



SURFACE VEHICLE RECOMMENDED PRACTICE

J2534™-2/BA_0500

JAN2022

Issued

2022-01

Pass-Thru Extended Feature - Base Document

RATIONALE

Each of the features contained in the original SAE J2534-2 document (API version 04.04) is being published as a separate document for ease of maintenance and extensibility. At the same time, these features are being updated to be compatible with SAE J2534-1_0500 (API version 05.00) and to address feedback received from the field.

TABLE OF CONTENTS

1.	SCOPE.....	6
1.1	Purpose.....	6
2.	REFERENCES.....	6
2.1	Applicable Documents	6
2.1.1	SAE Publications.....	6
2.1.2	ISO Publications.....	7
3.	DEFINITIONS	7
4.	DOCUMENTATIONS CONVENTIONS	7
5.	OVERVIEW OF SAE J2534-2/X_0500 DOCUMENT SET	7
6.	ACCESSING SAE J2534-2 FEATURES AND MULTIPLE DEVICES - MANDATORY FEATURE.....	10
6.1	Feature Overview	10
6.2	<pName>	10
6.2.1	Restrictions/Limitations	10
6.2.2	Examples	10
7.	CONNECTION MECHANISM - MANDATORY FEATURE.....	10
7.1	Connection Mechanism - Feature Overview.....	10
7.2	Use Case #1: Connect Using Specific Pins on a Defined Vehicle Diagnostics Connector	11
7.2.1	Example #1: Accessing the CAN Protocol on a Set of Pins not Allowed	11
7.2.2	Example #2: Opening Two CAN Channels on Two Different Sets of Pins	12
7.2.3	Example #3: Attempt to Open Two CAN Channels on Two Different Set of Pins	13
7.3	Use Case #2: Connect Using a Vendor-Specific Connector	14
7.3.1	Example #1: Opening Two ISO 9141 Channels on Two Different Sets of Pins on Vendor Specific Connector.....	14
7.4	Use Case #3: Connect without Specifying Any Connector/pin Details.....	15
7.4.1	Example #1: Application Opening Two SAE J1708 Channels Using Channel Numbers	15

SAE Executive Standards Committee 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 © 2022 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_BA_5_00

For more information on this standard, visit
https://www.sae.org/standards/content/J2534-2/BA_5_00

Documents

Document Sharing Hub

8.	DISCOVERY MECHANISM - MANDATORY FEATURE.....	16
8.1	Discovery Mechanism Feature Overview	16
8.2	Win32 Application Programming Interface.....	16
8.2.1	API Functions - Overview.....	16
8.2.2	API Functions - Detailed Information	17
9.	SWITCHING PLATFORMS AND/OR API VERSIONS WITH THE SAME PASS-THRU INTERFACE - MANDATORY FEATURE	20
9.1	Switching Platforms and/or API Versions with the Same Pass-Thru Interface Feature Overview	20
9.2	Win32 Application Programming Interface.....	21
9.2.1	API Functions - Overview.....	21
9.2.2	API Change list	21
9.2.3	IOCTL Section.....	21
9.2.4	Return Values	22
9.3	Discovery Mechanism Support	23
9.4	Example	23
10.	REPEAT MESSAGING - OPTIONAL FEATURE	23
10.1	Repeat Messaging Feature Overview.....	23
10.2	Win32 Application Programming Interface.....	23
10.2.1	API Functions - Overview.....	23
10.2.2	API Change List:	24
10.3	Discovery Mechanism Support	27
10.3.1	GET_PROTOCOL_INFO	27
11.	EXTENDED PROGRAMMING VOLTAGE SUPPORT - OPTIONAL FEATURE	28
11.1	Extended Programming Voltage Feature Overview	28
11.2	pin Usage	28
11.3	Win32 Application Programming Interface.....	29
11.3.1	API Functions - Overview.....	29
11.3.2	API Functions - Detailed Information	29
11.4	Discovery Mechanism Support	30
11.4.1	GET_RESOURCE_INFO	30
12.	NON-VOLATILE DEVICE CONFIGURATION PARAMETERS - OPTIONAL FEATURE	31
12.1	Non-Volatile Device Configuration Parameter Feature Overview	31
12.2	Win32 Application Programming Interface.....	31
12.2.1	API Functions - Overview.....	31
12.2.2	API Functions - Detailed Information	32
12.3	Discovery Mechanism Support	33
12.3.1	GET_DEVICE_INFO.....	33
13.	LISTEN ONLY MODE - OPTIONAL FEATURE.....	33
13.1	Scope of Listen Only Mode Feature Overview	33
13.2	Win32 Application Programming Interface.....	33
13.2.1	API Functions - Overview.....	33
13.2.2	API Functions - Detailed Information	34
13.3	Discovery Mechanism Support	38
13.3.1	GET_PROTOCOL_INFO	38
14.	EXTENDED READ_PIN_VOLTAGE SUPPORT - OPTIONAL FEATURE	39
14.1	Extended READ_PIN_VOLTAGE Feature Overview	39
14.2	Pin Usage.....	39
14.3	Win32 Application Programming Interface.....	39
14.3.1	API Functions - Overview.....	39
14.3.2	API Functions - Detailed Information	40
14.4	Discovery Mechanism.....	40
14.4.1	GET_DEVICE_INFO.....	40
14.4.2	GET_RESOURCE_INFO.....	41



15.	NON-BLOCKING 5 BAUD INITIALIZATION - OPTIONAL FEATURE	42
15.1	Non-Blocking 5 Baud Initialization Feature Overview	42
15.2	Win32 Application Programming Interface	42
15.2.1	API Functions - Overview	42
15.2.2	API Functions - Detailed Information	42
15.3	Discovery Mechanism Support	44
16.	NON-BLOCKING FAST INITIALIZATION - OPTIONAL FEATURE	44
16.1	Non-Blocking Fast Initialization Feature Overview	44
16.2	Win32 Application Programming Interface	44
16.2.1	API Functions - Overview	44
16.2.2	API Functions - Detailed Information	45
16.3	Discovery Mechanism Support	46
17.	ISO 15765 VERBOSE MODE - OPTIONAL FEATURE	47
17.1	ISO 15765 Verbose Mode Feature Overview	47
17.2	Win32 Application Programming Interface	47
17.2.1	API Functions - Overview	47
17.2.2	API Functions - Detailed Information	48
17.2.1	Indications	48
17.3	Discovery Mechanism Support	49
18.	SUPPORT FOR SAE J2534-1_0500 SPECIFIED PROTOCOLS - OPTIONAL FEATURE	49
18.1	Support for SAE J2534-1 Protocols Feature Overview	49
18.2	Support for CAN (Dual Wire, High Speed)	50
18.2.1	Discovery Mechanism Support	50
18.3	Support for ISO 15765	54
18.3.2	Discovery Mechanism Support	58
18.4	Support for SAE J1850 VPW	62
18.4.1	Discovery Mechanism Support	62
18.5	Support for SAE J1850 PWM	65
18.5.1	Discovery Mechanism Support	65
18.6	Support for ISO 9141	68
18.6.1	API Functions - Detailed Information	69
18.6.2	FIVE_BAUD_INIT (Updated to Support FIVE_BAUD_MOD = ISO_5_KEYBYTES)	70
18.6.3	Discovery Mechanism Support	70
18.7	Support for ISO 14230	76
18.7.1	API Functions - Detailed Information	77
18.7.2	FIVE_BAUD_INIT (Updated to Support FIVE_BAUD_MOD = ISO_5_KEYBYTES)	77
18.7.3	Discovery Mechanism Support	78
18.8	Support for SAE J2610	84
18.8.1	Discovery Mechanism Support	84
18.9	Support for Programming Voltages	87
18.9.1	Discovery Mechanism Support	87
19.	NOTES	88
19.1	Revision Indicator	88
Table 1	SAE J2534-2 document set	7
Table 2	SAE J2534 API functions	17
Table 3	IOCTL details	17
Table 4	GET_DEVICE_INFO details	18
Table 5	GET_DEVICE_INFO parameter details	18
Table 6	GET_PROTOCOL_INFO details	19
Table 7	GET_PROTOCOL_INFO parameter details	19
Table 8	GET_RESOURCE_INFO details	19
Table 9	Return values for PassThruIoctl	20
Table 10	SAE J2534 API functions	21
Table 11	IOCTL details	21

Table 12	GET_DEVICE_NAME	22
Table 13	Return values for PassThruIoctl.....	22
Table 14	SAE J2534 API functions	24
Table 15	IOCTL details	25
Table 16	START_REPEAT_MESSAGE details.....	25
Table 17	QUERY_REPEAT_MESSAGE details.....	26
Table 18	Status parameter values	26
Table 19	STOP_REPEAT_MESSAGE details.....	26
Table 20	Return values for PassThruIoctl.....	27
Table 21	GET_PROTOCOL_INFO parameters.....	28
Table 22	SAE J2534 API functions	29
Table 23	Short-to-ground IOCTL GET_RESOURCE_INFO parameter details.....	30
Table 24	SAE J2534 API functions	31
Table 25	IOCTL details	32
Table 26	GET_DEVICE_CONFIG details.....	32
Table 27	SET_DEVICE_CONFIG details	32
Table 28	IOCTL GET_DEVICE_CONFIG/SET_DEVICE_CONFIG parameter details	33
Table 29	Ioctl GET_DEVICE_INFO parameter details	33
Table 30	J2534 API functions	34
Table 31	Connect flag	34
Table 32	Return values for PassThruLogicalConnect	35
Table 33	Return values for PassThruQueueMsgs.....	36
Table 34	Return values for PassThruStartPeriodicMsg.....	37
Table 35	Return values for PassThruIoctl.....	38
Table 36	GET_PROTOCOL_INFO parameter details	38
Table 37	SAE J2534 API functions	39
Table 38	Return values for PassThruIoctl.....	40
Table 39	Ioctl GET_DEVICE_INFO parameter details	41
Table 40	GET_RESOURCE_INFO parameter details.....	41
Table 41	SAE J2534 API functions	42
Table 42	List of description codes for the ErrInd indication	42
Table 43	NON_BLOCKING_FIVE_BAUD_INIT details	43
Table 44	Non-blocking 5 baud initialization example.....	44
Table 45	IOCTL GET_PROTOCOL_INFO	44
Table 46	SAE J2534 API functions	45
Table 47	List of description codes for the ErrInd indication	45
Table 48	NON_BLOCKING_FAST_INIT details.....	45
Table 49	Non-blocking fast initialization example	46
Table 50	IOCTL GET_PROTOCOL_INFO	47
Table 51	SAE J2534 API functions	47
Table 52	List of fields for the ErrInd indication.....	48
Table 53	List of description codes for the ErrInd indication	48
Table 54	SET_CONFIG and GET_CONFIG details	49
Table 55	IOCTL GET_CONFIG/SET_CONFIG parameter details	49
Table 56	IOCTL GET_PROTOCOL_INFO	49
Table 57	IOCTL GET_DEVICE_INFO for CAN	50
Table 58	IOCTL GET_PROTOCOL_INFO for CAN	51
Table 59	IOCTL GET_RESOURCE_INFO for CAN	53
Table 60	Additional RxStatus bit definitions.....	56
Table 61	Additional TxFlags bit definitions	56
Table 62	Return values for PassThruLogicalConnect	57
Table 63	GET_CONFIG/SET_CONFIG.....	58
Table 64	IOCTL GET_CONFIG/SET_CONFIG details.....	58
Table 65	IOCTL GET_PROTOCOL_INFO for ISO 15765.....	59
Table 66	IOCTL GET_DEVICE_INFO for SAE J1850 VPW	63
Table 67	IOCTL GET_PROTOCOL_INFO for SAE J1850 VPW.....	63
Table 68	IOCTL GET_RESOURCE_INFO for SAE J1850 VPW	64
Table 69	IOCTL GET_DEVICE_INFO for SAE J1850 PWM.....	65
Table 70	IOCTL GET_PROTOCOL_INFO for SAE J1850 PWM	65
Table 71	IOCTL GET_RESOURCE_INFO for SAE J1850 PWM.....	67

Table 72	IOCTL GET_CONFIG/SET_CONFIG parameter details	69
Table 73	FIVE_BAUD_INIT details	70
Table 74	IOCTL GET_DEVICE_INFO for ISO 9141	71
Table 75	IOCTL GET_PROTOCOL_INFO for ISO 9141	71
Table 76	IOCTL GET_RESOURCE_INFO for ISO 9141	75
Table 77	IOCTL GET_CONFIG/SET_CONFIG parameter details	77
Table 78	FIVE_BAUD_INIT details	78
Table 79	IOCTL GET_DEVICE_INFO for ISO 14230	79
Table 80	IOCTL GET_PROTOCOL_INFO for ISO 14230	79
Table 81	IOCTL GET_RESOURCE_INFO for ISO 14230	83
Table 82	IOCTL GET_DEVICE_INFO for SAE J2610	84
Table 83	IOCTL GET_PROTOCOL_INFO for SAE J2610	85
Table 84	IOCTL GET_RESOURCE_INFO for SAE J2610	87
Table 85	IOCTL GET_RESOURCE_INFO for programming voltages	88

1. SCOPE

SAE J2534-1_0500 defines the pass-thru interface requirements for the reprogramming of emission related control modules.

The SAE J2534-2/X_0500 document set adds extensions to the SAE J2534-1_0500 API (version 05.00) specification so that the API can be used for features not covered in the SAE J2534-1_0500 specification. Together, these features provide a comprehensive framework for a common standard, to protect the software investment of the vehicle OEMs and scan tool manufacturers. There is no required for an SAE J2534-2/X_0500 pass-thru interface to be fully compliant with SAE J2534-1_0500. SAE J2534-2/X_0500 interfaces can implement some or all of the features specified in the SAE J2534-2/X_0500 document set.

This document must be used in conjunction with the SAE J2534-2/RE_0500 document.

1.1 Purpose

The purpose of SAE J2534-2/X_0500 document set is to extend the capabilities of an SAE J2534-1_0500 interface. The SAE J2534-2/X_0500 documents would typically be used by a wide spectrum of applications including, but not limited to, reprogramming of emission and non-emission related modules, diagnostics, end-of-line testers, etc. Some examples of the extensions offered by SAE J2534-2/X_0500 over SAE J2534-1_0500 are:

- Support for additional protocols (ex. SAE J1939, SAE J1708 etc.)
- No restriction to the number of filters allowed
- No restriction to the number of periodic transmits allowed
- Access to multiple, and simultaneous, channels of the same protocol
- No restriction to the number of logical channels allowed
- Support for additional baud rates (in addition to the ones specified in SAE J2534-1_0500)
- Support for enhanced timing parameters where applicable
- Support for extended programming voltage support

The purpose of this specific document, SAE J2534-2/BA_0500, is to provide an introduction to the SAE J2534-2/X_0500 document set. It also defines the mandatory features, defines a set of general optional features, and defines how protocols specified in SAE J2534-1_0500 shall be supported on an SAE J2534-2/X_0500 pass-thru interface.

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 J1850 Class B Data Communications Network Interface

SAE J1962 Diagnostic Connector

SAE J2534-1_0500 Recommended Practice for Pass-Thru Vehicle Programming (API version 05.00)

SAE J2534-2/RE_0500 J2534-2 Pass-Thru Extended Features - Resource Document (API version 05.00)

2.1.2 ISO Publications

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

ISO 15765-2:2016 Road Vehicles - Diagnostics on Controller Area Networks (CAN) - Network Layer Services

3. DEFINITIONS

FEATURE: Refers to a set of requirements that together form an extension to the SAE J2534 pass-thru interface. Each feature is typically defined in a separate document. Most features are optional, but there are a few mandatory ones. Unless otherwise noted, all features are optional.

FUNCTIONAL POINT: Refers to an individual requirement in a feature. A functional point can be either mandatory or optional. Unless otherwise noted, all functional points shall be mandatory.

4. DOCUMENTATIONS CONVENTIONS

In the document body and in tables:

- All SAE J2534 API functions shall be in bold print (for example, **PassThruOpen**).
- All SAE J2534 API variables shall be enclosed in "<" and ">" (for example, <DeviceID>).
- All SAE J2534 API DEFINES shall be in upper-case print (for example, STATUS_NOERROR).
- All numbers shall be in decimal unless otherwise indicated. (Hexadecimal values have a leading "0x"; for example, 0x7AF or 0x800 are hexadecimal values.)

5. OVERVIEW OF SAE J2534-2/X_0500 DOCUMENT SET

Previously, there had been a single document that contained all of the features for the SAE J2534-2 API version 04.04. For version 05.00 of the API, the old SAE J2534-2 specification is divided into multiple documents and will be updated to match SAE J2534-1_0500. Each document focuses on one feature or a set of related features. [Table 1](#) summarizes the documents contained in the set.

Table 1 - SAE J2534-2 document set

Specification Part	Description
SAE J2534-2/BA_0500	SAE J2534/2 base document (API version 05.00) • Scope • Specification structure • Enabling SAE J2534-2/BA_0500 features and accessing multiple devices (mandatory) • Connection mechanism (mandatory) • Discovery mechanism (mandatory) • Common features • Support for SAE J2534-1_0500 specified protocols
SAE J2534-2/1_0500	Single wire CAN (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details

Specification Part	Description
SAE J2534-2/2_0500	GM UART (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/3_0500	UART echo byte protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/4_0500	Honda diag-H protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/5_0500	SAE J1939 protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/6_0500	SAE J1708 protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/7_0500	TP 2.0 protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/8_0500	Fault-tolerant CAN protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/9_0500	Ethernet NDIS protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/10_0500	Analog inputs (API version 05.00) • References • Acronyms • Feature details • Pertinent discover mechanism details
SAE J2534-2/11_0500	CAN FD protocol (API version 05.00) • References • Acronyms • Feature details • Pertinent discover mechanism details
SAE J2534-2/12_0500	Digital inputs/outputs (API version 05.00)
SAE J2534-2/13_0500	Ethernet protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details

Specification Part	Description
SAE J2534-2/15_0500	IPv6 protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/16_0500	IPv4 protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/17_0500	UDP protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/18_0500	TCP client protocol (API version 05.00) • References • Acronyms • Protocol details • Pertinent discover mechanism details
SAE J2534-2/RE_0500	SAE J2534-2 resource section (API version 05.00) • Connect flag values • Protocol id values • Filter type values • IOCTL id values • GET/SET_CONFIG values • GET_DEVICE/SET_DEVICE_CONFIG values • Return values

The SAE J2534-2/BA_0500 (“base”) document provides an introduction to the SAE J2534-2/X_0500 document set. It also defines the mandatory features, defines a set of general optional features, and defines how protocols specified in SAE J2534-1_0500 shall be supported on an SAE J2534-2/X_0500 pass-thru interface.

The SAE J2534-2/RE_0500 (“resource”) document defines all of the data structures and constants reference throughout the SAE J2534-2/X_0500 document set.

Individual features are specified in SAE J2534-2/X_0500 (where X = 1,2,..., n). These features will all reference the SAE J2534-2/B_0500 and SAE J2534-2/RE_0500 documents, as well as the SAE J2534-1_0500 document. Support for these features is pass-thru vendor dependent. Each feature provides a discovery mechanism so that applications can determine what a specific pass-thru interface supports.

Within the SAE J2534-2/X_0500 document set there are both mandatory and optional features. The mandatory features are detailed the SAE J2534-2/BA_0500 document and contain the phrase “mandatory feature” in the title. Each SAE J2534-2/X_0500 interface shall support the mandatory features and may include some or all of the features specified in the SAE J2534-2/X_0500 document set.

To be considered compliant with each SAE J2534-2/X_0500 feature specification, the pass-thru interface shall implement all of the requirements, except those that are specifically listed optional. Functional points identified as optional may be implemented at the interface vendor’s discretion. The interface shall report its support of the various SAE J2534-2/X_0500 features/functional points through the discovery mechanism. As SAE J2534-2/X_0500 documents are intended to be independent, there is no concept of a fully compliant SAE J2534-2/X_0500 interface. Additionally, there is no required for an SAE J2534-2/X_0500 pass-thru interface to be fully compliant with SAE J2534-1_0500.

6. ACCESSING SAE J2534-2 FEATURES AND MULTIPLE DEVICES - MANDATORY FEATURE

6.1 Feature Overview

SAE J2534-2/X_0500 shall allow the simultaneous opening of multiple devices supported by the same DLL as long as the device Names are unique and the named device is available to be opened. When multiple devices are opened, each device can be opened in either SAE J2534-1_0500 or SAE J2534-2/X_0500 modes. The parameter <pName> in the function call **PassThruOpen** is used to enable this feature.

6.2 <pName>

J2534-1_0500 defines the overall size and the terminating character of the string pointed to by <pName>. The format of the string pointed to by <pName> is as follows:

API_DESIGNATOR:DEVICE_NAME

where:

API_DESIGNATOR is either "J2534-1" or "J2534-2"

DEVICE_NAME is the device name designated by the pass-thru interface manufacturer.

6.2.1 Restrictions/Limitations

Setting the API_DESIGNATOR to "J2534-1" will enable only SAE J2534-1_0500 mode. Setting the API_DESIGNATOR to "J2534-2" will enable SAE J2534-2/X_0500 mode (including the SAE J2534-1_0500 features that are available in SAE J2534-2_0500 mode). Setting the DEVICE_NAME portion of the string to NULL will use the default device. If <pName> does not begin with "J2534-1:" or "J2534-2:" no assumptions can be made regarding the SAE J2534-1_0500 or SAE J2534-2/X_0500 functionality.

6.2.2 Examples

For example, to open two devices from one vendor, one device (identified as "deviceA") in SAE J2534-1_0500 mode and the other in device (identified as "deviceB") in SAE J2534-2/X_0500 mode use, make consecutive calls to **PassThruOpen** with <pName> pointing to each of the strings:

J2534-1:deviceA
J2534-2:DeviceB

To open the default device in SAE J2534-2/X_0500 mode, make a call to **PassThruOpen** with <pName> pointing to the string:

J2534-2:

7. CONNECTION MECHANISM - MANDATORY FEATURE

7.1 Connection Mechanism - Feature Overview

Version 05.00 of the SAE J2534-1_0500 document introduced changes to **PassThruConnect** that allows the application to specify the connector and pins (via the parameter <ResourceStruct>) when connecting to a specific protocol. For SAE J2534-1_0500, the connector/pin combinations were limited, but this has been expanded in SAE J2534-2/X_0500.

This change has eliminated the need for the _PS and _CHx ProtocolIDs, which were used prior to version 05.00 of the API. The connection information that was implied by the _PS and _CHx ProtocolIDs is now passed as the <ResourceStruct> parameter to **PassThruConnect**.

To further extend this capability, the SAE J2534-2/RE_0500 defines new connector types (J1939_CONNECTOR, J1708_CONNECTOR, etc.) in addition to the J1962_CONNECTOR defined in SAE J2534-1_0500 specification. The resource section also defines two generic constants: DEVICE_SPECIFIC_CONNECTOR and DEVICE_SPECIFIC_CHANNEL. DEVICE_SPECIFIC_CONNECTOR is used when the application is working with a vendor specific connector/cable and DEVICE_SPECIFIC_CHANNEL is used when the application, either does not know or care about the connector pin configurations. The use of DEVICE_SPECIFIC_CONNECTOR and DEVICE_SPECIFIC_CHANNEL are intended for unique applications and their use shall be specified by the interface manufacturer.

This connection mechanism applies to protocols defined in SAE J2534-1_0500 and SAE J2534-2/X_0500. Applications must use the discovery mechanism to determine what connections a specific interface supports. All SAE J2534-2/X_0500 pass-thru interfaces shall support connection in this manner.

The following provides some use cases for the new connect mechanism.

7.2 Use Case #1: Connect Using Specific Pins on a Defined Vehicle Diagnostics Connector

Unless otherwise specified, these examples assume that the pass-thru interface being used supports the protocols, connectors, and pins being requested. If an error is detected, **PassThruConnect** in SAE J2534-1_0500 details the appropriate return values.

7.2.1 Example #1: Accessing the CAN Protocol on a Set of Pins not Allowed

This example will demonstrate an application accessing the CAN protocol on a set of pins not allowed by SAE J2534-1_0500.

```
// Assumption: Interface supports CAN channel on pins 9 and 11 of SAE J1962 Connector
```

```
RESOURCE_STRUCT resStruct;  
resStruct.Connector = J1962_CONNECTOR;  
resStruct.NumOfResources = 2;
```

```
unsigned long res[2];  
res[0] = 9; // Primary pin (CAN High)  
res[1] = 11; // Secondary pin (CAN Low)
```

```
resStruct.ResourceListPtr = &res; // set-up the resource assignment
```

```
PassThruConnect  
(  
    deviceId,  
    CAN,  
    flags,  
    baudrate,  
    resStruct,  
    pChannelID  
);
```

7.2.2 Example #2: Opening Two CAN Channels on Two Different Sets of Pins

This example will demonstrate an application opening two CAN channels on two different sets of pins.

```
// Assumption: Interface supports two CAN channels, that can be accessed either on pins 6 & 14 or pins 9 & 11
```

```
RESOURCE_STRUCT resStruct;  
resStruct.Connector = J1962_CONNECTOR;  
resStruct.NumOfResources = 2;
```

```
unsigned long res[2];  
res[0] = 6;  
res[1] = 14;
```

```
resStruct.ResourceListPtr = &res; // set-up the resource assignment for the first set of pins
```

```
PassThruConnect          // connect to first CAN channel on pins 6 & 14  
(  
    deviceId,  
    CAN,  
    flags,  
    baudrate,  
    resStruct,  
    pChannelID1  
);
```

```
res[0] = 9 ;  
res[1] = 11;          // set-up the resource assignment for the second set of pins
```

```
PassThruConnect          // connect to first CAN channel on pins 9 & 11  
(  
    deviceId,  
    CAN,  
    flags,  
    baudrate,  
    resStruct,  
    pChannelID2  
);
```

7.2.3 Example #3: Attempt to Open Two CAN Channels on Two Different Set of Pins

This example will demonstrate an application trying (and failing) to open two CAN channels on two different set of pins with a device that only supports one CAN channel.

```
// Assumption: Interface supports only one CAN channel but supports either pins 6 & 14 or
pins 9 & 11

RESOURCE_STRUCT resStruct;
resStruct.Connector = J1962_CONNECTOR;
resStruct.NumOfResources = 2;

unsigned long res[2];
res[0] = 6;
res[1] = 14;

resStruct.ResourceListPtr = &res; // set-up the resource assignment for the first set of
pins

PassThruConnect          // connect to first CAN channel on pins 6 & 14 succeeds
(
    deviceId,
    CAN,
    flags,
    baudrate,
    resStruct,
    pChannelID1
);
    res[0] = 9;
    res[1] = 11;          // set-up the resource assignment for the second set of
    pins

PassThruConnect          // connect to first CAN channel on pins 9 & 11 fails
(
    deviceId,
    CAN,
    flags,
    baudrate,
    resStruct,
    pChannelID2
);

//
// Note: Here connect fails as the only CAN resource available is already in
// use on pins 6 & 14.
// By releasing the resource and connecting again with pins 9 & 11, connect will succeed.
//
```

```
PassThruDisconnect(<pchannelID1>); // PassThruDisconnect is called to release the CAN resource

PassThruConnect // PassThruConnect to CAN channel on pins 9 & 11 succeeds
(
    deviceId,
    CAN,
    flags,
    baudrate,
    resStruct,
    pChannelID2
);
```

7.3 Use Case #2: Connect Using a Vendor-Specific Connector

The API also provides an application the ability to connect via specific pins on a vendor-defined connector. The application uses this option by setting the connector type to `DEVICE_SPECIFIC_CONNECTOR` and specifying pin(s), when using **PassThruConnect** function. It is assumed that the application is aware of the pin configurations on the vendor specific connector and that there is a vendor supplied/defined cable that ensures proper connections to end devices.

7.3.1 Example #1: Opening Two ISO 9141 Channels on Two Different Sets of Pins on Vendor Specific Connector

This example will demonstrate an application opening two ISO 9141 channels on two different sets of pins on the vendor specific connector.

```
// Assumption: Interface supports two ISO 9141 channels supported pins 23 & 24, or pin 35
(K Line only)

RESOURCE_STRUCT resStruct;
resStruct.Connector = DEVICE_SPECIFIC_CONNECTOR; // identify a Device Specific Connector
resStruct.NumOfResources = 2;

unsigned long res[2]; // Select pins 23 & 24
res[0] = 23;
res[1] = 24; // set-up the resource assignment for that connector

resStruct.ResourceListPtr = &res;

PassThruConnect // PassThruConnect to ISO 9141 channel on pins 23 & 24
    succeeds
(
    deviceId,
    ISO 9141,
    flags,
    baudrate,
    resStruct,
    pChannelID1
);
```

```
resStruct.NumOfResources = 1;    // Select pin 35
res[0] = 35;                    // set-up the second resource assignment for that
connector

PassThruConnect                // PassThruConnect to second ISO 9141 channel on pin 35
succeeds
(
    deviceId,
    ISO 9141,
    flags,
    baudrate,
    resStruct,
    pChannelID2
);
```

7.4 Use Case #3: Connect without Specifying Any Connector/pin Details

The API also provides applications the ability to access multiple channels of each protocol (if supported by the interface) without specifying the connector or pin numbers. The application uses this option by setting the connector type to `DEVICE_SPECIFIC_CHANNEL` and specifying a channel number, when using **PassThruConnect** function. The channel number starts with one and increments by one up to the number of channels supported by the interface. The pin(s) for each channel are vendor specific and must be part of the vendor interface documentation. The channels are typically accessed using custom cable built to interface vendor's specification.

7.4.1 Example #1: Application Opening Two SAE J1708 Channels Using Channel Numbers

This example will demonstrate an application opening two SAE J1708 channels using channel numbers.

```
// Assumption: Interface supports two SAE J1708 channels

RESOURCE_STRUCT resStruct;
resStruct.Connector = DEVICE_SPECIFIC_CHANNEL;    // identify a Device Specific Channel
resStruct.NumOfResources = 1;

unsigned long res[1];
res[0] = 1;    // set-up the resource assignment for the first SAE J1708
channel

resStruct.ResourceListPtr = &res;

PassThruConnect    // PassThruConnect to the first SAE J1708 channel
(
    deviceId,
    J1708,
    flags,
    baudrate,
    resStruct,
    pChannelID1
);
```

```
res[0] = 2;                // set-up resource assignment for second SAE J1708 channel

PassThruConnect            // PassThruConnect to the second SAE J1708 channel
(
    deviceId,
    J1708,
    flags,
    baudrate,
    resStruct,

pchannelID2,
    );
```

8. DISCOVERY MECHANISM - MANDATORY FEATURE

8.1 Discovery Mechanism Feature Overview

This section details the mechanism to programmatically determine/confirm the capabilities of a specific SAE J2534 interface. All SAE J2534-2/X_0500 interfaces shall support this feature.

8.2 Win32 Application Programming Interface

8.2.1 API Functions - Overview

The purpose of this feature is to provide a mechanism to programmatically determine/confirm the capabilities of a specific SAE J2534 device. This feature is accessed via calls to **PassThruIoctl** after successfully calling **PassThruOpen**. Using this feature, an application can access information about the device, a specific protocol that is supported by the device, or specific pins/connector configurations supported by the device. These capabilities are static and shall identify what is possible, which shall not change based on the current state of the device. (For example, the identification of which pins CAN can be switched to shall not be altered if some of those pins are currently in use by another protocol.)

[Table 2](#) summarizes the changes to the SAE J2534-1_0500 API functions.

Table 2 - SAE J2534 API functions

Function	Description
PassThruScanForDevices	Function implementation independent of this feature.
PassThruGetNextDevice	Function implementation independent of this feature.
PassThruOpen	Function implementation independent of this feature.
PassThruClose	Function implementation independent of this feature.
PassThruConnect	Function implementation independent of this feature.
PassThruDisconnect	Function implementation independent of this feature.
PassThruLogicalConnect	Function implementation independent of this feature.
PassThruLogicalDisconnect	Function implementation independent of this feature.
PassThruSelect	Function implementation independent of this feature.
PassThruReadMsgs	Function implementation independent of this feature.
PassThruQueueMsgs	Function implementation independent of this feature.
PassThruStartPeriodicMsg	Function implementation independent of this feature.
PassThruStopPeriodicMsg	Function implementation independent of this feature.
PassThruStartMsgFilter	Function implementation independent of this feature.
PassThruStopMsgFilter	Function implementation independent of this feature.
PassThruSetProgrammingVoltage	Function implementation independent of this feature.
PassThruReadVersion	Function implementation independent of this feature.
PassThruGetLastError	Function implementation independent of this feature.
PassThruIoctl	Add new ioctlID values.

8.2.2 API Functions - Detailed Information

8.2.2.1 IOCTL Section

[Table 3](#) details the ioctlIDs that are applicable to this feature.

Table 3 - IOCTL details

<ioctlID>	<InputPtr> Represents	<OutputPtr> Represents	Purpose
GET_DEVICE_INFO	NULL pointer	Pointer to SPARAM_LIST	To acquire the general capabilities of the device.
GET_PROTOCOL_INFO	Pointer to protocol ID	Pointer to SPARAM_LIST	To acquire the protocol specific capabilities of the device.
GET_RESOURCE_INFO	Pointer to unsigned long	Pointer to SRESOURCE_SUPPORTED	To acquire the specific connector/pin capabilities of the device.

8.2.2.1.1 GET_DEVICE_INFO

The <ioctlID> value of GET_DEVICE_INFO shall be used to request the general capabilities of a SAE J2534 device. The calling application is responsible for allocating and initializing the inputs described in [Table 4](#). Upon successful completion (a return code of STATUS_NOERROR), the corresponding <Supported> and/or <Value> elements shall have been updated by the pass-thru interface. Requests for parameters that are not supported by the device shall cause the corresponding <Supported> element to be set to NOT_SUPPORTED. In this case, the interface shall continue processing the rest of the parameters in the list and the return code shall be STATUS_NOERROR (as long as no other errors conditions exist). Some generic device parameters and their associated descriptions are detailed in [Table 5](#). The feature specific device parameters will be specified in each feature document.

Table 4 - GET_DEVICE_INFO details

Parameter	Description
<ControlTarget>	Device ID assigned by DLL during PassThruOpen
<IoctlID>	Is set to the define GET_DEVICE_INFO.
<InputPtr>	Is set to NULL, as this parameter is not used.
<OutputPtr>	Points to the structure SPARAM_LIST, where: <NumOfParms> is an INPUT, set by the application, which contains the number of SPARAM elements in the array pointed to by <ParamPtr>. <ParamPtr> is an INPUT, set by the application, which points to an array of SPARAM structures allocated by the application, where: <Parameter> is an input, set by the application, which represents the ID of the parameter requested (see Table 5 for a list of valid parameters). <Value> is typically an output, set by the Interface, which represents the parameter's value. However, for certain parameters Value may also be an INPUT set by the application (see Table 5 for more details). <Supported> is an output, set by the Interface to 0 (indicating the associated parameter is NOT_SUPPORTED) or 1 (indicating the associated parameter is SUPPORTED). The contents of Value shall remain unchanged if the parameter is NOT_SUPPORTED.

[Table 5](#) contains a list of parameters for SAE J2534-2/X_0500 interfaces. Parameters that are denoted as mandatory shall be supported by all SAE J2534-2/X_0500 interfaces.

Table 5 - GET_DEVICE_INFO parameter details

Parameter	Description	Associated Value	Mandatory
SERIAL_NUMBER	Represents the device serial number or a unique identifier defined by the vendor.	Value is an OUTPUT that, upon return, contains an unsigned long that is the device serial number or a unique identifier defined by the vendor.	No
PART_NUMBER	Represents the part number of the device.	Value is an OUTPUT that, upon return, contains an unsigned long that is the device part number.	No
DEVICE_CONNECT_MEDIA	Represents the connection media of the device.	Value is an OUTPUT that, upon return, contains an unsigned long that is the value that would be returned for <DeviceConnectionMedia> in the SDEVICE structure.	No

8.2.2.1.2 GET_PROTOCOL_INFO

The <IoctlID> value of GET_PROTOCOL_INFO is used to request the protocol specific capabilities of a SAE J2534 device. It is expected that the application will call this function for each protocol it is interested in. The calling application is responsible for allocating and initializing the inputs described in [Table 6](#). Upon successful completion (a return code of STATUS_NOERROR), the corresponding <Supported> and/or <Value> elements shall have been updated. Requests for parameters that are not supported by the device shall cause the corresponding <Supported> element to be set to NOT_SUPPORTED. In this case, the interface shall continue processing the rest of the parameters in the list and the return code shall be STATUS_NOERROR (as long as no other errors conditions exist). If the device does not support the requested protocol ID, the return value shall be ERR_PROTOCOL_ID_NOT_SUPPORTED.

Table 6 - GET_PROTOCOL_INFO details

Parameter	Description
<ControlTarget>	Device ID assigned by DLL during PassThruOpen.
<IoctlID>	Is set to the define GET_PROTOCOL_INFO.
<InputPtr>	Points to an unsigned long that contains the protocol ID (either physical or logical).
<OutputPtr>	Points to the structure SPARAM_LIST, where: <NumOfParms> is an input, set by the application, which contains the number of SPARAM elements in the array pointed to by <ParamPtr>. <ParamPtr> is an input, set by the application, which points to an array of SPARAM structures allocated by the application, where: <Parameter> is an input, set by the application, which represents the ID of the parameter requested. <Value> is typically an output, set by the Interface, which represents the parameter's value. However, for certain parameters Value may also be an INPUT set by the application. <Supported> is an output, set by the Interface to 0 (indicating the associated parameter is NOT_SUPPORTED) or 1 (indicating the associated parameter is SUPPORTED). The contents of Value shall remain unchanged if the parameter is NOT_SUPPORTED.

[Table 7](#) contains a list of parameters for SAE J2534-2/X_0500 interfaces. Parameters that are denoted as mandatory shall be supported by all SAE J2534-2/X_0500 interfaces.

Table 7 - GET_PROTOCOL_INFO parameter details

Parameter	Description	Associated Value	Mandatory
TIMESTAMP_RESOLUTION	Represents the resolution of the message timestamp in microseconds.	Value is an OUTPUT that, upon return, contains the interface's timestamp resolution, i.e., the smallest time interval that the Interface can measure and report. For example, if the device has a 1 ms internal timer, the value would be 1000.	Yes

8.2.2.1.3 GET_RESOURCE_INFO

The <IoctlID> value of GET_RESOURCE_INFO shall be used to determine if a set of pins on a connector can be used for a specific purpose. The calling application is responsible for allocating and initializing the inputs described in [Table 8](#). Upon successful completion (a return code of STATUS_NOERROR), the <Supported> element shall have been updated. Requests for combinations that are not supported by the device shall cause the <Supported> element to be set to NOT_SUPPORTED and the return code shall be STATUS_NOERROR (as long as no other errors conditions exist).

If the parameter <InputPtr> points to an invalid value, then the return code shall be ERR_INVALID_PARAMETER. If the parameter <InputPtr> points to USE_FOR_PROTOCOL, then <ResourceID> shall contain the corresponding protocol ID. If that protocol ID is not supported, invalid, or unknown, then the return code shall be ERR_PROTOCOL_ID_NOT_SUPPORTED. If the protocol ID designates a logical communication channel, then the return code shall be ERR_LOG_CHAN_NOT_ALLOWED.

Table 8 - GET_RESOURCE_INFO details

Parameter	Description
<ControlTarget>	Device ID assigned by DLL during PassThruOpen.
<IoctlID>	Is set to the define GET_RESOURCE_INFO.
<InputPtr>	Points to an unsigned long that represents the desired use of the connector/pins. Valid values are: - USE_FOR_PROTOCOL /* when ResourceList contains protocol pins */ - USE_FOR_READ_PIN_VOLTAGE /* when ResourceList contains a pin for READ_PIN_VOLTAGE */ - USE_FOR_SHORT_TO_GND /* when ResourceList contains a SHORT_TO_GROUND pin */ - USE_FOR_PROGRAMMING_VOLTAGE /* when ResourceList contains a programmable voltage pin */
<OutputPtr>	Points to the structure SRESOURCE_SUPPORTED, where: <ResourceID> is an input, set by the application that contains the protocol ID when <InputPtr> points to USE_FOR_PROTOCOL. Otherwise, this field is ignored. <ResourceStruct> is an input structure, set by the application, which details the connector and associated pin whose use is being verified. This structure is defined and described in SAE J2534-1_0500. <Supported> is an output, set by the Interface to NOT_SUPPORTED (the specified pins/connector are not available) or SUPPORTED (the specified pins/connector are available). The other elements of the structure shall remain unchanged if the parameter is NOT_SUPPORTED.

8.2.2.2 Return Values

[Table 9](#) lists the possible return values for this function.

Table 9 - Return values for PassThruIoctl

Name	Description
ERR_CONCURRENT_API_CALL	A SAE J2534 API function has been called before the previous SAE J2534 function call has completed.
ERR_DEVICE_NOT_OPEN	PassThruOpen has not successfully been called.
ERR_IOCTL_ID_NOT_SUPPORTED	<IoctlID> value is not supported (either invalid, unknown, or not applicable for the current channel).
ERR_INVALID_DEVICE_ID	PassThruOpen has been successfully called, but the current device ID is not valid.
ERR_INVALID_CHANNEL_ID	Invalid channel ID value.
ERR_DEVICE_NOT_CONNECTED	Unable to communicate with the pass-thru device.
ERR_NOT_SUPPORTED	Device does not support this API function for the associated channel ID.
ERR_LOG_CHAN_NOT_ALLOWED	Logical communication channel is not allowed for the designated physical communication channel and <ProtocolID> combination.
ERR_PROTOCOL_ID_NOT_SUPPORTED	<ProtocolID> value is not supported (either invalid or unknown).
ERR_NULL_PARAMETER	NULL pointer supplied where a valid pointer is required.
ERR_NULL_REQUIRED	A parameter that is required to be NULL is not set to NULL.
ERR_MSG_PROTOCOL_ID	<ProtocolID> in the PASSTHRU_MSG structure does not match the <ProtocolID> from the original call to PassThruConnect for the channel ID.
ERR_INVALID_MSG	Message structure is invalid for the given channel ID.
ERR_INVALID_PARAMETER	A parameter passed is either invalid, unknown, or not applicable.
ERR_IOCTL_PARAM_ID_NOT_SUPPORTED	Parameter referenced in the SCONFIG_LIST structure is not supported (either invalid, unknown, or not applicable for the current channel).
ERR_IOCTL_VALUE_NOT_SUPPORTED	<Value> referenced in the SCONFIG_LIST structure is not supported (either unknown, invalid, out of range, or not applicable for the current channel).
ERR_PIN_NOT_SUPPORTED	One or more of the following is either invalid or unknown: The pin number specified by <ResourceList.ResourceListPrt>, the connector specified by <ResourceList.Connector>, or the number of resources specified by <ResourceList.NumOfResources>.
ERR_EXCEEDED_LIMIT	Exceeded the allowed limits.
ERR_INIT_FAILED	Physical vehicle bus initialization failed.
ERR_BUFFER_OVERFLOW	Indicates a buffer overflow occurred and messages were lost.
ERR_BUFFER_TOO_SMALL	The size of <DataBuffer>, as indicated by the parameter <DataBufferSize> in the PASSTHRU_MSG structure, is too small to accommodate the full message.
ERR_FAILED	Undefined error, use PassThruGetLastError for text description.
STATUS_NOERROR	Function call was successful.

9. SWITCHING PLATFORMS AND/OR API VERSIONS WITH THE SAME PASS-THRU INTERFACE - MANDATORY FEATURE

9.1 Switching Platforms and/or API Versions with the Same Pass-Thru Interface Feature Overview

This section details the switching platforms and/or API versions feature. All SAE J2534-2/X_0500 interfaces shall support this feature. This feature shall require the following:

1. The application shall call **PassThruClose** before switching.
2. The Host Interface shall not change when the pass-thru interface is re-opened.
3. Updates to the pass-thru device driver and/or firmware (if required) shall be transparent to the application.

9.2 Win32 Application Programming Interface

9.2.1 API Functions - Overview

Some uses of a SAE J2534 pass-thru interface may require switching to a different application that may use a different platform (for example, switching from a 64-bit version to a 32-bit version) and/or a different API version (for example, switching from API version 05.00 to API version 04.04). In these cases, the application will want to avoid prompting the user to reselect the pass-thru interface.

The operation of the feature is in three stages:

1. The application shall call **PassThruIoctl** to obtain the current device name, which may be specific to the current host interface.
2. The application shall call **PassThruClose** before switching to a different platform and/or API version.
3. The application shall load the appropriate pass-thru DLL for the desired platform and/or API version. Then call **PassThruOpen** with the previously obtained device name.

9.2.2 API Change list

[Table 10](#) summarizes the changes to the SAE J2534-1_0500 API functions for this feature.

Table 10 - SAE J2534 API functions

Function	Description
PassThruScanForDevices	Function implementation independent of this feature.
PassThruGetNextDevice	Function implementation independent of this feature.
PassThruOpen	As defined in SAE J2534-1_0500.
PassThruClose	As defined in SAE J2534-1_0500.
PassThruConnect	Function implementation independent of this feature.
PassThruDisconnect	Function implementation independent of this feature.
PassThruLogicalConnect	Function implementation independent of this feature.
PassThruLogicalDisconnect	Function implementation independent of this feature.
PassThruSelect	Function implementation independent of this feature.
PassThruReadMsgs	Function implementation independent of this feature.
PassThruQueueMsgs	Function implementation independent of this feature.
PassThruStartPeriodicMsg	Function implementation independent of this feature.
PassThruStopPeriodicMsg	Function implementation independent of this feature.
PassThruStartMsgFilter	Function implementation independent of this feature.
PassThruStopMsgFilter	Function implementation independent of this feature.
PassThruSetProgrammingVoltage	Function implementation independent of this feature.
PassThruReadVersion	Function implementation independent of this feature.
PassThruGetLastError	Function implementation independent of this feature.
PassThruIoctl	Added new IOCTL ID.

9.2.3 IOCTL Section

This section lists all of the applicable <IoctlID>s used for this feature:

Table 11 - IOCTL details

<IoctlID>	<InputPtr>	<OutputPtr>	Purpose
GET_DEVICE_NAME	Pointer to an unsigned long that holds the size of the buffer. Upon return, this pointer holds the number of bytes used.	Pointer to a string that, upon return, shall contain the specific name for the current pass-thru device.	To get the specific pass-thru device name for the device currently open. This name will then be passed to a subsequent PassThruOpen call, which will open this same device.



9.2.3.1 GET_DEVICE_NAME

The <IoctlID> value of GET_DEVICE_NAME (whose value is defined in SAE J2534-2/RE_0500) is used to get the specific device Name portion of <pName> for the pass-thru device currently open. This may not necessarily be the same name that was passed to the DLL during the original call to PassThruOpen, but it shall contain enough information for a subsequent call to PassThruOpen to open the exact same device using the same host interface.

[Table 12](#) describes the function parameters for this function call.

Table 12 - GET_DEVICE_NAME

Parameter	Description
<ControlTarget>	Device ID assigned by DLL during PassThruOpen.
<IoctlID>	Is set to the define GET_DEVICE_NAME.
<InputPtr>	Shall point to an unsigned long that contains the size of the buffer pointer to by <OutputPtr>, which upon successful return shall contain the number of bytes used. If the function is not successful, then the value pointed to by <OutputPtr> shall not be altered.
<OutputPtr>	Shall be a pointer to a string, which upon return shall contain the device name string. The device name portion of this string is specific to the pass-thru interface, designated by the device ID. This string, combined with the appropriate API designator, can be used in subsequent calls to PassThruOpen. It should be noted that the device name portion of this string may not be the name originally used or the corresponding registry key name.

If the size pointed to by <InputPtr> is too small, then the return value shall be ERR_BUFFER_TOO_SMALL.

9.2.4 Return Values

[Table 13](#) lists the possible return values for this function.

Table 13 - Return values for PassThruIoctl

Name	Description
ERR_CONCURRENT_API_CALL	An SAE J2534 API function has been called before the previous SAE J2534 function call has completed.
ERR_DEVICE_NOT_OPEN	PassThruOpen has not successfully been called.
ERR_IOCTL_ID_NOT_SUPPORTED	<IoctlID> value is not supported (either invalid, unknown, or not applicable for the current channel).
ERR_INVALID_DEVICE_ID	PassThruOpen has been successfully called, but the current device ID is not valid.
ERR_INVALID_CHANNEL_ID	Invalid channel ID value.
ERR_DEVICE_NOT_CONNECTED	Pass-thru device communication error. This indicates that the pass-thru interface DLL has, at some point, failed to communicate with the pass-thru device—even though it may not currently be disconnected.
ERR_NOT_SUPPORTED	Device does not support this API function for the associated channel ID.
ERR_NULL_PARAMETER	NULL pointer supplied where a valid pointer is required.
ERR_NULL_REQUIRED	A parameter that is required to be NULL is not set to NULL.
ERR_MSG_PROTOCOL_ID	<ProtocolID> in the PASSTHRU_MSG structure does not match the <ProtocolID> from the original call to PassThruConnect for the channel ID.
ERR_INVALID_MSG	Message structure is invalid for the given channel ID.
ERR_IOCTL_PARAM_ID_NOT_SUPPORTED	Parameter referenced in the SCONFIG_LIST structure is not supported (either invalid, unknown, or not applicable for the current channel).
ERR_IOCTL_VALUE_NOT_SUPPORTED	Value referenced in the SCONFIG_LIST structure is not supported (either unknown, invalid, out of range, or not applicable for the current channel).
ERR_PIN_NOT_SUPPORTED	One or more of the following is either invalid or unknown: The pin number specified by <InputPtr->ResourceListPtr>, the connector specified by <InputPtr->Connector>, or the number of resources specified by <InputPtr->NumOfResources>.
ERR_EXCEEDED_LIMIT	Exceeded the allowed limits.
ERR_INIT_FAILED	Physical vehicle bus initialization failed.
ERR_BUFFER_OVERFLOW	Indicates a buffer overflow occurred and messages were lost.
ERR_BUFFER_TOO_SMALL	The size of <DataBuffer>, as indicated by the parameter <DataBufferSize> in the PASSTHRU_MSG structure, is too small to accommodate the full message.
ERR_FAILED	Undefined error; use PassThruGetLastError for text description. A fully compliant SAE J2534-1 pass-thru interface shall never return ERR_FAILED.
STATUS_NOERROR	Function call was successful.

9.3 Discovery Mechanism Support

There are no discovery mechanisms for this feature. If the call to **PassThruIoctl** with the ioctlID of GET_DEVICE_NAME is not supported, then the return value shall be ERR_NOT_SUPPORTED.

9.4 Example

Application is running using SAE J2534-2 (V05.00) DLL

```
...
```

```
PassThruIoctl(uldeviceID, GET_DEVICE_NAME, pBufferSize, pdeviceName);
```

```
PassThruClose(uldeviceID);
```

```
...
```

```
load SAE J2534 (V04.04) DLL
```

```
...
```

```
*pName = "J2534-2:" + *pdeviceName;
```

```
PassThruOpen(pName, pdeviceID);
```

```
...
```

10. REPEAT MESSAGING - OPTIONAL FEATURE

10.1 Repeat Messaging Feature Overview

This section details the repeat messaging feature. This feature is optional, so the discovery mechanism must be used to determine if an SAE J2534-2/X_0500 interface supports this feature.

It may not be possible for all protocols to support this feature. Each protocol section shall clearly specify if this feature can be supported for the protocol it specifies.

10.2 Win32 Application Programming Interface

10.2.1 API Functions - Overview

Some operations on ECUs require a message to be transmitted repeatedly at a rapid and predictable rate until the operation is completed. Completion of the operation is indicated by a change in a byte value in the response message to the repeated request. This feature defines an extension to the API that allows the interface to perform autonomous transmissions based on the content of messages received from a vehicle bus. The configuration of this behavior is performed by the application using new IOCTLs. In this way, the feature provides a general purpose, protocol independent method of achieving high performance repeated transmissions while certain conditions persist on a vehicle.

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

The operation of the feature is in three stages:

1. Specifying the message to transmit transmission rate and stop criteria via **PassThruIoctl START_REPEAT_MESSAGE**.
2. Querying to see if the transmission has been stopped automatically via **PassThruIoctl QUERY_REPEAT_MESSAGE**.
3. Stopping an in-progress transmission and releasing resources via **PassThruIoctl STOP_REPEAT_MESSAGE**.

In general, repeat messages are almost identical to periodic messages. Repeat messages shall have the same <DataLength> and <TxFlags> limitations as periodic messages. Repeat messages shall generate a TxDone indications (or a TxFailed indications if TxDone was requested), if enabled on the channel. Additionally, repeat messages shall have the same priority as periodic messages. The queuing mechanism used by periodic messages shall be extended to include repeat messages. Finally, repeat messages shall not violate bus idle timing parameters (like P3_MIN) or other protocol specific requirements.

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

Known uses are for this feature are with ISO 9141 and ISO 14230.

10.2.2 API Change List:

[Table 14](#) summarizes the changes to the SAE J2534-1_0500 API functions.

Table 14 - SAE J2534 API functions

Function	Description
PassThruScanForDevices	Function implementation independent of this feature.
PassThruGetNextDevice	Function implementation independent of this feature.
PassThruOpen	Function implementation independent of this feature.
PassThruClose	Function implementation independent of this feature.
PassThruConnect	Function implementation independent of this feature.
PassThruDisconnect	Function implementation independent of this feature.
PassThruLogicalConnect	Function implementation independent of this feature.
PassThruLogicalDisconnect	Function implementation independent of this feature.
PassThruSelect	Function implementation independent of this feature.
PassThruReadMsgs	Function implementation independent of this feature.
PassThruQueueMsgs	Function implementation independent of this feature.
PassThruStartPeriodicMsg	Function implementation independent of this feature.
PassThruStopPeriodicMsg	Function implementation independent of this feature.
PassThruStartMsgFilter	Function implementation independent of this feature.
PassThruStopMsgFilter	Function implementation independent of this feature.
PassThruSetProgrammingVoltage	Function implementation independent of this feature.
PassThruReadVersion	Function implementation independent of this feature.
PassThruGetLastError	Function implementation independent of this feature.
PassThruIoctl	Add new ioctlID values.

10.2.2.1 IOCTL Section

[Table 15](#) details the ioctlIDs that are applicable to this feature:

Table 15 - IOCTL details

<ioctlID>	<InputPtr> Represents	<OutputPtr> Represents	Purpose
START_REPEAT_MESSAGE	Pointer to REPEAT_MSG_SETUP	Pointer to unsigned long	Sets up and starts a repeat message transmission.
QUERY_REPEAT_MESSAGE	Pointer to unsigned long	Pointer to unsigned long	Reports the status of a repeat message transmission.
STOP_REPEAT_MESSAGE	Pointer to unsigned long	NULL pointer	Terminates an existing repeat message transmission.
GET_PROTOCOL_INFO	Pointer to protocol ID	Pointer to SPARAM_LIST	Support for discovery mechanism.

10.2.2.1.1 START_REPEAT_MESSAGE

This function is used to specify and initiate a repeat message. The parameters are specified in [Table 16](#). If this feature is not supported, then function call shall return ERR_NOT_SUPPORTED.

Table 16 - START_REPEAT_MESSAGE details

Parameter	Description
<ControlTarget>	Channel ID assigned by DLL during PassThruConnect .
<ioctlID>	Is set to the define START_REPEAT_MESSAGE.
<InputPtr>	Is a pointer to the structure REPEAT_MSG_SETUP, where: <TimeInterval> is the time interval between the end of a message transmission and the start of the next time this message is sent, in milliseconds. The valid range is 5 to 65535 ms. <Condition> specifies whether the repeat transmission should: 0 = continue until a message is received that matches the pattern 1 = continue while every received message matches the pattern <RepeatMsgData[0]> is a PASSTHRU_MSG structure that specifies the message to be repeatedly transmitted. The <DataBuffer>, <DataLength>, and <TxFlags> elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. <RepeatMsgData[1]> is a PASSTHRU_MSG structure that specifies the mask message that will be ANDed with each incoming message to mask any unimportant bits. The <DataBuffer>, <DataLength>, and <TxFlags> elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. <RepeatMsgData [2]> is a PASSTHRU_MSG structure that specifies the pattern message that will be compared to the incoming message after the mask message has been applied. If the result matches this pattern message, the designated action shall be taken. The <DataBuffer>, <DataLength>, and <TxFlags> elements of the PASSTHRU_MSG structure shall be populated and appropriate for the designated channel. Additionally, the <DataLength>, and <TxFlags> elements shall match that of the mask message.
<OutputPtr>	Is a pointer to an unsigned long, MsgID, the message ID that is assigned by the DLL.

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

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

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

ERR_EXCEEDED_LIMIT shall be returned if more than the maximum number of repeat messages supported by the interface is requested. ERR_INVALID_MSG shall be returned if the data size of an individual message exceeds maximum data size allowed for periodic messages. ERR_INVALID_MSG shall also be returned if the <TxFlags> do not match those allowed for periodic messages.

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

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

10.2.2.1.2 QUERY_REPEAT_MESSAGE

This function is used to check the status of the designated repeat message. The parameters are specified in [Table 17](#).

Table 17 - QUERY_REPEAT_MESSAGE details

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

If this function call is successful the return value shall be STATUS_NOERROR and the value of the Status parameter shall be set as follows:

Table 18 - Status parameter values

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

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

10.2.2.1.3 STOP_REPEAT_MESSAGE

This function is used to terminate a defined repeat transmission and release the specified MsgID. The parameters are specified in [Table 19](#).

Table 19 - STOP_REPEAT_MESSAGE details

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

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

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

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

10.2.2.2 Return Values

[Table 20](#) lists the possible return values for this function.

Table 20 - Return values for PassThruIoctl

Name	Description
ERR_CONCURRENT_API_CALL	A SAE J2534 API function has been called before the previous SAE J2534 function call has completed.
ERR_DEVICE_NOT_OPEN	PassThruOpen has not successfully been called.
ERR_IOCTL_ID_NOT_SUPPORTED	<ioctlID> value is not supported (either invalid, unknown, or not applicable for the current channel).
ERR_INVALID_DEVICE_ID	PassThruOpen has been successfully called, but the current device ID is not valid.
ERR_INVALID_CHANNEL_ID	Invalid channel ID value.
ERR_DEVICE_NOT_CONNECTED	Unable to communicate with the pass-thru device.
ERR_NOT_SUPPORTED	Device does not support this API function for the associated channel ID.
ERR_LOG_CHAN_NOT_ALLOWED	Logical communication channel is not allowed for the designated physical communication channel and <ProtocolID> combination.
ERR_PROTOCOL_ID_NOT_SUPPORTED	<ProtocolID> value is not supported (either invalid or unknown).
ERR_NULL_PARAMETER	NULL pointer supplied where a valid pointer is required.
ERR_NULL_REQUIRED	A parameter that is required to be NULL is not set to NULL.
ERR_MSG_PROTOCOL_ID	<ProtocolID> in the PASSTHRU_MSG structure does not match the <ProtocolID> from the original call to PassThruConnect for the channel ID.
ERR_INVALID_MSG	Message structure is invalid for the given channel ID.
ERR_EXCEEDED_LIMIT	Exceeded the allowed limits.
ERR_FAILED	Undefined error, use PassThruGetLastError for text description.
STATUS_NOERROR	Function call was successful.

10.3 Discovery Mechanism Support

This section details the ioctlIDs (accessed via **PassThruIoctl**) that shall be supported for this feature.

10.3.1 GET_PROTOCOL_INFO

The <ioctlID> value of GET_PROTOCOL_INFO is used to obtain the protocol information from the pass-thru interface. The pass-thru interface shall report support for Repeat Messaging and associated parameters through the discovery mechanism defined in “discovery mechanism” section. This feature shall support the parameters in [Table 21](#).