Table 42	Transmitter electrical requirements	53
Table 43	Transmitter timing requirements	53
Table 44	Receiver electrical requirements	54
Table 45	Receiver timing requirements	54
Table 46	SAE J2534 API functions	55
Table 47	Protocol ID values	55
Table 48	Allowed message sizes per protocol	56
Table 49	SAE J2534 API functions	57
Table 50	IOCTL details	57
Table 51	START_REPEAT_MESSAGE details	58
Table 52	QUERY_REPEAT_MESSAGE details	60
Table 53	Status parameter values	60
Table 54	STOP_REPEAT_MESSAGE details	60
Table 55	SAE J2534 API functions	61
Table 56	SAE J2534 API functions	63
Table 57	Protocol ID values	63
Table 58	Return values	63
Table 59	IOCTL details	64

Table 60	IOCTL GET_CONFIG/SET_CONFIG parameters details	64
Table 61	Protect_J1939_addr details	65
Table 62	Allowed message sizes per protocol.....	66
Table 63	RxStatus bit definitions.....	67
Table 64	SAE J2534 API functions	68
Table 65	Connect flag values.....	69
Table 66	Protocol ID values	69
Table 67	Allowed message sizes per protocol.....	69
Table 68	TxFlags bit definitions	70
Table 69	SAE J2534 API functions	71
Table 70	IOCTL details	71
Table 71	GET_DEVICE_CONFIG details	72
Table 72	SET_DEVICE_CONFIG details	73
Table 73	IOCTL GET_DEVICE_CONFIG/SET_DEVICE_CONFIG parameter details	74
Table 74	SAE J2534 API functions	76
Table 75	Protocol ID values	77
Table 76	IOCTL details	77
Table 77	TP2.0 configuration parameters.....	78
Table 78	REQUEST_CONNECTION details	79
Table 79	TEARDOWN_CONNECTION details	80
Table 80	Allowed message sizes per protocol.....	80
Table 81	RxStatus bit definitions.....	81
Table 82	TxFlag bit definition.....	82
Table 83	Additional return values for the TP2.0 protocol.....	82
Table 84	SAE J2534 API functions	83
Table 85	ProtocolID descriptions	83
Table 86	New RxStatus definitions	83
Table 87	SAE J2534 API functions	85
Table 88	Connect flag values.....	86
Table 89	<ProtocolID> descriptions	86
Table 90	IOCTL GET_CONFIG/SET_CONFIG parameters details	87
Table 91	Allowed message sizes per protocol.....	87
Table 92	<RxStatus> bit definitions	88
Table 93	<TxFlags> bit definitions	89
Table 94	SAE J2534 API functions	90
Table 95	Connect flag values.....	91
Table 96	<ProtocolID> descriptions	91
Table 97	IOCTL GET_CONFIG/SET_CONFIG parameters details	92
Table 98	Allowed message sizes per protocol.....	93
Table 99	<RxStatus> bit definitions	94
Table 100	<TxFlags> bit definitions	95
Table 101	READ_J1962PIN_VOLTAGE details.....	96
Table 102	Simultaneous communication options	98
Table 103	ETHERNET_NDIS pin usage details	98
Table 104	SAE J2534 API functions	99
Table 105	<Flags> descriptions	99
Table 106	<ProtocolID> descriptions	100
Table 107	GET_NDIS_ADAPTER_INFO details	101
Table 108	SAE J2534 API functions	102
Table 109	IOCTL details	103
Table 110	GET_DEVICE_INFO details	104
Table 111	GET_DEVICE_INFO parameter details.....	105
Table 112	GET_PROTOCOL_INFO details	131
Table 113	Protocol groups	132
Table 114	GET_PROTOCOL_INFO parameter details	132
Table 115	Connect flag values.....	137
Table 116	ProtocolID values	138
Table 117	Filter type values	140
Table 118	IOCTL ID values.....	141
Table 119	IOCTL GET_CONFIG/SET_CONFIG ID values	141

Table 120	IOCTL GET_DEVICE_CONFIG/SET_DEVICE_CONFIG details.....	142
Table 121	GET_DEVICE_INFO defines values.....	142
Table 122	GET_PROTOCOL_INFO defines values.....	144
Table 123	Error return values	144
Table 124	New SAE J2534-1 return values	145
Table 125	TxFlags values	145
Table 126	RxStatus bits	145

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: Flexible 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. <table><tr><th>SAE J1939-13 PIN</th><th>Function</th><th>PIN Number</th></tr><tr><td>A</td><td>Ground</td><td>1</td></tr><tr><td>B</td><td>Battery+</td><td>2</td></tr><tr><td>C</td><td>CAN_H</td><td>3</td></tr><tr><td>D</td><td>CAN_L</td><td>4</td></tr><tr><td>E</td><td>Shield/NC</td><td>5</td></tr><tr><td>F</td><td>J1708+</td><td>6</td></tr><tr><td>G</td><td>J1708-</td><td>7</td></tr><tr><td>H</td><td>-</td><td>8</td></tr><tr><td>J</td><td>-</td><td>9</td></tr></table>	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
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 Deutsch connector, onto which the physical layer is to be switched. <table><tr><th>SAE J1708 PIN</th><th>Function</th><th>PIN Number</th></tr><tr><td>A</td><td>J1708+</td><td>1</td></tr><tr><td>B</td><td>J1708-</td><td>2</td></tr><tr><td>C</td><td>Battery+</td><td>3</td></tr><tr><td>D</td><td>-</td><td>4</td></tr><tr><td>E</td><td>Ground</td><td>5</td></tr><tr><td>F</td><td>-</td><td>6</td></tr></table> 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
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.

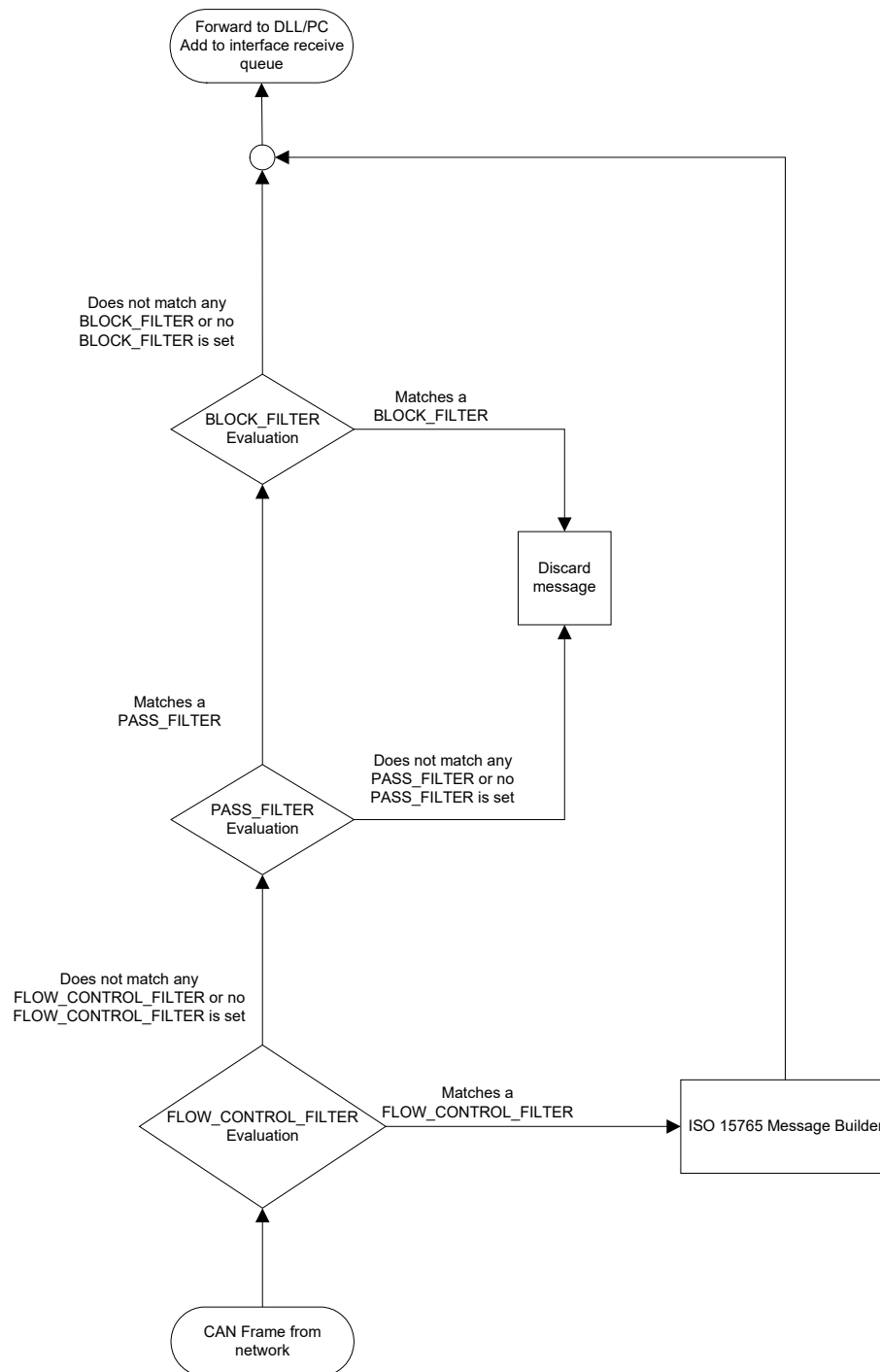


Figure 1 - Processing of received messages when `CAN_MIXED_FORMAT` is `CAN_MIXED_FORMAT_ON`

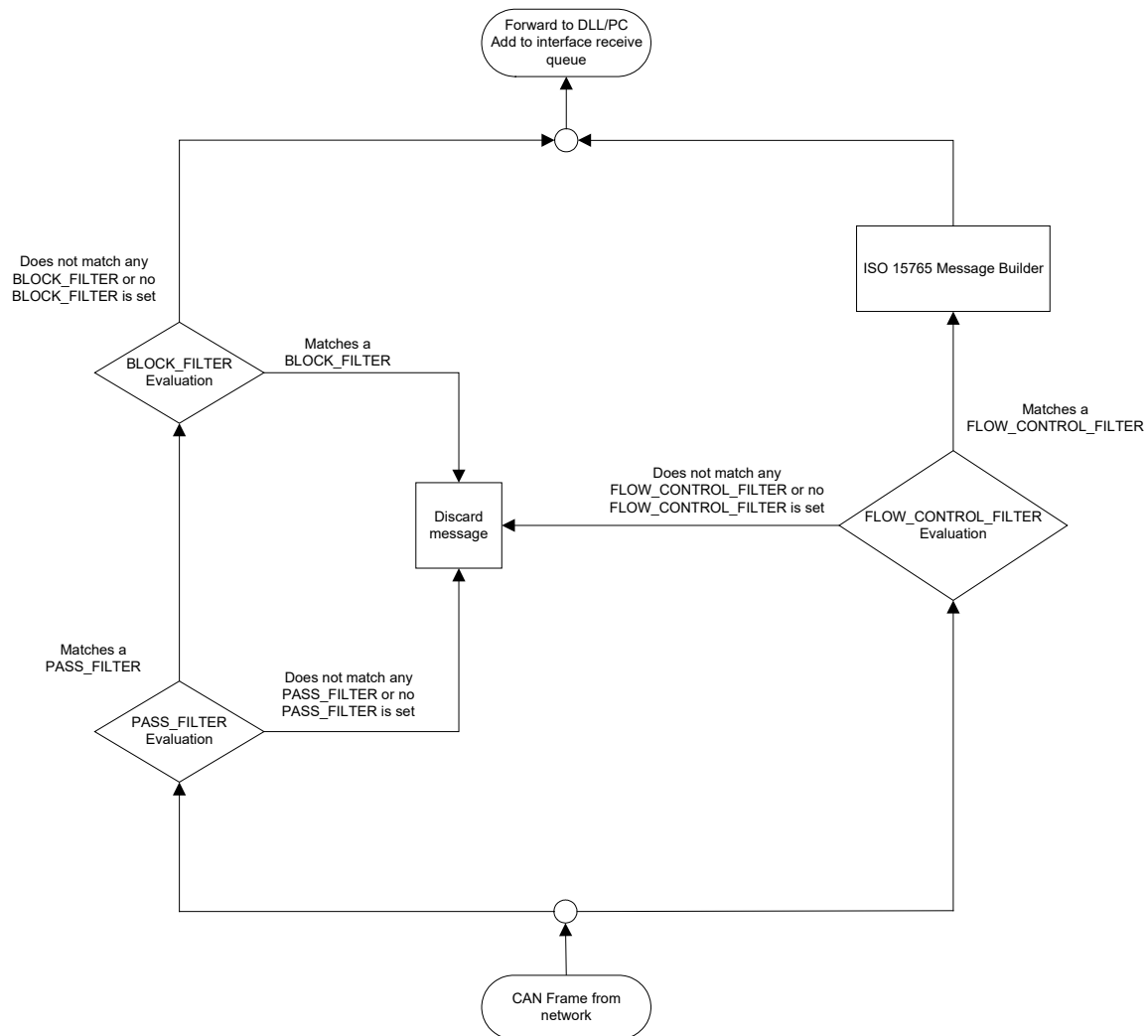


Figure 2 - Processing of received messages when CAN_MIXED_FORMAT is CAN_MIXED_FORMAT_ALL_FRAMES

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 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:

$$\text{Reading} = (\text{Sample}_1 + \text{Sample}_2 + \dots + \text{Sample}_{\text{SAMPLES_PER_READING}}) / \text{SAMPLES_PER_READING}$$

10.3.3.2.5.2 MAX_LIMIT_AVERAGE

The MAX_LIMIT_AVERAGE simply chooses the maximum value.

$$\text{Reading} = \text{Max} (\text{Sample}_1 + \text{Sample}_2 + \dots + \text{Sample}_{\text{SAMPLES_PER_READING}})$$

10.3.3.2.5.3 MIN_LIMIT_AVERAGE

The MIN_LIMIT_AVERAGE simply chooses the minimum value.

$$\text{Reading} = \text{Min} (\text{Sample}_1 + \text{Sample}_2 + \dots + \text{Sample}_{\text{SAMPLES_PER_READING}})$$

10.3.3.2.5.4 MEDIAN_AVERAGE

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

$$\text{Reading} = \text{Sample}_{(\text{SAMPLES_PER_READING}+1)/2}$$

Or if SAMPLES_PER_READING is even:

$$\text{Reading} = (\text{Sample}_{\text{SAMPLES_PER_READING}/2} + \text{Sample}_{(\text{SAMPLES_PER_READING}/2)+1}) / 2$$

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.

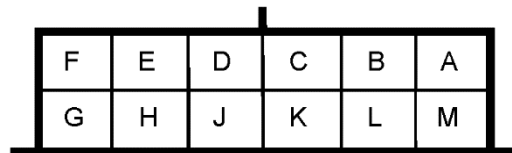


Figure 3 - 12-pin vehicle connector

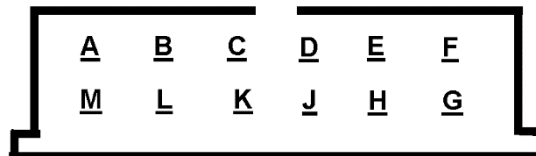


Figure 4 - 12-pin tool connector

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 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
<code>ChannelID</code>	Channel ID assigned by DLL during <code>PASSTHRUCONNECT</code>
<code>IoctlID</code>	Is set to <code>BECOME_MASTER</code>
<code>InputPtr</code>	Points to unsigned char <code>Poll_ID</code> (Value to be compared to Poll Message MIB)
<code>OutputPtr</code>	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.

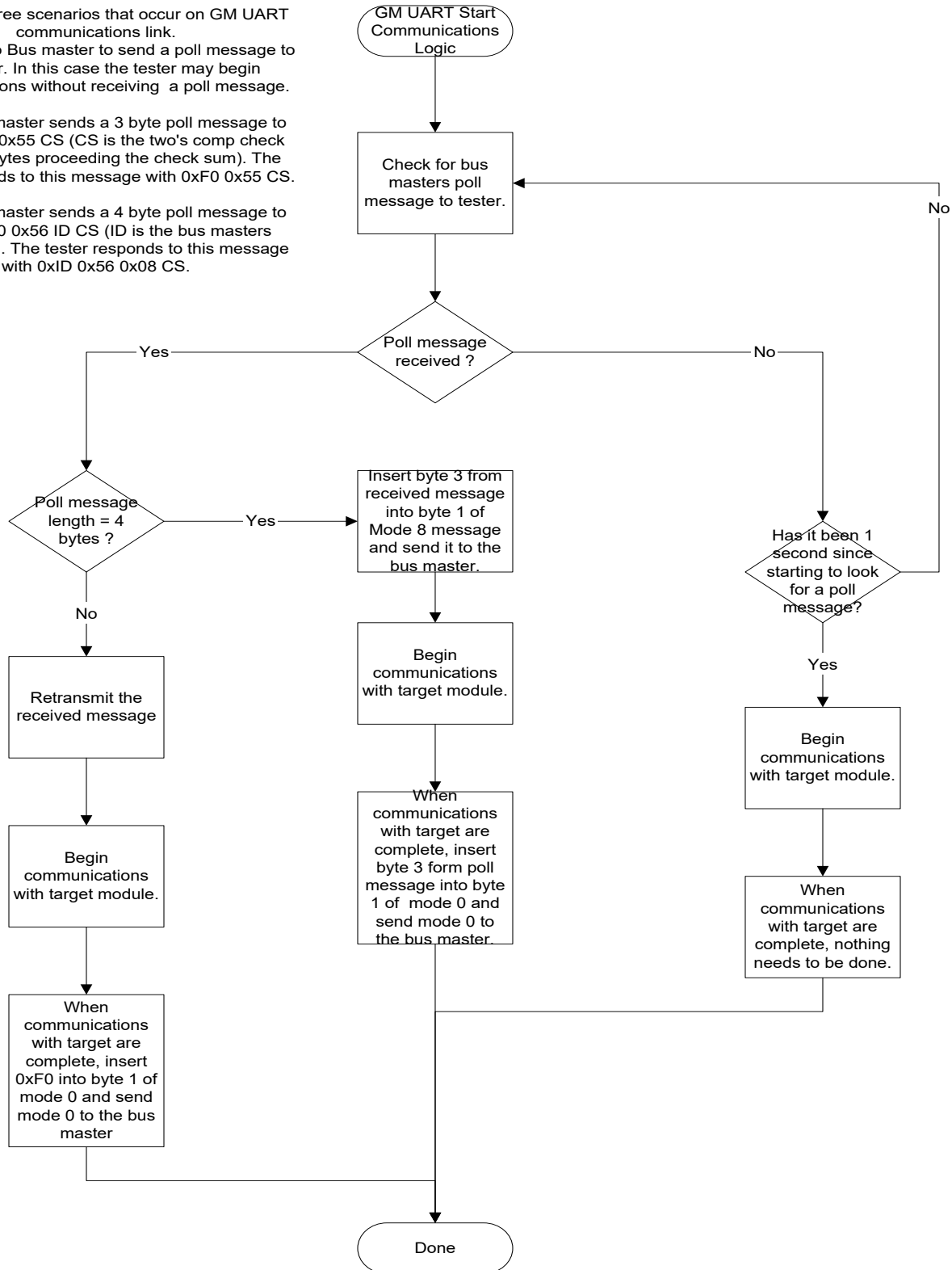


Figure 5 - Process flow for obtaining bus mastership

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.