Drew Technologies, Inc.

SAE J2534
API
REFERENCE

Copyright 2003 Drew Technologies, Inc.

Copyright 2003 Drew Technologies, Inc.

Copyright 2003 Drew Technologies, Inc.

Copyright 2003 Drew Technologies, Inc.

Abbreviations/Acronyms

Acronym Name	Acronym Description
API	Application Programming Interface
BS	Block Size
CAN	Controller Area Network
CF	Consecutive Frame
CRC	Cyclic Redundancy Check
CS	Checksum
CTS	Continue to Send
DLL	Dynamic Link Library
ECU	Electronic Control Unit
FC	Flow Control Frame
FF	First Frame
FFS	For Future Study
FS	Flow Status
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IEEE 1394	High speed serial bus(up to 400 Mbps)
IFR	In-Frame Response(single or multiple bytes)
ISO	International Standards Organization
kbit	Kilo bit
KWP	Key Word Protocol
LSB	Least Significant Bit/Byte
MSB	Most Significant Bit/Byte
PC	Personal Computer
PCI	Protocol Control Information
PID	Parameter Identification Number
PWM	Pulse Width Modulation
SCI	Serial Communications Interface
SCP	Standard Corporate Protocol
STMin	Separation Time Minimum
TBD	To Be Determined
USB	Universal Serial Bus
VPW	Variable Pulse Width
Wi-Fi	Wireless Fidelity

Copyright 2003 Drew Technologies, Inc.

Introduction

This paper describes a communication API that connects ECUs on a vehicle network to a PC based user application running in a standard Windows® environment.

Using external reprogramming tools for updating ECU firmware and calibration settings is a growing trend. Before the J2534 specification multiple hardware and software tools were necessary to reprogram vehicles from different manufacturers. The "multiple tool" situation proved confusing and expensive for any aftermarket business that repaired or customized vehicles from a wide array of international and/or domestic manufacturers. In addition the U.S. Environmental Protection Agency(EPA) has developed vehicle network requirements that all vehicle manufacturers must meet for the 2004 model year and beyond. The EPA mandated compliance test suite(J1699) requires communicating with the vehicle network through a scan tool device. Both private business demands and new governmental regulations have provided the impetus for developing a hardware platform that is compatible with all vehicles.

The J2534 specification defines a hardware device that interfaces to the vehicle network and a PC based software Application API that controls the hardware device. The PC based Application manages all aspects of the ECU reprogramming process through the J2534 API. The hardware device is concerned with providing physical and data link layer support for the specified J2534 protocols. This arrangement allows the vehicle manufactures to restrict access to proprietary reprogramming algorithms and support files and allows one hardware device to be used for programming all vehicles.

PassThru Overview

The PassThru User Application executes on a Windows® PC and contains the proprietary software steps for reprogramming, calibrating and monitoring ECUs within the target vehicle network. The PassThru User Application contains the GUI presented to the user/operator, parses the Windows® registry to locate and dynamically link to the vendor's PassThru J2534 API/DLL, provides access to proprietary downloadable software and calibration files, maintains and enforces security policies to control user access and provides vehicle specific programming algorithms for controlling the ECU reflashing process. The generic PC must have a Win32® operating system installed(i.e. Windows® 95/98, Windows NT™, Windows® Millennium Edition, Windows® 2000, Windows® XP, ….). Also the PC may need to have an Internet connection, as some User Applications may be WEB based or require access to external WEB sites for obtaining the flashable ECU files.

The User Application references J2534 API routines to control the PassThru device and control the message flow between the PassThru device and the target vehicle's ECUs. The PassThru device does not interpret, modify or examine User Application messages destined for the vehicle network(except for ISO15765-2 protocol). Its role is to provide physical layer connectivity for the required list of network layer protocols and to take care of the data link layer formatting details for each network protocol. The PassThru device allows network messages to flow seamlessly between the ECUs connected to the vehicle network and the User Application running on the PC.

The PassThru device manufacturer supplies the hardware appliance and cable that connects to and communicates with the vehicle network. The physical interface to the vehicle network is the SAE J1962 connector, which supports the various vehicle serial data protocols. The following network layer protocols are supported:

ISO 9141
ISO 14230-4(KWP2000, Keyword Protocol)
SAE J1850 41.6 KBPS PWM(Pulse Width Modulation)
SAE J1850 10.4 KBPS VPW(Variable Pulse Width)
CAN(Controller Area Network, ISO 11898)
ISO 15765-4(CAN)
SAE J2610 DaimlerChrysler SCI(Serial Communications Interface)

Copyright 2003 Drew Technologies, Inc.

The PassThru device manufacturer also supplies the cable and any required specialized Comm Port drivers that form the communication link to the PC. This communication link can be RS-232, USB, Ethernet, IEEE1394 or Wi-Fi. The PassThru manufacturer also supplies the J2534 DLL, which is installed on the PC and referenced by the User Application. In order to reduce programming complexity the J2534 DLL does not support multi-threading so the User Application is restricted to a single thread design.

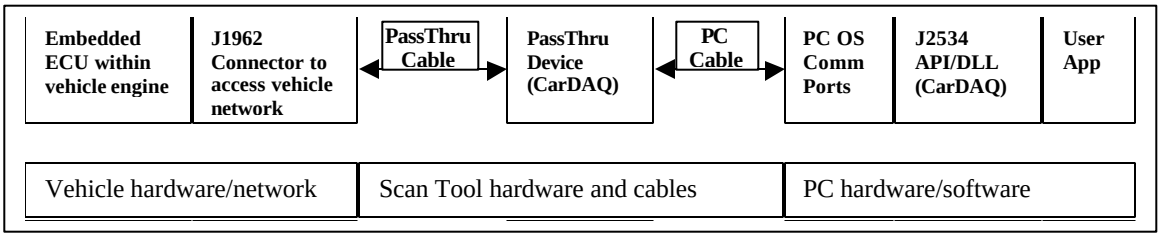


Figure 1 Relationship between the vehicle network, the PassThru device and the PC environment

As a result of the functionality division between the various PassThru components the User Application has no PC hardware dependencies and the PassThru device has no ECU specific message dependencies. This functional compartmentalization allows any User Application to work with any J2534 compliant PassThru device and allows any J2534 compliant PassThru device to connect to any vehicle network.

PassThru Device Physical and Data Link Layer Protocol Support

ISO 9141

The PassThru device must not apply voltage lower than –1.0 V or higher than +40.0 V to the open K and L lines.

The PassThru device must support transmission of the initialization address byte at 5 bit/sec(5-baud initialization) and reception of the KeyWord header at 10.4 kbit/second with a tolerance of ±0.5%.

The minimum time that the bus remains idle between retransmissions of the initialization address byte by the PassThru device is 300 milliseconds.

The PassThru device must support transmission of the UserApplication data at any of the following rates: 9.6 kbit/sec., 9615 bit/sec., 10.0 kbit/sec., 10.4 kbit/sec., 10870 bit/sec., 11905 bit/sec., 12.5 kbit/sec., 13158 bit/sec., 13889 bit/sec., 14706 bit/sec., and 15625 bit/sec. with a tolerance of ±2.0%.

For the initialization address byte and KeyWord2 inversion handshake byte the PassThru device must use no parity, 8 data bits and one start bit and one stop bit. For the synchronization pattern, KeyWord bytes and initialization address inversion handshake byte the ECU device must use odd parity, 7 data bits and one start bit and one stop bit. The PassThru device allows the UserApplication to configure odd, even or no parity(PassThruIoctl SET_CONFIG, PARITY). The PassThru device will always use one start and stop bit. The UserApplication cannot configure the number of data bits(i.e. 7-bit, 8-bit) through the generic API. The PassThru device default is presumably 8-bit data bits.

ISO 14230-4

The PassThru device supports the same physical and data link requirements listed for ISO9141 plus the additional requirements listed below.

The PassThru device must support both 5-baud initialization and fast initialization vehicle network activation methods.

Copyright 2003 Drew Technologies, Inc.

The PassThru device automatically sends TesterPresent messages to the ECUs after network initialization to keep network communications active.

The PassThru device automatically handles the RequestCorrectlyReceived-ResponsePending(0x78) ECU negative response by remaining in receive mode longer than the P2_MAX(maximum ECU response time) time interval. The PassThru device may receive multiple RequestCorrectlyReceived-ResponsePending responses until the ECU completes the request and either returns a positive or negative response(other than 0x78).

J1850PWM

The PassThru device supports two baud rates, 41.6 kbit/sec. for normal diagnostic operations and an 83.3 kbit/sec. for reprogramming.

The PassThru device must follow J1850PWM(SCP) physical layer requirements.

J1850VPW

The PassThru device supports two baud rates, 10.4 kbit/sec. for normal diagnostic operations and a 41.6 kbit/sec. for reprogramming. The PassThru device must accommodate receive and transmit frames up to 4,128 bytes in size.

CAN

The PassThru device supports sample point positioning at 80% ±2% and 68.5% ±2% of the nominal bit rate.

The PassThru device supports two baud rates 250 kbit/second and 500 kbit/second.
The PassThru device does not provide support for a flow control mechanism and the UserApplication must support any custom flow control mechanism.

ISO 15765-4

The PassThru device supports sample point positioning at 80% ±2% of the nominal bit rate.

The PassThru device supports two baud rates 250 kbit/second and 500 kbit/second. The PassThru device support s both 11-bit and 29-bit Identifiers with or without extended addressing. The PassThru device provides support for the ISO 15765-2 flow control mechanism.

SCI

Refer to J2610 DaimlerChrysler SCI for physical and data link layer details.

Copyright 2003 Drew Technologies, Inc.

PassThru API Functions

The tool vendor's PassThru API/DLL must support the functions listed below.

PassThru Function Name	PassThru Function Description
PassThruConnect……..	Establish a logical communication channel using the specified vehicle network protocol. Protocols supported can be extended by the Scan Tool vendor.
PassThruDisconnect …….	Terminate an existing logical communication channel.
PassThruReadMsgs……...	Receive network protocol messages from an existing logical communication channel.
PassThruWriteMsgs……..	Transmit network protocol messages over an existing logical communication channel.
PassThruStartPeriodicMsg ……...	Transmit network protocol messages at the specified time interval over an existing logical communication channel.
PassThruStopPeriodicMsg ……	Terminate the specified periodic message.
PassThruStartMsgFilter ……	Transmit a network protocol filter that will selectively restrict or limit network protocol messages received by the User Application.
PassThruStopMsgFilter ……	Terminate the specified message filter.
PassThruSetProgrammingVoltage ……	Set the programmable voltage level on the specified J1962 connector pin.
PassThruReadVersion ……..	Retrieve the PassThru device firmware version, DLL version and API version information.
PassThruGetLastError ……..	Retrieve the text description for the last PassThru function that generated an error.
PassThruIoctl ……	General purpose I/O control function for retrieving and setting the various network protocol timing related parameters. Can be extended by the Scan Tool vendor to provide tool specific functionality.

Copyright 2003 Drew Technologies, Inc.

The **PassThruConnect** function is used to establish a logical communication channel between the User Application and the vehicle network(via the PassThru device) using the specified network layer protocol and selected protocol options.

long PassThruConnect(
  unsigned long *ProtocolID*,
  unsigned long *Flags*,
  unsigned long **pChannelID*
);

Parameters

ProtocolID

The protocol identifier selects the network layer protocol that will be used for the communications channel. The network layer protocol identifier must be one of the values listed below.

Protocol Name	Protocol Description	Protocol Value
J1850VPW	GM / DaimlerChrysler CLASS2	0x01
J1850PWM	Ford SCP(Standard Corporate Protocol)	0x02
ISO9141	ISO9141 and ISO9141-2	0x03
ISO14230	ISO14230-4(Keyword Protocol 2000)	0x04
CAN	Raw CAN(custom flow control implemented in software)	0x05
ISO15765	ISO15765-2(CAN physical and data link layers with Network layer flow control)	0x06
SCI_A_ENGINE	J2610(DaimlerChrysler Serial Communications Interface) Configuration A for engine	0x07
SCI_A_TRANS	J2610(DaimlerChrysler Serial Communications Interface) Configuration A for transmission	0x08
SCI_B_ENGINE	J2610(DaimlerChrysler Serial Communications Interface) Configuration B for engine	0x09
SCI_B_TRANS	J2610(DaimlerChrysler Serial Communications Interface) Configuration B for transmission	0x0A
UNUSED	Reserved	0x0B - 0xFFFF
UNUSED	PassThru tool vendor specific	0x10000 - 0xFFFFFFFF

Flags

Protocol specific options that are defined by bit fields. This parameter is usually set to zero. The flag value must adhere to the bit field values listed below.

Flags Definition	Flags Bit Field	Flags Description	Flags Value
Tool Vendor extension	31-9	Tool Vendor defined	Tool Vendor specific
CAN_29BIT_ID	8	CAN ID Type	0 = 11-bit 1 = 29-bit
ISO15765_ADDR_TYPE	7	ISO15765-2 Addressing mode	0 = No extended address 1 = Extended address is first byte following the CAN ID
Reserved	6-0	Unused	Must be set to zero

CAN_29BIT_ID

This option is used to select between CAN Standard Data Frame and Extended Data Frame. The Standard Data Frame contains a 12-bit Arbitration Field that is comprised of an 11-bit Identifier and a Remote

Transmission Request(RTR) bit(page 65). The Extended Data Frame contains a 32-bit Arbitration Field where the first 11-bits are the most significant Identifier bits followed by Substitute Remote Request(SRR) bit, Identifier Extension(IDE) bit, 18-bit least significant Identifier bits and a Remote Transmission Request(RTR) bit. The CAN Identifier within the Extended Data Frame is comprised of 29-bits(page 66).

ISO15765_ADDR_TYPE

This option is used to extend the available address range for large networks. The extended address encodes both the transmitting and receiving peers that are located on a subnetwork connected to the main vehicle network. The extended address byte is part of the CAN header and immediately follows the CAN Identifier(page 67).

This option is used for interpreting receive messages as it specifies the PCI byte location within the frame(5^{th} byte for normal addressing, 6^{th} byte for extended addressing). Transmit messages always adhere the ISO15765_ADDR_TYPE setting in the *TxFlags* field.

pChannelID

Pointer to an unsigned long(4 bytes) that receives the handle to the open communications channel. The returned handle is used as an argument to other PassThru functions which require a communications channel reference.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruConnect function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined in below.

PassThruConnect Error Definition	PassThruConnect Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_INVALID_PROTOCOL_ID	Protocol identifier is not recognized. Or attempting to connect mutually exclusive physical layer protocols to the vehicle network(i.e. J1850PWM and J1850VPW or CAN and SCI_A_XXXX).
ERR_NULL_PARAMETER	pChannelID is a NULL pointer, which is an invalid address.
ERR_INVALID_FLAGS	Flags bit field contains an invalid/undefined value.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_CHANNEL_IN_USE	An existing communications channel is currently using the specified network protocol.

Remarks

This function will terminate all active protocol filters(PASS, BLOCK and FLOW_CONTROL) and all active periodic messages for the specified network protocol before establishing a new instance of the communications channel.

The J2534 API/DLL only supports one instance of a communications channel using a particular network layer protocol. Also due to pin assignment conflicts on the J1962 connector some physical layer protocols are mutually exclusive(i.e. J1850PWM and J1850VPW, CAN and SCI_A_XXXX, etc). The valid protocol combinations that can be active simultaneously on the vehicle bus are listed below.

Copyright 2003 Drew Technologies, Inc.

Data Link Protocol 1	Data Link Protocol 2	Data Link Protocol 3
J1850VPW	ISO9141	CAN
J1850VPW	ISO9141	ISO15765
J1850VPW	ISO14230	CAN
J1850VPW	ISO14230	ISO15765
J1850PWM	ISO9141	CAN
J1850PWM	ISO9141	ISO15765
J1850PWM	ISO14230	CAN
J1850PWM	ISO14230	ISO15765
J1850VPW	SCI_A_XXXX	
J1850VPW	SCI_B_XXXX	
J1850PWM	SCI_A_XXXX	
J1850PWM	SCI_B_XXXX	

The code sample below is an example of the PassThruConnect function call.

```
unsigned long status;
unsigned long Flags;
unsigned long ChannelID;  /* Logical channel identifier returned by PassThruConnect */
char errstr[256];

/*
** Select Extended CAN Data Frame and Network Address Extension.
*/
Flags = (CAN_29BIT_ID | ISO15765_ADDR_TYPE);

status = PassThruConnect(CAN, Flags, &ChannelID);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruConnect failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruDisconnect** function is used to terminate an existing logical communication channel between the User Application and the vehicle network(via the PassThru device). Once disconnected the channel identifier or handle is invalid. For the associated network protocol this function will terminate the transmitting of periodic messages and the filtering of receive messages. The PassThru device periodic and filter message tables will be cleared.

```
long PassThruDisconnect(
   unsigned long ChannelID
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruDisconnect function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruDisconnect Error Definition	PassThruDisconnect Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.

Remarks

This function will terminate all active protocol filters(PASS, BLOCK and FLOW_CONTROL) and all active periodic messages for the specified network protocol.

The code sample below is an example of the PassThruDisconnect function call.

```
unsigned long status;
unsigned long ChannelID;  /* Logical channel identifier returned by PassThruConnect */
char errstr[256];

status = PassThruDisconnect(ChannelID);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruDisconnect failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruReadMsgs** function is used to receive network protocol messages, receive indications and transmit indications from an existing logical communication channel. The network protocol messages will flow from the PassThru device to the User Application.

```
long PassThruReadMsgs(
   unsigned long ChannelID,
   PASSTHRU_MSG *pMsg
   unsigned long *pNumMsgs,
   unsigned long Timeout
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

pMsg
Pointer to the message structure where the J2534 API/DLL will write the receive message(s). For reading more than one message, this must be a pointer to an array of PASSTHRU_MSG structures.
Refer to PASSTHRU_MSG structure definition on page 46.

pNumMsgs
Pointer to the variable that contains the number of PASSTHRU_MSG structures that are allocated for receive frames. The API regards this value as the maximum number of receive frames that can be returned to the UserApplication. On function completion this variable will contain the actual number of receive frames contained in the PASSTHRU_MSG structure. The number of receive messages returned may be less than the number requested by the UserApplication.

Timeout
Timeout interval(in milliseconds) to wait for read completion. A value of zero instructs the API/DLL to read buffered receive messages and return immediately. A nonzero timeout value instructs the API/DLL to return after the timeout interval has expired. The API/DLL will not wait the entire timeout interval if an error occurs or the specified number of messages have been read.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruReadMsgs function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruReadMsgs Error Definition	PassThruReadMsgs Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_NULL_PARAMETER	pMsg and/or pNumMsgs are a NULL pointer, which is an invalid address.
ERR_TIMEOUT	Timeout violation. PassThru device is unable to read specified number of messages from the vehicle network. The actual number of messages returned is written in NumMsgs.
ERR_BUFFER_EMPTY	PassThru device could not read any messages from the vehicle network.
ERR_BUFFER_OVERFLOW	PassThru device experienced a buffer overflow and receive

Copyright 2003 Drew Technologies, Inc.

messages were lost.

Remarks

This function also handles transmit and receive indications from the PassThru device.

For ISO15765 the PassThru device sends a transmit confirmation indication for each UserApplication frame successfully transmitted. The RxStatus TX_MSG_TYPE bit will be set to one, the ExtraDataIndex value set to zero(the original transmit frame is not returned) and the Data array will contain the CAN header address information.

For the LOOPBACK option(refer to PassThruIoctl section) the Passthru device sends a transmit confirmation indication for each successfully transmitted UserAplication frame. The RxStatus TX_MSG_TYPE bit will be set to one, the ExtraDataIndex will be the same value as the DataSize and the Data array will contain the original UserApplication transmit frame. The Timestamp is the time that the last message byte was transmitted.

For ISO15765 the PassThru device sends a receive indication when it assembles the first frame of a multi-segmented transfer. The RxStatus ISO15765_FIRST_FRAME bit will be set to one, the ExtraDataIndex will be zero and the Data array will contain the CAN header address information. The Timesatmp is the time that the first message byte was received.

The code sample below demonstrates how to setup the PassThruReadMsgs function parameters. On successful completion the NumMsgs parameter will contain the actual number of messages received from the PassThru device. Refer to PASSTHRU_MSG structure definition on page 46.

```
typedef struct {
   unsigned long ProtocolID;        /* vehicle network protocol */
   unsigned long RxStatus;          /* receive message status */
   unsigned long TxFlags;           /* transmit message flags */
   unsigned long Timestamp;         /* receive message timestamp(in microseconds) */
   unsigned long DataSize;          /* byte size of message payload in the Data array */
   unsigned long ExtraDataIndex;    /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];        /* message payload or data */
} PASSTHRU_MSG;

unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */
unsigned long NumMsgs;
PASSTHRU_MSG Msg[2];
unsigned long Timeout;
char errstr[256];

/*
** Establish a ISO15765 communication channel to the vehicle network.
*/
status = PassThruConnect(ISO15765, 0x00000000, &ChannelID);

/*
** Initialize the PASSTHRU_MSG structure to all zeroes.
** Set the ProtocolID to select protocol frames of interest.
** The API/DLL will fill in RxStatus, TxStaus, Timestamp, DataSize, ExtraDataIndex and Data
** after the function call completes.
*/
memset(&Msg, 0, sizeof(Msg));
```

Copyright 2003 Drew Technologies, Inc.

```
Msg[0].ProtocolID = ISO15765;
Msg[1].ProtocolID = ISO15765;

/*
** Indicate that PASSTHRU_MSG array contains two messages.
*/
NumMsgs = 2;

/*
** API/DLL should read first two messages in receive queue and immediately return. If there aren't any
** messages in the receive queue then API/DLL will return ERR_BUFFER_EMPTY.
*/
Timeout = 0;

status = PassThruReadMsgs(ChannelID, &Msg, &NumMsgs, Timeout);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruReadMsgs failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruWriteMsgs** function is used to transmit network protocol messages over an existing logical communication channel. The network protocol messages will flow from the User Application to the PassThru device.

```
long PassThruWriteMsgs(
   unsigned long ChannelID,
   PASSTHRU_MSG *pMsg,
   unsigned long *pNumMsgs,
   unsigned long Timeout
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

pMsg
Pointer to the message structure containing the UserApplication transmit message(s).
For sending more than one message, this must be a pointer to an array of PASSTHRU_MSG structures.
Refer to PASSTHRU_MSG structure definition on page 46.

PNumMsgs
Pointer to the variable that contains the number of PASSTHRU_MSG structures that are allocated for transmit frames. On function completion this variable will contain the actual number of messages sent to the vehicle network. The transmitted number of messages may be less than the number requested by the UserApplication.

Timeout
Timeout interval(in milliseconds) to wait for transmit completion. A value of zero instructs the API/DLL to queue as many transmit messages as possible and return immediately. A nonzero timeout value instructs the API/DLL to wait for the timeout interval to expire before returning. The API/DLL will not wait the entire timeout interval if an error occurs or the specified number of messages have been sent.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruWriteMsgs function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruWriteMsgs Error Definition	PassThruWriteMsgs Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_NULL_PARAMETER	pMsg and/or pNumMsgs are a NULL pointer, which is an invalid address.
ERR_TIMEOUT	Timeout violation. PassThru device is unable to send specified number of messages to the vehicle network. The actual number of messages sent is written in NumMsgs.
ERR_MSG_PROTOCOL_ID	The specified protocol type within the message structure is different from the protocol associated with the communications channel(ChannelID) when it was opened.
ERR_BUFFER_FULL	PassThru device could not queue any more transmit messages

Copyright 2003 Drew Technologies, Inc.

	destined for the vehicle network.
ERR_INVALID_MSG	Message contained a min/max length, ExtraData support or J1850PWM specific source address conflict violation.

Remarks

The UserApplication can monitor its transmit traffic by configuring the LOOPBACK option(refer to PassThruIoctl section) and by testing the RxStatus error flags in the transmit confirmation frames returned by the PassThru device.

For ISO15765 the PassThru device sends a transmit confirmation indication for each UserApplication frame successfully transmitted. The RxStatus TX_MSG_TYPE bit will be set to one, the ExtraDataIndex value set to zero(the original transmit frame is not returned) and the Data array will contain the CAN header address information.

The code sample below demonstrates how to setup the PassThruWriteMsgs function parameters. On successful completion the NumMsgs parameter will contain the actual number of messages sent by the PassThru device. Refer to PASSTHRU_MSG structure definition on page 46.

```
typedef struct {
   unsigned long ProtocolID;        /* vehicle network protocol */
   unsigned long RxStatus;          /* receive message status */
   unsigned long TxFlags;           /* transmit message flags */
   unsigned long Timestamp;         /* receive message timestamp(in microseconds) */
   unsigned long DataSize;          /* byte size of message payload in the Data array */
   unsigned long ExtraDataIndex;    /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];        /* message payload or data */
} PASSTHRU_MSG;

unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */
unsigned long NumMsgs;
unsigned long Timeout;
PASSTHRU_MSG Msg;
char errstr[256];

status = PassThruConnect(J1850VPW, 0x00000000, &ChannelID);

/*
** Initialize the PASSTHRU_MSG structure to all zeroes.
** Set the ProtocolID to select protocol frames of interest.
** Timestamp and ExtraDataIndex are not used for transmit messages.
** TxFlags does not contain any J1850VPW specific options.
*/
memset(&Msg, 0, sizeof(Msg));
Msg.ProtocolID = J1850VPW;

/*
** Create a Diagnostic command message to reset all Emission-related diagnostic information stored by
** all ECUs within the vehicle network.
** Program the Priority/Message type byte(first Header byte) for 3-byte form of the consolidated header,
** functional addressing mode for the target address and a message body that will contain diagnostic
** command/status messages.
**
** Priority/message type options selected(refer to J2178-1 for Detailed Header Formats):
** Priority:                Header Bits (7-5) = 6, (0 = highest, 7 = lowest)
```

Copyright 2003 Drew Technologies, Inc.

```
** Header Type            Header Bit  (4)    = 0, three byte consolidated header
** In-Frame Response      Header Bit  (3)    = 0, In-Frame response required
** Addressing Type        Header Bit  (2)    = 0, Functional
** Type Modifier          Header Bits (1-0) = 1, Message Type is broadcast
*/
Msg.Data[0] = 0x61;       /* Priority/Message type, first Header byte */

/*
** Program the target address(second Header byte) for the legislated diagnostic command message type.
** (refer to J2178-4 for Message Definitions)
*/
Msg.Data[1] = 0x6A;       /* Target functional address, second Header byte */

/*
** Program the physical source address(third Header byte) of the PassThru device.
** (refer to J2178-4 for Physical Address Assignments)
*/
Msg.Data[2] = 0xF1;       /* PassThru device physical source address, third Header byte */

/*
** Program the Diagnostic Test Mode(first Data byte) value for selecting Reset Emission-Related
** Diagnostic Information. It is expected that all ECUs will respond to this test mode command with
** a status message that has a one byte data field that contains 0x44.
** (refer to J1979 for Diagnostic Test Modes)
*/
Msg.Data[3] = 0x04;       /* Test Mode 4, reset emission-related diagnostic information, first Data byte */
Msg.DataSize = 4;         /* Mode 0x04 message contains 4 bytes */

/*
** Indicate that PASSTHRU_MSG array contains just a single message.
*/
NumMsgs = 1;

/*
** API/DLL should place transmit message in transmit queue and immediately return.
*/
Timeout = 0;

status = PassThruWriteMsgs(ChannelID, &Msg, &NumMsgs, Timeout);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruWriteMsgs failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

The **PassThruStartPeriodicMsg** function is used to repetitively transmit network protocol messages at the specified time interval over an existing logical communication channel. The periodic messages will flow from the User Application to the PassThru device. There is a limit of ten periodic messages per network layer protocol.

```
long PassThruStartPeriodicMsg(
   unsigned long ChannelID,
   PASSTHRU_MSG *pMsg,
   unsigned long *pMsgID,
   unsigned long TimeInterval
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

pMsg
Pointer to the message structure containing the User Application's periodic message.
Refer to PASSTHRU_MSG structure definition on page 46.

pMsgID
Pointer to the variable that receives the handle to the periodic message. The returned handle is used as an argument to PassThruStopPeriodicMsg to identify a specific periodic message.

TimeInterval
Time interval(in milliseconds) at which the periodic message is repetitively transmitted. The acceptable range is 5 t0 65,535 milliseconds.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruStartPeriodicMsg function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

StartPeriodicMsg Error Definition	PassThruStartPeriodicMsg Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_NULL_PARAMETER	pMsg and/or pNumMsgs are a NULL pointer, which is an invalid address.
ERR_ INVALID_TIME_INTERVAL	The TimeInterval value is outside the specified range.
ERR_MSG_PROTOCOL_ID	The specified protocol type within the message structure is different from the protocol associated with the communications channel(ChannelID) when it was opened.
ERR_ EXCEEDED_LIMIT	The limit(ten) of periodic messages has been exceeded for the protocol associated the communications channel.
ERR_INVALID_MSG	Message contained a min/max length, ExtraData support or J1850PWM specific source address conflict violation.

Copyright 2003 Drew Technologies, Inc.

Remarks

If the PassThru device has a resource or timing issue between transmitting a periodic message and a generic UserApplication message then transmitting the periodic message will take precedence.

In order to add more periodic messages after the ERR_EXCEEDED_LIMIT return code, the User Application is expected to delete one or more existing periodic messages. This will free up space in the PassThru devices periodic message table and allow the addition of more periodic messages.

The code sample below demonstrates how to setup the PassThruStartPeriodicMsg function parameters. On successful completion the MsgID parameter will contain the message handle that is required for the PassThruStopPeriodicMsg function call. Refer to PASSTHRU_MSG structure definition on page 46.

```
typedef struct {
   unsigned long ProtocolID;        /* vehicle network protocol */
   unsigned long RxStatus;          /* receive message status */
   unsigned long TxFlags;           /* transmit message flags */
   unsigned long Timestamp;         /* receive message timestamp(in microseconds) */
   unsigned long DataSize;          /* byte size of message payload in the Data array */
   unsigned long ExtraDataIndex;    /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];        /* message payload or data */
} PASSTHRU_MSG;

typedef struct {
   unsigned long NumOfBytes;        /* Number of functional addresses in array */
   unsigned char *BytePtr;          /* pointer to functional address array */
} SBYTE_ARRAY;

typedef struct {
   unsigned long Parameter;         /* Name of configuration parameter */
   unsigned long Value;             /* Value of configuration parameter */
} SCONFIG

typedef struct {
   unsigned long NumOfParams;       /* size of SCONFIG array */
   SCONFIG *ConfigPtr;              /* array containing configuration item(s) */
} SCONFIG_LIST;

SCONFIG CfgItem;
SCONFIG_LIST Input;
SBYTE_ARRAY InputMsg;

unsigned char FuncAddr[2];          /* Functional address array – address values defined in J2178-4 */
unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */
unsigned long MsgID;
unsigned long TimeInterval;
PASSTHRU_MSG Msg;
char errstr[256];

status = PassThruConnect(1850PWM, 0x00000000, &ChannelID);

/*
** Setup the J1850PWM transceiver's LAN message filtering registers(physical source address register
** and address look-up table registers). These registers determine which Network messages from the
```

```
** vehicle LAN will enter the transceiver's receive register. Their default powerup values cause all
** Network messages to be ignored.
**
** Set the PassThru device physical source address. This value is written to the J1850PWM transceivers
** physical address register. Physical address values are defined in J2178-1.
*/
CfgItem.Parameter = NODE_ADDRESS;
CfgItem.Value = 0xF1;

Input.NumOfParams = 1;
Input.ConfigPtr = &CfgItem;

status = PassThruIoctl(ChannelID, SET_CONFIG, (void *)&Input, (void *)NULL);

/*
** Set the functional addresses for nodes of interest that are connected to the vehicle LAN.
** This value is written to the J1850PWM transceiver's look-up table register block.
** Functional address values defined in J2178-4.
*/
FuncAddr[0] = 0x0A;       /* Engine Air Intake functional address. */
FuncAddr[1] = 0x12;       /* Throttle functional address. */

InputMsg.NumOfBytes = 2;                  /* Functional address array contains two addresses. */
InputMsg.BytePtr = &FuncAddr[0];          /* Assign pointer to functional address array. */

status = PassThruIoctl(ChannelID, ADD_TO_FUNCT_MSG_LOOKUP_TABLE, (void *)&InputMsg,
                       (void *)NULL);

memset(&Msg, 0, sizeof(Msg));

/*
** Create a Diagnostic command periodic message to obtain supported PIDs from powertrain ECUs.
** Program the Priority/Message type byte(first Header byte) for 3-byte form of the consolidated header,
** functional addressing mode for the target address and a message body that will contain diagnostic
** command/status messages.
**
** Priority/message type options selected(refer to J2178-1 for Detailed Header Formats):
** Priority:              Header Bits (7-5) = 6, (0 = highest, 7 = lowest)
** Header Type            Header Bit  (4)   = 0, three byte consolidated header
** In-Frame Response      Header Bit  (3)   = 1, In-Frame response not allowed
** Addressing Type        Header Bit  (2)   = 0, Functional
** Type Modifier          Header Bits (1-0) = 0, Message Type is function command/status
*/
Msg.ProtocolID = 1850PWM;
Msg.Data[0] = 0x68;       /* Priority/Message type, first Header byte */

/*
** Program the target address(second Header byte) for the legislated diagnostic command message type.
** (refer to J2178-4 for Message Definitions)
*/
Msg.Data[1] = 0x6A;       /* Target functional address, second Header byte */

/*
** Program the physical source address(third Header byte) of the PassThru device.
** (refer to J2178-4 for Physical Address Assignments)
*/
```

Copyright 2003 Drew Technologies, Inc.

```
Msg.Data[2] = 0xF1;         /* PassThru device physical source address, third Header byte */

/*
** Program the Diagnostic Test Mode(first Data byte) value for selecting Request Current Powertrain
** Diagnostic Data. (refer to J1979 for Diagnostic Test Modes)
*/
Msg.Data[3] = 0x01;         /* Test Mode 1, request current powertrain diagnostic data, first Data byte */

/*
** Program the Parameter Identifier(second Data byte) value for selecting the supported PIDs request .
** It is expected that Powertrain specific ECUs will respond with a status message containing a 4 byte bit
** encoded data field that indicates support or non-support for PIDs 0x01 through 0x20.
*/
Msg.Data[4] = 0x00;         /* Request Indication of Supported Parameters(PIDs), second Data byte */
Msg.DataSize = 5;           /* Mode 0x01, PID 0x00 diagnostic command message contains 5 bytes */

/* The PassThru device will transmit the periodic message every 4.9 seconds. */
TimeInterval = 4900;

status = PassThruStartPeriodicMsg(ChannelID, &Msg, &MsgID, TimeInterval);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruStartPeriodicMsg failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

The **PassThruStopPeriodicMsg** function is used to terminate the specified periodic message. Once terminated the message identifier or handle value is invalid.

```
long PassThruStopPeriodicMsg(
   unsigned long ChannelID,
   unsigned long MsgID
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

MsgID
The message identifier or handle assigned by the J2534 API/DLL when PassThruStartPeriodicMsg function was called.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruStopPeriodicMsg function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

StopPeriodicMsg Error Definition	PassThruStopPeriodicMsg Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_INVALID_MSG_ID	The message identifier or handle is not recognized.

Remarks

The code sample below is an example of the PassThruStopPeriodicMsg function call. It assumes that a successful PassThruStartPeriodicMsg function call has previously occurred.

```
unsigned long status;
unsigned long ChannelID;        /* Logical channel identifier returned by PassThruConnect */
unsigned long MsgID;            /* Periodic Message handle returned by PassThruStartPeriodicMsg */
char errstr[256];

status = PassThruStopPeriodicMsg(ChannelID, MsgID);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruStopPeriodicMsg failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruStartMsgFilter** function is used to setup a network protocol filter that will selectively restrict or limit network protocol messages received by the PassThru device. The filter messages will flow from the User Application to the PassThru device. There is a limit of ten filter messages per network layer protocol.

The PassThru device will pass all vehicle network receive frames to the UserApplication when no filters are defined.

The CLEAR_RX_BUFFER(PassThruIoctl function) command must be used prior to setting filters to ensure that the receive queue only contains receive frames that adhere to the filter criteria. The PassThruStartMsgFilter function does not cause existing receive messages to be removed from the PassThru device receive queue.

```
long PassThruStartMsgFilter(
   unsigned long ChannelID,
   unsigned long FilterType,
   PASSTHRU_MSG *pMaskMsg,
   PASSTHRU_MSG *pPatternMsg,
   PASSTHRU_MSG *pFlowControlMsg,
   unsigned long *pMsgID
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

FilterType
The message filter types and filter Identifier values are defined below.

PASS_FILTER	PassThru device adds receive messages matching the Mask and Pattern criteria to its receive message queue.
BLOCK_FILTER	PassThru device ignores receive messages matching the Mask and Pattern criteria.
FLOW_CONTROL_FILTER	PassThru device adds receive messages matching the Mask and Pattern criteria to its receive message queue. The PassThru device transmits a flow control message(only for ISO 15765-4) when receiving multi-segmented frames.

Filter Name	Filter Value
PASS_FILTER	0x00000001
BLOCK_FILTER	0x00000002
FLOW_CONTROL_FILTER	0x00000003

pMaskMsg
Pointer to the message structure containing the User Application's mask data. The mask message is applied by the PassThru device to every receive message. The mask is used to isolate the receive message header sections(s) that are of interest to the User Application. A "1" bit value means that the corresponding message bit will be examined; a "0" bit value means that the corresponding message bit will be ignored. Refer to PASSTHRU_MSG structure definition on page 46.

pPatternMsg
Pointer to the message structure containing the User Application's pattern data. The PassThru device compares the Pattern message to the receive message after it is ANDed with the Mask message.

Copyright 2003 Drew Technologies, Inc.

For a PASS_FILTER if the Pattern message matches the result of received message ANDed with the Mask message, then the PassThru device adds that message to its receive message queue. Otherwise that receive message will be ignored.

For a BLOCK_FILTER if the Pattern message matches the result of received message ANDed with the Mask message, then the PassThru device ignores that message. Otherwise that receive message will be added to the PassThru device receive message queue.

For a FLOW_CONTROL_FILTER if the Pattern message matches the result of received message ANDed with the Mask message, then the PassThru device adds that message to its receive message queue. Otherwise that receive message will be ignored.

Receive message data bytes outside the range of the pattern are not used in the comparison with the Pattern message. Refer to PASSTHRU_MSG structure definition on page 46.

pFlowControlMsg

The FlowControl message is only valid for the FLOW_CONTROL_FILTER and the *pFlowControlMsg* pointer must be set to NULL for the PASS_FILTER or BLOCK_FILTER.

Pointer to the message structure containing the User Application's flow control message. The User Application specifies either an 11-bit or 29-bit CAN Identifier and an optional one byte extended CAN Address(used for identifying subnetworks connected to CAN gateways). The 11-bit CAN Identifier must be represented as a 4-byte object with the unused 2 bytes and 5 bits set to zero. The 29-bit CAN Identifier is also represented as a 4-byte object with the 3 unused bits set to zero.

This FlowControl mechanism is ISO15765 specific. The ISO15765-2 protocol is a network layer protocol devised for exchanging datagrams between CAN nodes or between a CAN node and an external PassThru device. The ISO15765-2 protocol defines a segmentation method that allows for large transfers(up to 4095 data bytes) to be segmented or broken into a series of CAN frames(up to 7 data bytes).

The FlowControl mechanism is used in a peer to peer data transfer where the transmitting peer is sending more data than can fit into one CAN frame. The transmitting peer splits the data between multiple CAN frames making the data transfer multi-framed or multi-segmented. The purpose of FlowControl is to allow the receiving peer to regulate the rate at which frames are sent by the transmitting peer. The receiving peer's FlowControl message contains a BlockSize and SeparationTime parameters, which are used by the transmitting peer to shape its data transfer into consecutive frame bursts. The BlockSize is the number of consecutive frames that can be sent in a burst before waiting for the next FlowControl message. The SeparationTime is the minimum time to pause between transmitting consecutive frames.

The FlowControl message is transmitted by the PassThru device when the receive message ANDed with the Mask message matches the Pattern message and the receive message type is FirstFrame or the receive message type is ConsecutiveFrame and the receive frame is the last frame of the consecutive frame burst. FlowControl message filtering occurs only on the CAN Identifier portion of the CAN frame.

The User Application specifies the CAN Identifier(either 11-bit or 29-bit) and the extended address byte(if configured) for the FlowControl message. Extended addressing is configured by PassThruConnect(*Flags*, option ISO15765_ADDR_TYPE) or configured by PassThruWriteMsgs(*TxFlags*, option ISO15765_ADDR_TYPE in PASSTHRU_MSG structure).

The PassThru device will supply the Protocol Control Information(PCI) bytes for the FlowControl message. The PCI data includes message type(FlowControl, FirstFrame and ConsecutiveFrame) and associated message specific parameters FlowStatus(ContinueToSend, Wait or Overflow), BlockSize and SeparationTime minimum. The PassThru device will use the default BlockSize(0, FlowControl frames not used) and SeparationTimeMin(0, no time pause between consecutive frames) values or use BS and

STmin set by the User Application through the PassThruIoctl function(SET_CONFIG).

Refer to **Flow Control Filter** section(page 64) for a detailed description of using a flow control filter.
Refer to PASSTHRU_MSG structure definition on page 46.

pMsgID
Pointer to the variable that receives the handle to the message filter. The returned handle is used as an argument to PassThruStopMsgFilter to identify a specific message filter.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruStartMsgFilter function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

StartMsgFilter Error Definition	PassThruStartMsgFilter Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_NULL_PARAMETER	pMsg and/or pNumMsgs are a NULL pointer, which is an invalid address.
ERR_MSG_PROTOCOL_ID	The specified protocol type within the message structure is different from the protocol associated with the communications channel(ChannelID) when it was opened.
ERR_ EXCEEDED_LIMIT	The limit(ten) of filter messages has been exceeded for the protocol associated the communications channel.
ERR_INVALID_MSG	Message contained a min/max length, ExtraData support or J1850PWM specific source address conflict violation.

Remarks

In the case where both the BLOCK_FILTER and PASS_FILTER are both set, the policy is to apply the BLOCK_FILTER before the PASS_FILTER. A receive message matching a BLOCK_FILTER criteria will be discarded and not be compared against any PASS_FILTER criteria.

For the PASS_FILTER and BLOCK_FILTER filters the Mask and Pattern messages must be defined and the FlowControl message must be undefined(NULL pFlowControlMsg pointer). For the FLOW_CONTROL_FILTER filter the Mask, Pattern and FlowControl messages must all be defined.

Refer to **Flow Control Program Example**(page 75) for a FLOW_CONTROL_FILTER code example.

The code sample below demonstrates how to setup a PASS_FILTER using the PassThruStartMsgFilter function parameters. On successful completion the FilterID parameter will contain the filter handle that is required for the PassThruStopMsgFilter function call. Refer to PASSTHRU_MSG structure definition on page 46.

```
typedef struct {
   unsigned long ProtocolID        /* vehicle network protocol */
   unsigned long RxStatus;         /* receive message status */
   unsigned long TxFlags;          /* transmit message flags */
   unsigned long Timestamp;        /* receive message timestamp(in microseconds) */
   unsigned long DataSize;         /* byte size of message payload in the Data array */
```

```
   unsigned long ExtraDataIndex;    /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];        /* message payload or data */
} PASSTHRU_MSG;

typedef struct {
   unsigned long NumOfBytes;        /* Number of ECU target addresses in array */
   unsigned char *BytePtr;          /* Pointer to ECU target address array */
} SBYTE_ARRAY;

SBYTE_ARRAY InputMsg;
SBYTE_ARRAY OutputMsg;
PASSTHRU_MSG MaskMsg;
PASSTHRU_MSG PatternMsg;
PASSTHRU_MSG Msg[2];

unsigned char EcuAddr[1];           /* ECU target address array */
unsigned char KeyWord[2];           /* Keyword identifier array */
unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */
unsigned long FilterID;
unsigned long NumMsgs;
char errstr[256];

status = PassThruConnect(ISO9141, 0x00000000, &ChannelID);

/*
** Setup FIVE_BAUD_INIT parameters to initialize target ECUs for communicating on the serial bus.
*/
EcuAddr[0] = 0x33;          /* Target functional address of legislated OBD system */

InputMsg.NumOfBytes = 1;            /* ECU target address array contains one address. */
InputMsg.BytePtr = &EcuAddr[0];  /* Assign pointer to ECU target address array. */

OutputMsg.NumOfBytes = 0;                   /* KeyWord array is empty. */
OutputMsg.BytePtr = &KeyWord[0];            /* Assign pointer to KeyWord array. */

/*
** Complete FIVE_BAUD_INIT initialization sequence so ECU can detect diagnostic request messages
** and send diagnostic response messages on the serial bus.
*/
status = PassThruIoctl(ChannelID, FIVE_BAUD_INIT, (void *)&InputMsg, (void *)&OutputMsg);

memset(&MaskMsg, 0, sizeof(MaskMsg));

/*
** Set filter to select network protocol messages using the 3 byte form of the consolidated header,
** functional addressing mode for the target address and a message body that will contain diagnostic
** command/status messages. The filter is set to examine the first two bytes of the three byte header and
** completely ignore the third header byte(source address).
*/
MaskMsg.ProtocolID = ISO9141;

/* Isolate 3 byte header(H bit) and functional addressing(Y bit) in priority/type byte of message header. */
MaskMsg.Data[0] = 0x14;  /* First Mask byte set to isolate H bit and Y bit in first header byte. */

/*
```

Copyright 2003 Drew Technologies, Inc.

```
** Isolate 7 most significant bits of the target address byte of the message header.
** The LSB determines whether the message type is a command(bit = 0) or a status(bit =1).
** The filter is set to ignore the message type bit.
*/
MaskMsg.Data[1] = 0xFE;/* 2nd Mask byte set to isolate 7 MSB of target address in 2nd header byte.  */
MaskMsg.DataSize = 2;     /* Mask message contains 2 bytes */

memset(&PatternMsg, 0, sizeof(PatternMsg));

/*
** Compare H bit and Y bit fields from first header byte to zero value. This selects receive messages with
** the 3 byte header and target address using functional addressing.
*/
PatternMsg.ProtocolID = ISO9141;
PatternMsg.Data[0] = 0x00;              /* First Pattern byte, H bit and Y bit values must be zero. */

/*
** Compare target address type to the legislated diagnostic message value. The Mask is set to ignore
** whether the diagnostic message type is command or status.
*/
PatternMsg.Data[1] = 0x6A;/* Second Pattern byte,  diagnostic message value defined in J2178-4. */
PatternMsg.DataSize = 2;   /* Pattern message contains 2 bytes */

status = PassThruStartMsgFilter(ChannelID, PASS_FILTER, &MaskMsg, &PatternMsg, NULL,
                                        &FilterID,);
if (status != STATUS_NOERROR)
{
            /*
            ** PassThruStartMsgFilter failed! Get descriptive error string.
            */
            PassThruGetLastError(&errstr[0]);

            /*
            ** Display Error dialog box and/or write error description to Log file.
            */
}

/*
** Set the ProtocolID to select ISO9141 protocol frames of interest.
** The API/DLL will fill in RxStatus, TxStaus, Timestamp, DataSize, ExtraDataIndex and Data
** after the function call completes.
*/
memset(&Msg, 0, sizeof(Msg));
Msg[0].ProtocolID = MsISO9141;
Msg[1].ProtocolID = ISO9141;

/*
** Indicate that PASSTHRU_MSG array contains two messages.
*/
NumMsgs = sizeof(Msg)/sizeof(Msg[0]);

/*
** API/DLL should read first two filtered messages in receive queue and immediately return. If there aren't
** any messages in the receive queue then API/DLL will return ERR_BUFFER_EMPTY.
*/
status = PassThruReadMsgs(ChannelID, &Msg, &NumMsgs, 0);
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruStopMsgFilter** function is used to terminate the specified network protocol filter. Once terminated the filter identifier or handle value is invalid.

```
long PassThruStopMsgFilter(
   unsigned long ChannelID,
   unsigned long MsgID
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

MsgID
The message identifier or handle assigned by the J2534 API/DLL when PassThruStartMsgFilter function was called.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruStopMsgFilter function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

StartMsgFilter Error Definition	PassThruStartMsgFilter Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_INVALID_MSG_ID	The message identifier or handle is not recognized.

Remarks

The code sample below is an example of the PassThruStopMsgFilter function call. It assumes that a successful PassThruStartMsgFilter function call has previously occurred.

```
unsigned long status;
unsigned long ChannelID; /* Logical channel identifier returned by PassThruConnect */
unsigned long FilterID;     /* Message filter handle returned by PassThruStartMsgFilter */
char errstr[256];

status = PassThruStopMsgFilter(ChannelID, FilterID);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruStopMsgFilter failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);
        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

The **PassThruSetProgrammingVoltage** function is used to set the programmable voltage level on the specified J1962 connector pin. Only one pin can have a specified voltage applied at a time. The UserApplication must turn the voltage off(VOLTAGE_OFF option) on the selected pin before applying voltage to a different pin. It is permissible to program pin 15 for SHORT_TO_GROUND and program another pin to a voltage level.

The User Application must protect against applying incorrect voltage levels to any of the programmable pins.

long PassThruSetProgrammingVoltage(
  unsigned long *PinNumber*,
  unsigned long *Voltage*
);

Parameters

PinNumber
The J1962 connector pin to which the PassThru device will apply the specified voltage.

J1962 Pin Description	PinNumber Value
Auxiliary Output Pin non J1962 connectors	0
Pin 6 - *Discretionary (GMLAN HS CAN High)	6
Pin 9 - *Discretionary (GM ALDL)	9
Pin 11 - *Discretionary (GMLAN MS CAN Low)	11
Pin 12 - *Discretionary	12
Pin 13 - *Discretionary	13
Pin 14 - *Discretionary (GMLAN HS CAN Low)	14
Pin 15 L line of ISO 9141-2 and ISO 14230-4	15

*Discretionary implies that J1979 specification leaves the pin available for the manufacturers use.

Voltage
The voltage value(in millivolts) that will be applied to the specified pin. The PassThru device supports a programmed voltage range of 5000mV to 20000mV with a tolerance of ±100 millivolts. The voltage change slope has a maximum settling time of 1 millisecond.

Voltage Definition	Voltage Value
Programmed Voltage (pins 0, 6, 9, 11, 12, 13 & 14)	0x00001388 (5000 mV) to 0x00004E20 (20000 mV)
SHORT_TO_GROUND (pin 15 only)	0xFFFFFFFE
VOLTAGE_OFF (all pins)	0xFFFFFFFF

Return Values

Copyright 2003 Drew Technologies, Inc.

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruSetProgrammingVoltage function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

SetProgrammingVoltage Error Definition	PassThruSetProgrammingVoltage Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_NOT_SUPPORTED	Voltage value not supported.
ERR_PIN_INVALID	Unknown pin number.

Remarks

This function is a request directed to the PassThru device and does not depend on an active communication channel to the vehicle network. The default state of the programmable pins is VOLTAGE_OFF.

The SCI protocol has a more complicated procedure for setting the voltage on the programmable pins. The User Application selects the PinNumber and sets the Voltage value to VOLTAGE_OFF. The SCI_TX_VOLTAGE bit in the *TxFlags* field of the PASSTHRU_MSG for the next transmit message must be set. After the UserApplication calls PassThruWriteMsg function, the PassThru device will pulse the voltage to 20 V DC on the selected pin after it transmits the message onto the vehicle network.

The code sample below is an example of the PassThruSetProgrammingVoltage function call.

```
unsigned long PinNumber;
unsigned long status;
char errstr[256];

PinNumber = 15;

status = PassThruSetProgrammingVoltage(PinNumber, SHORT_TO_GROUND);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruSetProrammingVoltage failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

The **PassThruReadVersion** function is used to retrieve the PassThru device firmware version, the PassThru device DLL version and the version of the J2534 specification that was referenced. The version information is in the form of NULL terminated strings. The User Application must dimension the version character arrays to be at least 80 characters in size.

long PassThruReadVersion(
  char **pFirmwareVersion*,
  char * *pDllVersion*,
  char **pApiVersion*
);

Parameters

pFirmwareVersion
Pointer to Firmware Version array, which will receive the firmware version string in XX.YY format. The PassThru device supplies this string.

pDllVersion
Pointer to DLL Version array, which will receive the DLL version string in XX.YY format. The PassThru device installed DLL supplies this string.

pApiVersion
Pointer to API Version array, which will receive the API version string in XX.YY format. The PassThru device installed DLL supplies this string. It corresponds to the ballot date of the J2534 specification that was referenced by the PassThru DLL implementers.

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruReadVersion function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruReadVersion Error	PassThruReadVersion Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_NULL_PARAMETER	pFirmwareVersion, pDllVersion and/or pApiVersion is a NULL pointer, which is an invalid address

Remarks

This function is a request directed to the PassThru device and does not depend on an active communication channel to the vehicle network.

The code sample below is an example of the PassThruReadVersion function call.

Copyright 2003 Drew Technologies, Inc.

```
char FirmwareVersion[80];
char DllVersion[80];
char ApiVersion[80]
char errstr[256];
unsigned long status;

status = PassThruReadVersion(FirmwareVersion, DllVersion, ApiVersion);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruSetReadVersion failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

The **PassThruGetLastError** function is used to retrieve the text description for the last PassThru function that generated an error. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThru function call. The error information is in the form of a NULL terminated string. The User Application must dimension the error character array to be at least 80 characters in size.

```
long PassThruGetLastError(
   char *pErrorDescription
);
```

Parameters

pErrorDescription
Pointer to Error Description array, which will receive the error description string.

Return Values

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruGetLastError Error	PassThruGetLastError Error Description
ERR_NULL_PARAMETER	pErrorDescription is a NULL pointer, which is an invalid address.

Remarks

This function is a request directed to the PassThru device and does not depend on an active communication channel to the vehicle network.

This function does not reset or alter the error code generated by the other PassThru function calls.

The code sample below is an example of the PassThruGetLastError function call.

```
char ErrorMsg[80]
char errstr[256];
unsigned long status;

status = PassThruGetLastError(ErrorMsg);
if (status != STATUS_NOERROR)
{

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

The **PassThruIoctl** function is a general purpose I/O control function for modifying the vehicle network interface characteristics, measuring battery and voltage levels and clearing internal message tables and Receive/Transmit queues of the PassThru device. The input and output structures are dependent on the selected Ioctl function. The Ioctl supported function list is described below.

Ioctl Function Name	Ioctl Function Description
GET_CONFIG ……...	Retrieve network protocol timing parameters for controlling Receive/Transmit Frame transfers.
SET_CONFIG ……	Set network protocol timing parameters for controlling Receive/Transmit Frame transfers.
READ_VBATT ……	Read voltage on pin 16 of the J1962 connector.
READ_PROG_VOLTAGE ……	Read the voltage from the J1962 connector pin that was set by PassThruSetProgrammingVoltage
FIVE_BAUD_INIT ……...	Initiate a 5-baud initialization sequence.
FAST_INIT ……	Initiate a fast initialization sequence.
CLEAR_TX_BUFFER ……..	Clear all buffered messages from the transmit queue.
CLEAR_RX_BUFFER ……..	Clear all buffered messages from the receive queue.
CLEAR_PERIODIC_MSGS …….	Clear all periodic messages from the periodic message table.
CLEAR_MSG_FILTERS ……..	Clear all message filters from the message filter table.
CLEAR_FUNCT_MSG_LOOKUP_TABLE ……	Clear Functional Message Look-up Table.
ADD_TO_FUNCT_MSG_LOOKUP_TABLE …….	Add functional addresses to the Functional Message Look-up Table.
DELETE_FROM_FUNCT_MSG_LOOUP_TABLE	Delete functional addresses from the Functional Message Look-up Table.

```
long PassThruIoctl(
   unsigned long ChannelID,
   unsigned long IoctlID,
   void *pInput,
   void *pOutput
);
```

Parameters

ChannelID
The logical communication channel identifier assigned by the J2534 API/DLL when the communication channel was opened via the PassThruConnect function.

IoctlID
Ioctl function identifier, refer to Ioctl detailed function description below.

pInput
Input structure pointer, refer to Ioctl detailed function description below.

pOutput
Output structure pointer, refer to Ioctl detailed function description below.

PassThruIoctl Function Name	Ioctl ID Value	Input Structure Pointer	Output Structure Pointer
GET_CONFIG ……	0x01	SCONFIG_LIST	NULL
SET_CONFIG ……	0x02	SCONFIG_LIST	NULL
READ_VBATT ……	0x03	NULL	unsigned long
READ_PROG_VOLTAGE ……	0x0E	NULL	unsigned long
FIVE_BAUD_INIT ……..	0x04	SBYTE_ARRAY	SBYTE_ARRAY
FAST_INIT ……..	0x05	PASSTHRU_MSG	PASSTHRU_MSG
CLEAR_TX_BUFFER …….	0x07	NULL	NULL
CLEAR_RX_BUFFER …….	0x08	NULL	NULL
CLEAR_PERIODIC_MSGS ……	0x09	NULL	NULL
CLEAR_MSG_FILTERS …….	0x0A	NULL	NULL
CLEAR_FUNCT_MSG_LOOKUP_TABLE ……..	0x0B	NULL	NULL
ADD_TO_FUNCT_MSG_LOOKUP_TABLE ……	0x0C	SBYTE_ARRAY	NULL
DELETE_FROM_FUNCT_MSG_LOOUP_TABLE …..	0x0D	SBYTE_ARRAY	NULL
Reserved ……	0x0F - 0xFFFF	FFS	FFS
Scan Tool Vendor specific ……..	0x10000 - 0xFFFFFFFF	TBD	TBD

Return Values

To obtain a descriptive error string for each numeric error code use PassThruGetLastError. This function should be called immediately after an error code is returned as any intervening successful function call will wipe out the error code set by the most recently failed PassThruIoctl function call.

If the function succeeds the return value is STATUS_NOERROR. If the function fails, the return value is a nonzero error code and is defined below.

PassThruIoctl Error Definition	PassThruIoctl Error Description
ERR_DEVICE_NOT_CONNECTED	PassThru device is not connected to the PC.
ERR_FAILED	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_NULL_PARAMETER	pInput and/or pOutput is a NULL pointer and is required to be a valid address.
ERR_INVALID_CHANNEL_ID	Communication channel identifier or handle is not recognized.
ERR_INVALID_IOCTL_ID	Ioctl Identifier is not recognized.
ERR_NOT_SUPPORTED	pInput parameter or value is not recognized.

Remarks

The individual Ioctl functions are described below. The function examples contain the definitions for the SCONFIG_LIST and SBYTE_ARRAY structures. Refer to the **PassThru Message Structure** section(page 46) for the detailed description of the PASSTHRU_MSG structure.

Copyright 2003 Drew Technologies, Inc.

GET_CONFIG

This function retrieves the various network protocol-timing parameters that control the transmission/reception of network frames by the PassThru device. The code sample below demonstrates how to setup the GET_CONFIG function parameters. On successful completion the PassThru device DATA_RATE value will be placed in CfgItem.Value. The values for multiple GET_CONFIG network parameters may be obtained with one function call by initializing an array of SCONFIG items.

```
typedef struct {
   unsigned long Parameter;          /* Name of configuration parameter */
   unsigned long Value;              /* Value of configuration parameter */
} SCONFIG

typedef struct {
   unsigned long NumOfParams;        /* sizeof SCONFIG array */
   SCONFIG *ConfigPtr;               /* array containing configuration item(s) */
} SCONFIG_LIST;

unsigned long status;
unsigned long ChannelID;             /* Logical channel identifier returned by PassThruConnect */
SCONFIG CfgItem;
SCONFIG_LIST Input;
char errstr[256];

status = PassThruConnect(J1850VPW, 0x00000000, &ChannelID);

CfgItem.Parameter = DATA_RATE;
CfgItem.Value = 0;

Input.NumOfParams = 1;
Input.ConfigPtr = &CfgItem;

status = PassThruIoctl(ChannelID, GET_CONFIG, (void *)&Input, (void *)NULL);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruIoctl(GET_CONFIG) failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

SET_CONFIG

This function sets the various network protocol timing parameters that control the transmission/reception of network frames by the PassThru device. The code sample below demonstrates how to setup the SET_CONFIG function parameters. On successful completion the PassThru device DATA_RATE value will be set to the value placed in CfgItem.Value. The values for multiple SET_CONFIG network parameters may be set with one function call by initializing an array of SCONFIG items.

```
typedef struct {
   unsigned long Parameter;          /* Name of configuration parameter */
   unsigned long Value;              /* Value of configuration parameter */
} SCONFIG;

typedef struct {
   unsigned long NumOfParams;        /* sizeof SCONFIG array */
   SCONFIG *ConfigPtr;               /* array containing configuration item(s) */
} SCONFIG_LIST;

unsigned long status;
unsigned long ChannelID;             /* Logical channel identifier returned by PassThruConnect */
SCONFIG CfgItem;
SCONFIG_LIST Input;
char errstr[256];

status = PassThruConnect(J1850VPW, 0x00000000, &ChannelID);

CfgItem.Parameter = DATA_RATE;
CfgItem.Value = 10400;

Input.NumOfParams = 1;
Input.ConfigPtr = &CfgItem;

status = PassThruIoctl(ChannelID, SET_CONFIG, (void *)&Input, (void *)NULL);
if (status != STATUS_NOERROR)
{
        /*
        ** PassThruIoctl(SET_CONFIG) failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}
```

Copyright 2003 Drew Technologies, Inc.

GET_CONFIG and SET_CONFIG Parameter Names

All the GET_CONFIG and SET_CONFIG accessible parameters that control the PassThru device communication capabilities are listed below.

Parameter Name	ID Value	Parameter Value Range	Parameter Description
DATA_RATE	0x01	5 – 5000000	Baud rate value used for vehicle network. No default value specified.
RESERVED	0x02		Unused.
LOOPBACK	0x03	0(OFF)	0 = Do not echo transmitted messages to the Receive queue.
		1(ON)	1 = Echo transmitted messages to the Receive queue. Default value is 0(OFF).
NODE_ADDRESS	0x04	0x00-0xFF	J1850PWM specific, physical address for node of interest in the vehicle network. Default is no nodes are recognized by scan tool.
NETWORK_LINE	0x05	0(BUS_NORMAL) 1(BUS_PLUS) 2(BUS_MINUS)	J1850PWM specific, network line(s) active during message transfers. Default value is 0(BUS_NORMAL).
P1_MIN	0x06	0x0-0xFFFF	ISO 9141 specific, minimum ECU inter-byte time (in milliseconds) for responses. Default value is 0 milliseconds.
P1_MAX	0x07	0x-0xFFFF	ISO 9141 specific, maximum ECU inter-byte time (in milliseconds) for responses. Default value is 20 milliseconds.
P2_MIN	0x08	0x0-0xFFFF	ISO 9141 specific, minimum ECU response time (in milliseconds) to a tester request or between ECU responses. Default value is 25 milliseconds.
P2_MAX	0x09	0x-0xFFFF	ISO 9141 specific, maximum ECU response time (in milliseconds) to a tester request or between ECU responses. Default value is 50 milliseconds.
P3_MIN	0x0A	0x0-0xFFFF	ISO 9141 specific, minimum ECU response time (in milliseconds) between end of ECU response and next tester request. Default value is 55 milliseconds.
P3_MAX	0x0B	0x0-0xFFFF	ISO 9141 specific, maximum ECU response time (in milliseconds) between end of ECU response and next tester request. Default value is 5000 milliseconds.
P4_MIN	0x0C	0x0-0xFFFF	ISO 9141 specific, minimum tester inter-byte time (in milliseconds) for a request. Default value is 5 milliseconds.
P4_MAX	0x0D	0x0-0xFFFF	ISO 9141 specific, maximum tester inter-byte time (in milliseconds) for a request. Default value is 20 milliseconds.
W1	0x0E	0x0-0xFFFF	ISO 9141 specific, maximum time(in milliseconds) from the address byte end to synchronization pattern start. Default value is 300 milliseconds.
W2	0x0F	0x0-0xFFFF	ISO 9141 specific, maximum time(in milliseconds) from the synchronization byte end to key byte 1 start. Default value is 20 milliseconds.
W3	0x10	0x0-0xFFFF	ISO 9141 specific, maximum time(in milliseconds) between key byte 1 and key byte 2. Default value is 20 milliseconds.
W4	0x11	0x0-0xFFFF	ISO 9141 specific, maximum time(in milliseconds) between key byte 2 and its inversion from the tester. Default value is 50 milliseconds.

Copyright 2003 Drew Technologies, Inc.

Parameter Name	ID Value	Parameter Value Range	Parameter Description
W5	0x12	0x0-0xFFFF	ISO 9141 specific, minimum time(in milliseconds) before the tester begins retransmission of the address byte. Default value is 300 milliseconds.
TIDLE	0x13	0x0-0xFFFF	ISO 9141 specific, bus idle time required before starting a fast initialization sequence. Default value is W5 value.
TINL	0x14	0x0-0xFFFF	ISO 9141 specific, the duration(in milliseconds) of the fast initialization low pulse. Default value is 25 milliseconds.
TWUP	0x15	0x0-0xFFFF	ISO 9141 specific, the duration(in milliseconds) of the fast initialization wake-up pulse. Default value is 50 milliseconds.
PARITY	0x16	0(NO_PARITY) 1(ODD_PARITY) 2(EVEN_PARITY)	ISO9141 specific, parity type for detecting bit errors. Default value is 0(NO_PARITY).
BIT_SAMPLE_POINT	0x17	0-100	CAN specific, the desired bit sample point as a percentage of bit time. Default value is 80%.
SYNCH_JUMP_WIDTH	0x18	0-100	CAN specific, the desired synchronization jump width as a percentage of the bit time. Default value is 15%.
RESERVED	0x19		Unused.
T1_MAX	0x1A	0x0-0xFFFF	SCI_X_XXXX specific, the maximum interframe response delay. Default value is 20 milliseconds.
T2_MAX	0x1B	0x0-0xFFFF	SCI_X_XXXX specific, the maximum interframe request delay. Default value is 100 milliseconds.
T4_MAX	0x1C	0x0-0xFFFF	SCI_X_XXXX specific, the maximum intermessage response delay. Default value is 20 milliseconds.
T5_MAX	0x1D	0x0-0xFFFF	SCI_X_XXXX specific, the maximum intermessage request delay. Default value is 100 milliseconds.
ISO15765_BS	0x1E	0x0-0xFF	ISO15765 specific, the block size for segmented transfers. The scan tool may override this value to match the capabilities reported by the ECUs. Default value is 0.
ISO15765_STMIN	0x1F	0x0-0xFF	ISO15765 specific, the separation time for segmented transfers. The scan tool may override this value to match the capabilities reported by the ECUs. Default value is 0.
RESERVED	0x20 to 0xFFFF		Unused.
TOOL EXTENSION	0x10000 to 0xFFFFFFFF		Tool manufacturer specific.

Copyright 2003 Drew Technologies, Inc.

READ_VBATT

This function is used to obtain the voltage level value from pin 16 on the J1962 connector that the PassThru device plugs into. The READ_VBATT command is a request directed to the PassThru device and does not depend on an active communication channel. The ChannelID parameter is ignored by the API/DLL and should be set to zero by the UserApplication.

The code sample below demonstrates how to setup the READ_VBATT function parameters. On successful completion the pin 16 voltage level will be placed in the VoltageLevel variable. The voltage level value is expressed in milli-volts and is rounded up to the nearest tenth of a volt.

```
unsigned long status;
unsigned long ChannelID;          /* Logical channel identifier returned by PassThruConnect */
unsigned long VoltageLevel;

status = PassThruConnect(J1850VPW, 0x00000000, &ChannelID);

status = PassThruIoctl(0, READ_VBATT, (void *)NULL, (void *)&VoltageLevel);
```

READ_PROG_VOLTAGE

This function is used to obtain the voltage level value from the J1962 connector pin that was set by the User Application via the PassThruSetProgrammingVoltage function. The READ_PROG_VOLTAGE command is a request directed to the PassThru device and does not depend on an active communication channel. The ChannelID parameter is ignored by the API/DLL and should be set to zero by the UserApplication.

The code sample below demonstrates how to setup the READ_PROG_VOLTAGE function parameters. On successful completion the PassThruSetProgrammingVoltage specified pin voltage level will be placed in the VoltageLevel variable. The voltage level value is expressed in milli-volts and is rounded up to the nearest tenth of a volt.

```
unsigned long status;
unsigned long ChannelID;          /* Logical channel identifier returned by PassThruConnect */
unsigned long VoltageLevel;

status = PassThruConnect(J1850VPW, 0x00000000, &ChannelID);

status = PassThruIoctl(0, READ_PROG_VOLTAGE, (void *)NULL, (void *)&VoltageLevel);
```

Copyright 2003 Drew Technologies, Inc.

FIVE_BAUD_INIT

This function is ISO9141 and ISO14230-4(KWP2000) specific. This function causes the PassThru device to initiate a 5-baud initialization sequence with the target ECU. The ECUs on the ISO9141 bus require initialization in order to communicate with the PassThru device. The 5-baud initialization method specifies that the PassThru device send a one byte ECU target address at 5 baud on the ISO9141 bus lines. The target ECU responds with a baud rate synchronization pattern(used by the PassThru Device to gauge the baud rate) followed by two key word bytes, which form an encoded communication identifier. The PassThru device returns the 2-byte KeyWord parameters to the UserApplication. The UserApplication must decode the communication identifier and configure the PassThru device transmission parameters(via PassThruIoctl) so the PassThru device can correctly send and receive messages over the vehicle network.

This function must be called after the PassThruConnect function and the UserApplication must configure the ECU specified transmission parameters or else the PassThru device will not be able transfer messages to/from the vehicle network.

The code sample below demonstrates how to setup the FIVE_BAUD_INIT function parameters. On successful completion the KeyWord array will contain the target ECU's key word identifiers.

```
typedef struct {
   unsigned long NumOfBytes;     /* Number of ECU target addresses in array */
   unsigned char *BytePtr;       /* Pointer to ECU target address array */
} SBYTE_ARRAY;

SBYTE_ARRAY InputMsg;
SBYTE_ARRAY OutputMsg;
unsigned long status;
unsigned long ChannelID;         /* Logical channel identifier returned by PassThruConnect */

unsigned char EcuAddr[1];        /* ECU target address array */
unsigned char KeyWord[2];        /* Keyword identifier array */

EcuAddr[0] = 0x33;          /* Initialization address used to activate all ECUs */

InputMsg.NumOfBytes = 1;         /* ECU target address array contains one address. */
InputMsg.BytePtr = &EcuAddr[0];  /* Assign pointer to ECU target address array. */

OutputMsg.NumOfBytes = 0;                /* KeyWord array is empty. */
OutputMsg.BytePtr = &KeyWord[0];         /* Assign pointer to KeyWord array. */

status = PassThruConnect(ISO9141, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, FIVE_BAUD_INIT, (void *)&InputMsg, (void *)&OutputMsg);
```

Copyright 2003 Drew Technologies, Inc.

FAST_INIT

This function is ISO14230-4(KWP2000) specific. This function causes the PassThru device to initiate a fast initialization sequence with the target ECU. The ECUs on the ISO9141 bus require initialization in order to communicate with the PassThru device. The fast initialization sequence will use 10.4K-baud rate. The PassThru device transmits a Wake Up pattern on the serial bus. After the Wake Up time interval expires the PassThru device transmits the StartCommunicationRequest message. The ECU answers back with the StartCommunicationResponse message with its supported transmission mode parameters defined by the two key bytes. The PassThru device returns the 2-byte KeyWord parameters to the UserApplication. The UserApplication must decode the communication identifier and configure the PassThru device transmission parameters(via PassThruIoctl) so the PassThru device can correctly send and receive messages over the vehicle network.

This function must be called after the PassThruConnect function and the UserApplication must configure the ECU specified transmission parameters or else the PassThru device will not be able transfer messages to/from the vehicle network.

The code sample below demonstrates how to setup the FAST_INIT function parameters. On successful completion the OutputMsg will contain the target ECU's StartCommunicationPositive Response message. The key byte identifiers are located in the data field of the StartCommunicationPositive Response message. Refer to PASSTHRU_MSG structure definition on page 46.

```
typedef struct {
   unsigned long ProtocolID;        /* vehicle network protocol */
   unsigned long RxStatus;          /* receive message status */
   unsigned long TxFlags;           /* transmit message flags */
   unsigned long Timestamp;         /* receive message timestamp(in microseconds) */
   unsigned long DataSize;          /* byte size of message payload in the Data array */
   unsigned long ExtraDataIndex;    /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];        /* message payload or data */
} PASSTHRU_MSG;

unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */
PASSTHRU_MSG InputMsg;
PASSTHRU_MSG OutputMsg;

status = PassThruConnect(ISO14230, 0x00000000, &ChannelID);

memset(&InputMsg, 0, sizeof(InputMsg));

InputMsg.ProtocolID = ISO14230;
InputMsg.Data[0] = 0xC1; /* Format(functional addressing, 1 byte payload), first Header byte */
InputMsg.Data[1] = 0x33;  /* Initialization address used to activate all ECUs, second Header byte */
InputMsg.Data[2] = 0xF1; /* Scan Tool physical source address, third Header byte */
InputMsg.Data[3] = 0x81;  /* Start Communication Request Service Identifier, first Data byte */
InputMsg.DataSize = 4;      /* Input message contains 4 bytes */

memset(&OutputMsg, 0, sizeof(OutputMsg));

status = PassThruIoctl(ChannelID, FAST_INIT, (void *)&InputMsg, (void *)&OutputMsg);
```

Copyright 2003 Drew Technologies, Inc.

CLEAR_TX_BUFFER

This function causes the PassThru device to empty its transmit message queue. The code sample below demonstrates how to setup the CLEAR_TX_BUFFER function parameters. On successful completion the PassThru device's transmit message queue will contain no messages.

```
unsigned long status;
unsigned long ChannelID;        /* Logical channel identifier returned by PassThruConnect */

status = PassThruConnect(CAN, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, CLEAR_TX_BUFFER, (void *)NULL, (void *)NULL);
```

CLEAR_RX_BUFFER

This function causes the PassThru device to empty its receive message queue. The code sample below demonstrates how to setup the CLEAR_RX_BUFFER function parameters. On successful completion the PassThru device's receive message queue will contain no messages.

```
unsigned long status;
unsigned long ChannelID;        /* Logical channel identifier returned by PassThruConnect */

status = PassThruConnect(CAN, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, CLEAR_RX_BUFFER, (void *)NULL, (void *)NULL);
```

CLEAR_PERIODIC_MSGS

This function causes the PassThru device to empty its periodic message table for the associated network protocol. The code sample below demonstrates how to setup the CLEAR_PERIODIC_MSGS function parameters. On successful completion the PassThru device's periodic message table will contain no periodic messages.

```
unsigned long status;
unsigned long ChannelID;        /* Logical channel identifier returned by PassThruConnect */

status = PassThruConnect(CAN, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, CLEAR_PERIODIC_MSGS, (void *)NULL, (void *)NULL);
```

CLEAR_MSG_FILTERS

This function causes the PassThru device to empty its message filter table for the associated network protocol. The code sample below demonstrates how to setup the CLEAR_MSG_FILTER function parameters. On successful completion the PassThru device's message filter table will contain no filters.

```
unsigned long status;
unsigned long ChannelID;        /* Logical channel identifier returned by PassThruConnect */

status = PassThruConnect(CAN, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, CLEAR_MSG_FILTERS, (void *)NULL, (void *)NULL);
```

Copyright 2003 Drew Technologies, Inc.

CLEAR_FUNCT_MSG_LOOKUP_TABLE

This function is J1850PWM specific. This function removes all functional addresses from the PassThru device's J1850PWM transceiver look-up table. The functional address look-up table is a register block within the J1850PWM transceiver where the specified functional addresses are written.

The code sample below demonstrates how to setup the CLEAR_FUNCT_MSG_LOOKUP_TABLE function parameters. On successful completion the PassThru device's J1850PWM transceiver's look-up table will contain no functional addresses.

```
unsigned long status;
unsigned long ChannelID;            /* Logical channel identifier returned by PassThruConnect */

status = PassThruConnect(J1850PWM, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, CLEAR_FUNCT_MSG_LOOKUP_TABLE, (void *)NULL,
                       (void *)NULL);
```

ADD_TO_FUNCT_MSG_LOOKUP_TABLE

This function is J1850PWM specific. This function adds functional addresses to the PassThru device's J1850PWM transceiver look-up table. The functional address look-up table contains 31 emtries and is a register block within the J1850PWM transceiver where the specified functional addresses are written. The J1850PWM transceiver uses the look-up table functional addresses for filtering Network messages on the vehicle LAN bus. If the target functional address in the message header matches a functional address in the look-up table then the receive message will enter the transceiver receive register. Otherwise the Network message will be ignored. This look-up table must be initialized in order for the PassThru device to receive any J1850 protocol frames from the vehicle network.

As a related initialization issue, the physical source address of the PassThru device must also be written to the physical address register in the J1850PWM transceiver. This is accomplished by the PassThruIoctl SET_CONFIG command and defining the NODE_ADDRESS parameter.

The code sample below demonstrates how to setup the ADD_TO_FUNCT_MSG_LOOKUP_TABLE function parameters. On successful completion the functional address look-up table will be updated with the User Application specified functional addresses.

```
typedef struct {
   unsigned long NumOfBytes;     /* Number of functional addresses in array */
   unsigned char *BytePtr;       /* pointer to functional address array */
} SBYTE_ARRAY;

SBYTE_ARRAY InputMsg;
unsigned char FuncAddr[4];       /* Functional address array – address values defined in J2178-4 */

unsigned long status;
unsigned long ChannelID;         /* Logical channel identifier returned by PassThruConnect */

FuncAddr[0] = 0x0A;      /* Engine Air Intake functional address. */
FuncAddr[1] = 0x12;      /* Throttle functional address. */
FuncAddr[2] = 0x1A;      /* Engine RPM functional address. */
FuncAddr[3] = 0x32;      /* Brakes functional address. */

InputMsg.NumOfBytes = 4;              /* Functional address array contains four addresses. */
InputMsg.BytePtr = &FuncAddr[0];      /* Assign pointer to functional address array. */
```

Copyright 2003 Drew Technologies, Inc.

```
status = PassThruConnect(J1850PWM, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, ADD_TO_FUNCT_MSG_LOOKUP_TABLE, (void *)&InputMsg,
                       (void *)NULL);
```

Copyright 2003 Drew Technologies, Inc.

DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE

This function is J1850PWM specific. This function removes functional addresses from the PassThru device's J1850PWM transceiver look-up table. As a result, Network messages containing the "removed" address will not enter the J1850PWM transceiver receive register.

The code sample below demonstrates how to setup the DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE function parameters. On successful completion the functional address look-up table will be updated and the User Application specified functional addresses will be removed.

```
typedef struct {
   unsigned long NumOfBytes;     /* Number of functional addresses in array */
   unsigned char *BytePtr;       /* Pointer to functional address array */
} SBYTE_ARRAY;

SBYTE_ARRAY InputMsg;
unsigned char FuncAddr[2];       /* Functional address array – address values defined in J2178-4 */
unsigned long status;
unsigned long ChannelID;         /* Logical channel identifier returned by PassThruConnect */

FuncAddr[0] = 0x12;      /* Throttle functional address. */
FuncAddr[1] = 0x1A;      /* Engine RPM functional address. */

InputMsg.NumOfBytes = 2;                 /* Functional address array contains four addresses. */
InputMsg.BytePtr = &FuncAddr[0];         /* Assign pointer to functional address array. */

status = PassThruConnect(J1850PWM, 0x00000000, &ChannelID);

status = PassThruIoctl(ChannelID, DELETE_FROM_FUNCT_MSG_LOOKUP_TABLE,
                       (void *)&InputMsg, (void *)NULL);
```

Copyright 2003 Drew Technologies, Inc.

PassThru API Error Codes

All the error codes returned by J2534 API are listed below.

API Error Label/Name	Error Value	API Error Description
STATUS_NOERROR ……..	0x00	Function completed successfully.
ERR_NOT_SUPPORTED ……...	0x01	Function option is not supported.
ERR_INVALID_CHANNEL_ID ……	0x02	Channel Identifier or handle is not recognized.
ERR_INVALID_PROTOCOL_ID ……..	0x03	Protocol Identifier is not recognized.
ERR_NULL_PARAMETER …….	0x04	NULL pointer presented as a function parameter, NULL is an invalid address.
ERR_INVALID_IOCTL_VALUE ……..	0x05	Ioctl GET_CONFIG/SET_CONFIG parameter value is not recognized.
ERR_INVALID_FLAGS ……..	0x06	Flags bit field(s) contain(s) an invalid value.
ERR_FAILED ……..	0x07	Unspecified error, use PassThruGetLastError for obtaining error text string.
ERR_DEVICE_NOT_CONNECTED …….	0x08	PassThru device is not connected to the PC.
ERR_TIMEOUT ……..	0x09	Timeout violation. PassThru device is unable to read specified number of messages from the vehicle network. The actual number of messages returned is in NumMsgs.
ERR_INVALID_MSG ……..	0x0A	Message contained a min/max length, ExtraData support or J1850PWM specific source address conflict violation.
ERR_INVALID_TIME_INTERVAL ……..	0x0B	The time interval value is outside the specified range.
ERR_EXCEEDED_LIMIT ……..	0x0C	The limit(ten) of filter/periodic messages has been exceeded for the protocol associated the communications channel.
ERR_INVALID_MSG_ID ……...	0x0D	The message identifier or handle is not recognized.
ERR_INVALID_ERROR_ID ……..	0x0E	Not returned by any of the API functions.
ERR_INVALID_IOCTL_ID ……	0x0F	Ioctl identifier is not recognized.
ERR_BUFFER_EMPTY ……..	0x10	PassThru device could not read any messages from the vehicle network.
ERR_BUFFER_FULL ……..	0x11	PassThru device could not queue any more transmit messages destined for the vehicle network.
ERR_BUFFER_OVERFLOW ……..	0x12	PassThru device experienced a buffer overflow and receive messages were lost.
ERR_PIN_INVALID ……...	0x13	Unknown pin number specified for the J1962 connector.
ERR_CHANNEL_IN_USE ……...	0x14	An existing communications channel is currently using the specified network protocol.
ERR_MSG_PROTOCOL_ID ……..	0x15	The specified protocol type within the message structure is different from the protocol associated with the communications channel when it was opened.
RESERVED ……..	0x16 - Unused 0xFFFFFFFF	

Copyright 2003 Drew Technologies, Inc.

PassThru Message Structure

The PassThru message structure(PASSTHRU_MSG) is used for all User Application transmit and receive messages that flow to and from the vehicle network. The DataSize is the sum of all the bytes contained in the Data array, including the ExtraData. The ExtraDataIndex is the start location of the ExtraData within the Data array. The ExtraData is protocol specific and may be In-FrameResponse(IFR), CRC or checksum data. The ExtraData is only applicable for receive messages and is not used for transmit messages.

```
typedef struct {
   unsigned long ProtocolID;
   unsigned long RxStatus;
   unsigned long TxFlags;
   unsigned long Timestamp;
   unsigned long DataSize;
   unsigned long ExtraDataIndex;
   unsigned char Data[4128];
} PASSTHRU_MSG;
```

ProtocolID

The protocol identifier selects the network layer protocol that is used for the communications channel. It must match the protocol identifier that was used on the PassThruConnect function call. The UserApplication sets this field for transmit frames and the PassThru API/DLL sets this field for receive frames.

RxStatus

The PassThru device sets this field for all frames flowing from the vehicle network. The receive status value must adhere to the bit field values listed below.

RxStatus Definition	RxStatus Bit Field	RxStatus Description	RxStatus Value
Tool Vendor extension ……	31-24 …	Tool Vendor defined	Tool Vendor specific
Reserved ……..	23-9 …..	Unused	Must be set to zero
CAN_29BIT_ID ……..	8 ……....	CAN ID Type	0 = 11-bit 1 = 29-bit
Reserved ……..	7-3 ……	Unused	Must be set to zero
RX_BREAK ……	2 ……....	Receive Break	0 = No Break indication 1 = Break indication present
ISO15765_FIRST_FRAME	1 ……	ISO15765-2 only	0 = No First Frame indication 1 = First Frame indication
TX_MSG_TYPE ……..	0 ……	Receive Indication / Transmit Confirmation	0 = Rx Frame indication 1 = Tx Frame confirmation

CAN_29BIT_ID

This option is used to denote whether the received CAN frame is a Standard Data Frame(page 65) or an Extended Data Frame(page 66). The Standard Data Frame contains an 11-bit Identifier whereas the Extended Data Frame contains a 29-bit Identifier.

RX_BREAK

A break symbol appeared on the bus which terminated any in-progress receive operations for all nodes connected to the bus. The associated frame was partially assembled when the bus transfer was interrupted. The receive break indication is only supported for SCI_A_XXXX, SCI_B_XXXX and J1850VPW protocols.

Copyright 2003 Drew Technologies, Inc.

ISO15765_FIRST_FRAME

This indication is delivered when the PassThru device receives the first frame of a multi-segmented transfer. The PASSTHRU_MSG structure presented to the UserApplication will have the ExtraDataIndex set to zero and the Data array will contain the CAN header address information. The DataSize will be set to either four(4-byte CAN Identifier only) or five(4-byte CAN Identifier plus Extended Address byte).

The case where the UserApplication receives a first frame indication but the complete frame is never delivered is currently not specified by the PassThru API. Therefore the UserApplication must use a timeout mechanism to handle receive frames that are signaled as being assembled but later aborted by the PassThru device's CAN controller.

TX_MSG_TYPE

For a normal receive frame this bit will be set to zero and the DataSize will be set to the size of data contained in the Data array. For a frame received as a result of the LOOPBACK option(refer to PassThruIoctl section) being enabled this bit will be set to one and the DataSize will be set to the size of data contained in the Data array. The transmit confirmation implies that the PassThru device successfully transmitted the UserApplication frame onto the vehicle network.

For ISO15765 a transmit confirmation indication is provided by the PassThru device for each UserApplication frame that is successfully transmitted. The UserApplication will receive(via PassThruReadMsgs) a PASSTHRU_MSG frame with the RxStatus TX_MSG_TYPE bit set to one, the ExtraDataIndex value set to zero(the original transmit frame is not returned) and the Data array will contain the CAN header address information. The DataSize will be set to either four(4-byte CAN Identifier only) or five(4-byte CAN Identifier plus Extended Address byte).

If the PassThru device encounters a CAN network problem which causes a transmit failure then the UserApplication is never notified about the transmit failure. The UserApplication must use a timeout mechanism and the absence of an expected ECU response to determine that a transmit frame has been aborted.

TxFlags

The UserApplication sets this field for all frames flowing to the vehicle network. The transmit status value must adhere to the bit field values listed below.

TxFlags Definition	TxFlags Bit Field	TxFlags Description	TxFlags Value
Tool Vendor extension ……	31-24 ….	Tool Vendor defined	Tool Vendor specific
SCI_TX_VOLTAGE ……..	23 ……..	SCI programming	0 = do not apply voltage after transmitting message 1 = apply voltage(20V) after transmitting message
Reserved ……..	22-17 …	Unused	Must be set to zero
BLOCKING ……	16 ……..	Tx blocking mode	0 = non-blocking Transmit request 1 = blocking Transmit request
Reserved ……..	15-9 …..	Unused	Must be set to zero
CAN_29BIT_ID ……..	8 ……	CAN ID Type	0 = 11-bit 1 = 29-bit
ISO15765_ADDR_TYPE …	7 ……..	ISO15765-2 Addressing mode	0 = No extended address 1 = Extended address is first byte following the CAN ID
ISO15765_FRAME_PAD …	6 ……..	ISO15765-2 Frame Pad mode	0 = No frame padding 1 = Zero pad FlowControl, Single and Last ConsecutiveFrame to full

Copyright 2003 Drew Technologies, Inc.

CAN frame size.

Reserved …… 5-0 ….. Unused Must be set to zero

BLOCKING

With blocking enabled the API/DLL waits until the PassThru device sends either a Transmit Confirmation or Error Indication back before returning to the UserApplication. The Transmit Confirmation implies that the UserApplication message was successfully transmitted onto the vehicle network.

CAN_29BIT_ID

This option is used to select between CAN Standard Data Frame(page 65) and Extended Data Frame(page 66). The Standard Data Frame contains a 12-bit Arbitration Field that is comprised of an 11-bit Identifier and a Remote Transmission Request(RTR) bit. The Extended Data Frame contains a 32-bit Arbitration Field where the first 11-bits are the most significant Identifier bits followed by Substitute Remote Request(SRR) bit, Identifier Extension(IDE) bit, 18-bit least significant Identifier bits and a Remote Transmission Request(RTR) bit. The CAN Identifier within the Extended Data Frame is comprised of 29-bits.

ISO15765_ADDR_TYPE

This option is used to extend the available address range for large networks. It is used to encode both the transmitting and receiving peers that are located on a subnetwork connected to the main vehicle network. The extended address byte is part of the CAN header and immediately follows the CAN Identifier.

In case of a ISO15765_ADDR_TYPE conflict between the PASSTHRU_MSG *TxFlags* and the PassThruConnect *Flags* settings, the PASSTHRU_MSG *TxFlags* option will take precedence.

ISO15765_FRAME_PAD

This option, when enabled, instructs the PassThru device to zero fill all CAN flow control frames. The flow control frame types are FirstFrame, ConsecutiveFrame and FlowControl.

Timestamp

Receive message timestamp in microseconds. The PassThru device provides the value for this field. For normal receive frames it is the time that the first byte of the ECU message was received by the PassThru device. For LOOPBACK frames it is the time that the last byte is transmitted by the PassThru device.
The timestamp is not used with UserApplication generated transmit messages.

DataSize

Byte size of the message payload including the “extra data” (referenced by the ExtraDataIndex) in the Data array. The message payload is a complete protocol frame(header, protocol information, data and error detection bytes) transmitted by an ECU.

The UserApplication sets this field for transmit frames and the PassThru API/DLL sets this field for receive frames.

ExtraDataIndex

This field is the starting index of the “extra data”(i.e. CRC, IFR, checksum) in the Data array. The PassThru API/DLL sets this field for receive frames and the User Application must set it to zero for transmit frames.

If the Data array contains no extra data then the ExtraDataIndex will be the same as the DataSize value. If the Data array contains extra data then the ExtraDataIndex will be less than the DataSize value.

Copyright 2003 Drew Technologies, Inc.

The ExtraDataIndex is zero when the payload data is an ISO15765 Transmit indication.

The extra data is concatenated to the payload data and the ExtraDataIndex is zero based. The ExtraDataIndex is not used with transmit messages.

Data

The payload or User Application specific data.

The UserApplication sets this field for transmit frames and the PassThru API/DLL sets this field for receive frames.

The UserApplication does not provide the CRC or checksum value for transmit frames. The PassThru device transceiver generates and appends the appropriate error detection byte(s) to the UserApplication transmit frame.

The protocols returning ExtraData bytes(CRC, IFR, checksum, etc) are J1850PWM, J1850VPW, ISO9141 and ISO14230.

Copyright 2003 Drew Technologies, Inc.

Message Data Format

The Transmit and Receive frame layout(from the UserApplication perspective) for each network protocol is depicted below.

CAN

Bit 31 = 0	Bit 30 = 0	Bit 29 = 0	CAN ID Bits 28-24	CAN ID Bits 23-16	CAN ID Bits 15-8	CAN ID Bits 7-0	Data
			Byte 0	Byte 1	Byte 2	Byte 3	Byte(4) ... Byte(11)

Figure 2 Maximum size CAN Receive/Transmit frame with 29-bit Identifier

For a maximum size receive frame the ExtraDataIndex is 12 and the DataSize is 12.

CAN ID Bits 31-24	CAN ID Bits 23-16	Bit 15 = 0	Bit 14 = 0	Bit 13 = 0	Bit 12 = 0	Bit 11 = 0	CAN ID Bits 10-8	CAN ID Bits 7-0	Data
Byte 0 = 0	Byte 1 = 0						Byte 2	Byte 3	Byte(4) ... Byte(11)

Figure 3 Maximum size CAN Receive/Transmit frame with 11-bit Identifier

For a maximum size receive frame the ExtraDataIndex is 12 and the DataSize is 12.

ISO15765-4

The PCI byte, which follows the CAN Identifier or extended address(if configured), is added by the PassThru device on transmit frames and removed by the PassThru device on receive frames. The UserApplication does not have to account for the PCI byte in either transmit or receive frames.

Bit 31 = 0	Bit 30 = 0	Bit 29 = 0	CAN ID Bits 28-24	CAN ID Bits 23-16	CAN ID Bits 15-8	CAN ID Bits 7-0	Extended Address	Data
			Byte 0	Byte 1	Byte 2	Byte 3	Byte 4	Byte(5) … Byte(4,099)

Figure 4 Max. size ISO 15765-4 Receive/Transmit frame with 29-bit Identifier and extended address

For a maximum size receive frame the ExtraDataIndex is 4,101 and the DataSize is 4,101.

Bit 31 = 0	Bit 30 = 0	Bit 29 = 0	CAN ID Bits 28-24	CAN ID Bits 23-16	CAN ID Bits 15-8	CAN ID Bits 7-0	Data
			Byte 0	Byte 1	Byte 2	Byte 3	Byte(4) ... Byte(4,099)

Figure 5 Maximum size ISO 15765-4 Receive/Transmit frame with 29-bit Identifier

For a maximum size receive frame the ExtraDataIndex is 4,101 and the DataSize is 4,101.

Copyright 2003 Drew Technologies, Inc.

CAN ID Bits 31-24 Byte 0 = 0	CAN ID Bits 23-16 Byte 1 = 0	Bit 15 = 0	Bit 14 = 0	Bit 13 = 0	Bit 12 = 0	Bit 11 = 0	CAN ID Bits 10-8 Byte 2	CAN ID Bits 7-0 Byte 3	Extended Address Byte 4	Data Byte(5) … Byte(4,099)

Figure 6 Max. size ISO 15765-4 Receive/Transmit frame with 11-bit Identifier and extended address

For a maximum size receive frame the ExtraDataIndex is 4,101 and the DataSize is 4,101.

CAN ID Bits 31-24 Byte 0 = 0	CAN ID Bits 23-16 Byte 1 = 0	Bit 15 = 0	Bit 14 = 0	Bit 13 = 0	Bit 12 = 0	Bit 11 = 0	CAN ID Bits 10-8 Byte 2	CAN ID Bits 7-0 Byte 3	Data Byte(4) ... Byte(4,099)

Figure 7 Maximum size ISO 15765-4 Receive/Transmit frame with 11-bit Identifier

For a maximum size receive frame the ExtraDataIndex is 4,101 and the DataSize is 4,101.

J1850PWM

Header Byte 0	Header Byte 1	Header Byte 2	Data Byte(4) … Byte(9)

Figure 8 Maximum size J1850PWM Transmit frame

The PassThru device adds a one byte CRC to the UserApplication message. The UserApplication is not responsible for calculating and appending a CRC byte to its message.

Header Byte 0	Header Byte 1	Header Byte 2	Data Byte(4) … Byte(9)	CRC Byte 10

Figure 9 Maximum size J1850PWM Receive frame with CRC

For the case of a maximum size receive frame with a single CRC byte the ExtraDataIndex is 10 and the DataSize is 11.

Header Byte 0	Header Byte 1	Header Byte 2	Data Byte(4) … Byte(9)	In Frame Response(IFR) Byte 10	CRC Byte 11

Figure 10 Maximum size J1850PWM Receive frame with IFR and CRC

For the case of a maximum size receive frame with a IFR and CRC bytes the ExtraDataIndex is 10 and the DataSize is 12.

J1850VPW

Data Byte(0)	Data Byte(1) … Byte(4,125)	Data Byte(4,126)

Copyright 2003 Drew Technologies, Inc.

Figure 11 Maximum size J1850VPW Transmit frame

The PassThru device adds a one byte CRC to the UserApplication message. The UserApplication is not responsible for calculating and appending a CRC byte to its message.

Data	Data	CRC
Byte(0)	Byte(1) … Byte(4,126)	Byte(4,127)

Figure 12 Maximum size J1850VPW Receive frame with CRC

For the case of a maximum size receive frame with a single CRC byte the ExtraDataIndex is 4,127 and the DataSize is 4,128.

Data	Data	In Frame Response(IFR)	CRC
Byte(0)	Byte(1) … Byte(4,125)	Byte(4,126)	Byte(4,127)

Figure 13 Maximum size J1850VPW Receive frame with IFR and CRC

For the case of a maximum size receive frame with a IFR and CRC bytes the ExtraDataIndex is 4,126 and the DataSize is 4,128.

ISO9141

Data	Data	Data
Byte(0)	Byte(1) … Byte(4126)	Byte(4126)

Figure 14 Maximum size ISO 9141 Transmit frame

The PassThru device adds a one byte checksum to the UserApplication message. The UserApplication is not responsible for calculating and appending a checksum byte to its message.

Data	Data	Checksum
Byte(0)	Byte(1) … Byte(4126)	Byte(4127)

Figure 15 Maximum size ISO 9141 Receive frame with Checksum

The receive frame will contain extra data as a single checksum byte.
For the case of a maximum size receive frame with a single checksum byte the ExtraDataIndex is 260 and the DataSize is 261.

ISO14230-4

Header	Header	Header	Header	Data
Byte 0	Byte 1	Byte 2	Byte 3	Byte(4) … Byte(258)

Figure 16 Maximum size ISO 14230-4 Transmit frame

The maximum size UserApplication transmit frame has a 4 byte header and 256 bytes of data.
The PassThru device adds a one byte checksum to the UserApplication message. The UserApplication is not responsible for calculating and appending a checksum byte to its message.

Copyright 2003 Drew Technologies, Inc.

Header Byte 0	Header Byte 1	Header Byte 2	Header Byte 3	Data Byte(4) … Byte(258)	Checksum Byte(259)

Figure 17 Maximum size ISO 14230-4 Receive Frame with checksum

The receive frame will contain extra data as a single checksum byte.
For the case of a maximum size receive frame with a single checksum byte the ExtraDataIndex is 260 and the DataSize is 261.

Message Data Validity

The J2534 API by design does not verify content correctness of receive and transmit messages that it handles. The J2534 API uses the ERR_INVALID_MSG error code to reject UserApplication messages that violate minimum/maximum protocol length requirements and/or incorrect use of ExtraData option. The protocol specific frame length restrictions and receive message ExtraData support is highlighted in Table 1.

Protocol	Min. Message Size	Max. Message Size	ExtraData Support	ExtraData Description
CAN	4	12	No	Not applicable
ISO15765-4	4	4100	No	Not applicable
J1850PWM	3	12	Yes	In-Frame Response byte(s) CRC
J1850VPW	1	4128	Yes	In-Frame Response byte(s) CRC
ISO9141	1	4128	Yes	Checksum
ISO14230-4	4	260	Yes	Checksum
SCI	1	256	No	Not applicable

Table 1 Message Length Restrictions and ExtraData Support.

The ExtraData option is not used by any protocol for transmit messages that originate from the User Application.

For the J1850PWM protocol if the message's physical source address does not match the source address assigned to the PassThru device(via PassThruIoctl SET_CONFIG, NODE_ADDRESS) then the message is regarded as invalid. In this case the J2534 API will return the ERR_INVALID_MSG error code.

Copyright 2003 Drew Technologies, Inc.

PASSTHRU API INTERPRETATION ISSUES

The original J2534 specification contains a lot of useful information but from a practical standpoint omits details, inadequately explains functionality or is just plain vague. This section attempts to provide the missing details and clarify some of the ambiguities present in the original document

ISO9141

Contrary to the verbiage in the J2534 specification, the UserApplication cannot configure the number of data bits(i.e. 7-bit, 8-bit) through the generic API. The PassThru device default is presumably 8-bit data bits.

The PassThru device does not attempt to automatically detect(either through software or hardware) the vehicle network baud rate. The UserApplication must set the appropriate baud rate through PassThruIoctl(SET_CONFIG, BAUD_RATE) function.

The FIVE_BAUD_INIT function must be called after the PassThruConnect function and the UserApplication must configure the ECU specified transmission parameters or else the PassThru device will not be able transfer messages to/from the vehicle network.

ISO14230

The J2534 specification left undefined whether damaged ISO14230 frames should be passed to the UserApplication or discarded by the PassThru device. The consensus is that the PassThru device discards ISO14230 frames with a bad checksum.

The FIVE_BAUD_INIT or FAST_INIT function must be called after the PassThruConnect function and the UserApplication must configure the ECU specified transmission parameters or else the PassThru device will not be able transfer messages to/from the vehicle network.

CAN

CAN frames returned by the PassThru device do not contain all the bits comprising the Arbitration Field but just the Identifier field. The 11-bit CAN Identifier is returned as a 4-byte data object occupying the LSB portion. The 29-bit CAN Identifier by necessity is returned as a 4-byte data object occupying the LSB portion.

ISO15765

ISO15765 frames can’t be received or transmitted by the PassThru device until the UserApplication sets up a flow control filter.

A transmit confirmation indication is provided by the PassThru device for each UserApplication frame that is successfully transmitted. When LOOPBACK is disabled(default) the RxStatus TX_MSG_TYPE bit is set to one, the ExtraDataIndex value is zero(the original transmit frame is not returned) and the Data array will contain the CAN header address information. When LOOPBACK is enabled(via PassThruIoctl) the RxStatus TX_MSG_TYPE bit is set to zero, the DataSize value is the size of the original transmit frame and the original transmit frame is returned. Also when LOOPBACK is enabled the transmit confirmation indication(conataining just the CAN header address information) is returned.

If the PassThru device encounters a CAN network problem which causes a transmit failure then the UserApplication is never notified about the transmit failure. The lack of an ISO15765 Error Indication for aborted transmit frames is a deficiency in the original J2534 specification. The UserApplication must use a timeout mechanism and the absence of an expected ECU response to determine that a transmit frame has

Copyright 2003 Drew Technologies, Inc.

been aborted.

The J2534 specification does not prohibit applying a PASS or BLOCK filter to the ISO15765 frames, which remain after applying the FLOW_CONTROL_FILTER. However not all PassThru device implementations support this feature. Some PassThru devices may prohibit a UserApplication from configuring PASS or BLOCK filters for ISO15765.

The PCI byte is added to the User Applications transmit frames by the PassThru device. The PCI byte is stripped from the receive frames by the J2534 API. The User Application does not have to account for PCI bytes in the ISO15765 data traffic.

J1850PWM

The functional address look-up table must be initialized(ADD_TO_FUNCT_MSG_LOOKUP_TABLE) after PassThruConnect completes or else the PassThru device will block all J1850PWM receive frames.

The J2534 specification left undefined whether errored J1850PWM frames should be passed to the UserApplication or discarded by the PassThru device. The consensus is that the PassThru device discards J1850PWM frames with a bad CRC.

J1850VPW

The J2534 specification lists different sizes for the maximum size J1850VPW frame in different parts of the document. The PassThru requirement section lists a 4K(4,096) byte maximum transfer size and the message data format section lists 4,128 bytes as the maximum frame size. The consensus is that the maximum size of the J1850VPW frame is 4,128 bytes.

The J2534 specification left undefined whether errored J1850VPW frames should be passed to the UserApplication or discarded by the PassThru device. The consensus is that the PassThru device discards J1850VPW frames with a bad CRC.

PassThruConnect

Why the CAN_29BIT_ID occurs as both a *Flags* option and a *TxFlags* option(PASSTHRU_MSG structure) is not explained(i.e. what problem does it solve). Also what happens when there is a conflict between the *Flags* and *TxFlags* CAN_29BIT_ID option value is not explained. The safest solution is to always set the appropriate CAN Identifier size(via *TxFlags* CAN_29BIT_ID) for all UserApplication generated messages.

PassThruReadMsgs

Default filter policy

The PassThru device default filter policy was not stated in the original document. If the default receive message filter policy is BLOCK, then invoking PassThruReadMsgs without setting at least one filter is meaningless as no messages will ever be returned. If the default receive message filter policy is PASS, then invoking PassThruReadMsgs will return all queued messages. On a busy vehicle network the flood of messages may obscure frames of interest. For either case setting a message filter is a necessary strategy.

Most PassThru tool vendors implemented a PASS default filter policy.

Transmit Confirm

The transmit confirm indication only occurs when LOOPBACK mode is enabled for the supported protocol(except ISO15765). LOOPBACK mode is enabled by calling PassThruIoctl with SET_CONFIG

Copyright 2003 Drew Technologies, Inc.

command and with the LOOPBACK option enabled. Along with the transmit indication the original frame is also returned.

It is assumed that the order of messages returned to the UserApplication is the same order that the messages entered the PassThru device transceiver.

PassThruWriteMsgs

Late Transmit Indication - Race condition

There is an API/DLL versus PassThru device race condition when the UserApplication makes a transmit request with a nonzero timeout value. The API/DLL transmit timer may expire after the PassThru device transmits the frame but before its transmit indication is delivered to the API/DLL. From the UserApplication's perspective the number of requested frames was not transmitted however from the PassThru device perspective all the requested frames were transmitted. As a result the UserApplication may encounter unexpected ECU response frames.

PassThruStartPeriodicMsg

Maximum Periodic Message Size

The maximum allowable length for a periodic message was not specified. The inclination is to assume 4,128 bytes (size of the Data array in PASSTHRU_MSG structure) but most tool vendors probably implemented a much smaller size(12 to 16 byte range). The supported protocols all specify a maximum size frame that range from 12 bytes(CAN) to 4,101bytes(ISO15765). For the sake of simplicity most tool vendors probably use a single maximum size across all protocols instead of a protocol specific maximum size.

Maximum Number of Periodic Messages

The maximum number of periodic messages is 10. It is not specified whether it is 10 total periodic messages or 10 per supported protocol.

Periodic Message versus Generic Message Precedence

If the PassThru device has a resource or timing issue between transmitting a periodic message and a generic UserApplication message then the periodic message will take precedence.

PassThruStopPeriodicMsg

Late Periodic Message - Race condition

If the PassThru device is commanded to stop a periodic message but the periodic message has already been attached to the transceiver Transmit Ring, then the periodic message will be transmitted. From the UserApplication perspective the PassThru device sends an extra periodic message which may result in an unexpected ECU response.

PassThruStartMsgFilter

Default filter policy

The PassThru device's default filter state for receive messages was not specified. The options are to either PASS all messages or BLOCK all messages. The default filter policy determines how the PassThruReadMsgs function must be used. With a default PASS policy the PassThruReadMsgs will be able to obtain any queued network messages. The PassThru device queues all receive messages if it adheres to a default PASS policy. With a default BLOCK policy a filter must be defined before the PassThruReadMsgs

will be able to obtain network messages meeting the filter criteria. The PassThru device does not queue any receive messages if it adheres to a default BLOCK policy. Depending on whether filtering was implemented in software or hardware(i.e. address Identifier registers), the PASS/BLOCK default policy may vary on a per protocol basis.

Maximum Mask and Pattern Size

The maximum allowable length for a mask or pattern message was not specified. The inclination is to assume 4,128 bytes (size of the Data array in PASSTHRU_MSG structure) but most tool vendors probably implemented a much smaller size(12 to 16 byte range). The supported protocols all specify a maximum size frame that range from 12 bytes(CAN) to 4,101bytes(ISO15765). For the sake of simplicity most tool vendors probably use a single maximum mask and pattern size across all protocols instead of a protocol specific maximum size.

Mask Bit Pattern Rule

A “1” mask bit implies examine the corresponding pattern bit. A “0” mask bit implies ignore the corresponding pattern bit.

Maximum and Minimum Flow Control message size

The minimum size of the ISO15765 Flow Control message is 4 bytes. The 11-bit CAN Identifier must be represented as a 4-byte object with the unused 2 bytes and 5 bits set to zero. The 29-bit CAN Identifier is also represented as a 4-byte object with the 3 unused bits set to zero.

The maximum size of the ISO15765 Flow Control message is 5 bytes. The fifth byte is the extended address byte.

Any Flow Control message that is not either 4-bytes(CAN Identifier only) or 5-bytes(CAN Identifier plus the extended address) in length is invalid. Some J2534 API/DLL implementations may not reject Flow Control messages that are either too short or too long.

Maximum Number of Filter Messages

The maximum number of filter messages is 10. It is not specified whether it is 10 total filter messages or 10 per supported protocol.

Filtering loopback frames

It was not specified whether frames that are looped back by the PassThru device are passed through the filter criteria. It was intended that loopback frames be used as transmit confirm indications and therefore be independent of any filter criteria. The interaction between loopback frames and the filter mechanism may vary between tool vendor implementations.

PassThruIoctl

Commands with superfluous ChannelID parameter

The READ_VBATT and READ_PROG_VOLTAGE commands are requests directed to the PassThru device and do not depend on an active communication channel. The ChannelID parameter is ignored by the API/DLL and should be set to zero by the UserApplication.

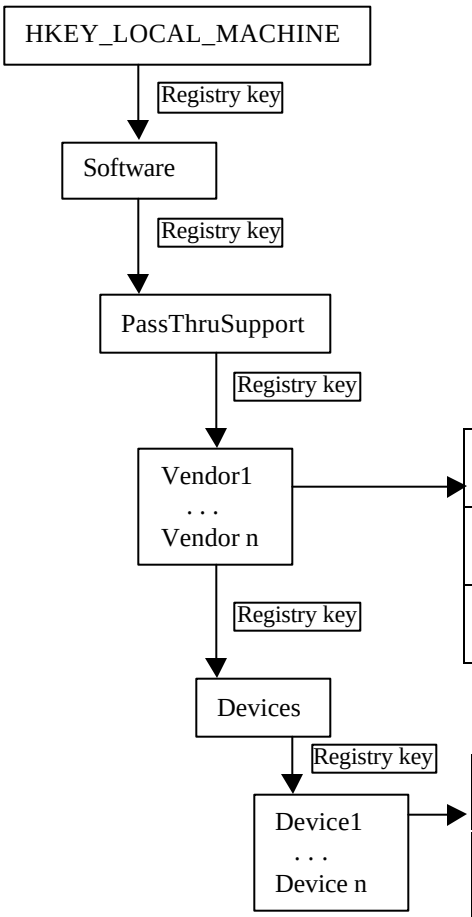
Copyright 2003 Drew Technologies, Inc.

Discovery of the J2534 DLL in the Windows® environment

The User Application is expected to parse the Windows® registry to discover which PassThru devices are installed and to determine their operational capabilities(i.e. supported vehicle network protocols). The registry is a system defined database in which applications and OS components store and retrieve configuration data. The User Application is expected to use the MS registry API to retrieve, modify or create registry data. In theory more than one PassThru device may be installed and the User Application may allow the operator to select and/or switch between PassThru devices. In practice there is typically only one installed PassThru device.

Copyright 2003 Drew Technologies, Inc.

The Windows® registry contains a section that is devoted to PassThru devices. This information is stored in the following hierarchy:



Name	Type	Data
Id	DWORD	0x00000000
Name	String	**Drew Technologies Inc.**

Name	Type	Data
DeviceIdExtra	**DWORD**	**0x00000000**
Name	**String**	**CarDAQ**
ProtocolSupported	**String**	**CAN,ISO9141, ISO14230, ISO15765, J1850VPW, J1850PWM**
ConfigApplication	**String**	**CARDAQ.EXE**
FunctionLibrary	**String**	**C:\WINDOWS\SYSTEM32\CARDAQ32.DLL**
Vendor Specific . . .	**Vendor Defined** . . .	**Vendor Specific Values** . . .

Finding the local path to the J2534 DLL

The following code sample is an example of using the Windows® registry API functions for obtaining the path to the vendor's PassThru J2534 DLL. The example does not look for multiple PassThru devices but only looks for the first PassThru device associated with the first vendor in the vendor's list. The DLL path string will be stored in the "DllLibraryPath" array.

Copyright 2003 Drew Technologies, Inc.

```
HKEY hKey1, hKey2, hKey3, hKey4, hKey5;
DWORD KeyType, KeySize;
char     DllLibraryPath[300];

If (RegOpenKeyEx(HKEY_LOCAL_MACHINE, “Software”, 0, KEY_READ, &hKey1) !=
   ERROR_SUCCESS)
{
   return;
}

If (RegOpenKeyEx(hKey1, “PassThruSupport”, 0, KEY_READ, &hKey2) != ERROR_SUCCESS)
{
   RegCloseKey(hKey1);
   return;
}

RegCloseKey(hKey1);

If (RegOpenKeyEx(hKey2, “Vendor1”, 0, KEY_READ, &hKey3) != ERROR_SUCCESS)
{
   RegCloseKey(hKey2);
   return;
}

RegCloseKey(hKey2);

If (RegOpenKeyEx(hKey3, “Devices”, 0, KEY_READ, &hKey4) != ERROR_SUCCESS)
{
   RegCloseKey(hKey3);
   return;
}

RegCloseKey(hKey3);

If (RegOpenKeyEx(hKey4, “Device1”, 0, KEY_READ, &hKey5) != ERROR_SUCCESS)
{
   RegCloseKey(hKey4);
   return;
}

RegCloseKey(hKey4);

/*
** Extract the Tool vendor’s J2534 API/DLL path from the registry.
*/
If (RegQueryValueEx(hKey5, “FunctionLibrary”, 0, &KeyType, DllLibraryPath, &KeySize) !=
   ERROR_SUCCESS)
{
   RegCloseKey(hKey5);
   return;
}

RegCloseKey(hKey5);
```

Attaching to the J2534 DLL and accessing the PassThru API

The User Application accesses J2534 DLL functions by using the Run Time Dynamic Linking feature, where the DLL is loaded and the DLL's PassThruXxxxx function addresses are explicitly extracted. In general the application will use the LoadLibrary function to map the DLL into the application's address space and the application will use the GetProcAddress function for obtaining all the J2534 API function addresses. It is expected that the User Application will use the FreeLibrary function to unmap the DLL as part of the application's termination process. The following code sample is an example of using the Win32® API functions, LoadLibrary and GetProcAddress. The path to the vendor's J2534 DLL was extracted from the Windows® registry and written to "DllLibraryPath" in the preceding code example.

```
/*
** Define all the PassThru function prototypes for the J2534 API.
*/
typedef long (WinAPI *PTCONNECT)(unsigned long, unsigned long, unsigned long *);
typedef long (WinAPI *PTDISCONNECT)(unsigned long);
typedef long (WinAPI *PTREADMSGS)(unsigned long, PASSTHRU_MSG *, unsigned long *,
        unsigned long);
typedef long (WinAPI *PTWRITEMSGS)(unsigned long, PASSTHRU_MSG *, unsigned long *,
        unsigned long);
typedef long (WinAPI *PTSTARTPERIODICMSG)(unsigned long, PASSTHRU_MSG *,
        unsigned long *, unsigned long);
typedef long (WinAPI *PTSTOPPERIODICMSG)(unsigned long, unsigned long);
typedef long (WinAPI *PTSTARTMSGFILTER)(unsigned long, unsigned long,
        PASSTHRU_MSG *,  PASSTHRU_MSG *, PASSTHRU_MSG *,
        unsigned long *, unsigned long);
typedef long (WinAPI *PTSTOPMSGFILTER)(unsigned long, unsigned long);
typedef long (WinAPI *PTSETPROGRAMMINGVOLTAGE)(unsigned long, unsigned long);
typedef long (WinAPI *PTREADVERSION)(char *, char *, char *);
typedef long (WinAPI *PTGETLASTERROR)(char *);
typedef long (WinAPI *PTIOCTL)(unsigned long, unsigned long, void *, void *);

/*
** Define permanent storage for all the PassThru function addresses.
*/
PTCONNECT PassThruConnect;
PTDISCONNECT PassThruDisconnect;
PTREADMSGS PassThruReadMsgs;
PTWRITEMSGS PassThruWriteMsgs;
PTSTARTPERIODICMSG PassThruStartPeriodicMsg;
PTSTOPPERIODICMSG PassThruStopPeriodicMsg;
PTSTARTMSGFILTER PassThruStartMsgFilter;
PTSTOPMSGFILTER PassThruStopMsgFilter;
PTSETPROGRAMMINGVOLTAGE PassThruSetProgrammingVoltage;
PTREADVERSION PassThruReadVersion;
PTGETLASTERROR PassThruGetLastError;
PTIOCTL PassThruIoctl;

/*
** Define permanent storage for the handle to the J2534 DLL library.
*/
HINSTANCE hDLL;

/*
** Attach to the J2534 DLL library.
*/
```

```
hDLL = LoadLibrary(DllLibraryPath);

/*
** Extract all the PassThru function addresses from the J2534 DLL.
*/
PassThruConnect = (PTCONNECT)(GetProcAddress(hDLL, “PassThruConnect”));
PassThruDisconnect = (PTDISCONNECT)(GetProcAddress(hDLL, “PassThruDisconnect”));
PassThruReadMsgs = (PTREADMSGS)(GetProcAddress(hDLL, “PassThruReadMsgs”));
PassThruWriteMsgs = (PTWRITEMSGS)(GetProcAddress(hDLL, “PassThruWriteMsgs”));
PassThruStartPeriodicMsg = (PTSTARTPERIODICMSG)(GetProcAddress(hDLL,
                                                “PassThruStartPeriodicMsg”));
PassThruStopPeriodicMsg = (PTSTOPPERIODICMSG)(GetProcAddress(hDLL,
                                                “PassThruStopPeriodicMsg”));
PassThruStartMsgFilter = (PTSTARTMSGFILTER)(GetProcAddress(hDLL,
                                                “PassThruStartMsgFilter”));
PassThruStopMsgFilter = (PTSTOPMSGFILTER)(GetProcAddress(hDLL,
                                                “PassThruStopMsgFilter”));
PassThruSetProgrammingVoltage = (PTSETPROGRAMMINGVOLTAGE)(GetProcAddress(
                                hDLL, “PassThruSetProgrammingVoltage”));
PassThruReadVersion = (PTREADVERSION)(GetProcAddress(hDLL, “PassThruReadVersion”));
PassThruGetLastError = (PTGETLASTERROR)(GetProcAddress(hDLL,
                                                “PassThruReadVersion));
PassThruIoctl = (PTIOCTL)(GetProcAddress(hDLL, “PassThruIoctl”));
```

Copyright 2003 Drew Technologies, Inc.

Declaring PassThru function prototypes

One confusing aspect is how to properly declare the DLL's J2534 public functions within the application. The J2534 API function declarations must be specified correctly or the application won't compile or link. The J2534 API function declarations must also account for how the API function names actually appear in the DLL. Many C++ compilers by default use the C calling convention which treat function names as case sensitive and always prefix an underscore to the name. To force the Standard Call convention, which eliminates the underscore prefix, the WINAPI keyword(defined in windef.h in the MinGW distribution) must be used in the function declaration. In addition the modifier, extern "C", is used to prevent the API function name from being mangled(it also disables function overloading). This allows the J2534 API function names to be seen and referenced exactly as they are defined instead of being encoded with a unique function signature. A generic J2534 API function prototype definition will look like the following:

```
typedef long (WINAPI *PTXXXXX)(argument list);
extern "C" PTXXXXX;
```

The module, which uses GetProcAddress to read the function address from the DLL, will define the following prototype:

```
PTXXXXX PassThruXxxxx;
```

Below is an example of how the User Application references the PassThru functions extracted from the J2534 DLL.

```
long status;
unsigned long ChannelID;

status = PassThruConnect(CAN, 0x00000000, &ChannelID);
```

Copyright 2003 Drew Technologies, Inc.

Flow Control Filter

ISO 15765-2 Background

ISO 15765-2 provides network layer services for a vehicle network comprised of a CAN based data link and physical layer. ISO 15765-2 was developed for on-board diagnostics but has been adopted for vehicle manufacturer enhanced diagnostics.

One important aspect is that it overcomes the raw CAN 8 byte data frame limitation and allows for message transfers of up to 4,095 bytes. Another important aspect is that it allows the receiving peer to regulate the rate of the transfer.

A discussion of the ISO 15765-2 flow control mechanism begins with a brief overview of CAN before progressing into the detailed flow control example using the J2534 flow control filter.

CAN Identifier

CAN messages do not contain the addresses of either the transmitting or receiving peer. The CAN Identifier is unique throughout the vehicle network and encodes the message content as well as the priority of the message(the lower the numerical value the, higher the priority). The numerical value of each CAN message identifier is assigned during the initial layout of the vehicle network.

All ECUs connected to the CAN bus will receive the CAN message and examine the Identifier. Based on preprogrammed acceptance test criteria, the ECU will either process or ignore the message.

CAN Identifier and Non-Destructive Bitwise Arbitration

To solve bus contention issues CAN uses Carrier Sense, Multiple Access with Collision Detect(CSMA/CD) combined with non-destructive bitwise arbitration to handle collision resolution.

The CAN Identifier with the lowest numerical value has the highest priority since the bitwise arbitration algorithm allows a dominant state(logic 0) to overwrite a recessive state(logic 1).

In bus arbitration conflicts(two ECUs transmitting simultaneously) the non-destructive arbitration mechanism guarantees that messages are sent in order of priority and that no messages are lost.

Copyright 2003 Drew Technologies, Inc.

CAN Frame Structure

The Standard CAN frame structure contains an Arbitration field(includes 11-bit Identifier), Control field(includes the Data Length Code) and Data field(up to eight bytes). The Standard CAN frame is depicted in Figure 18.

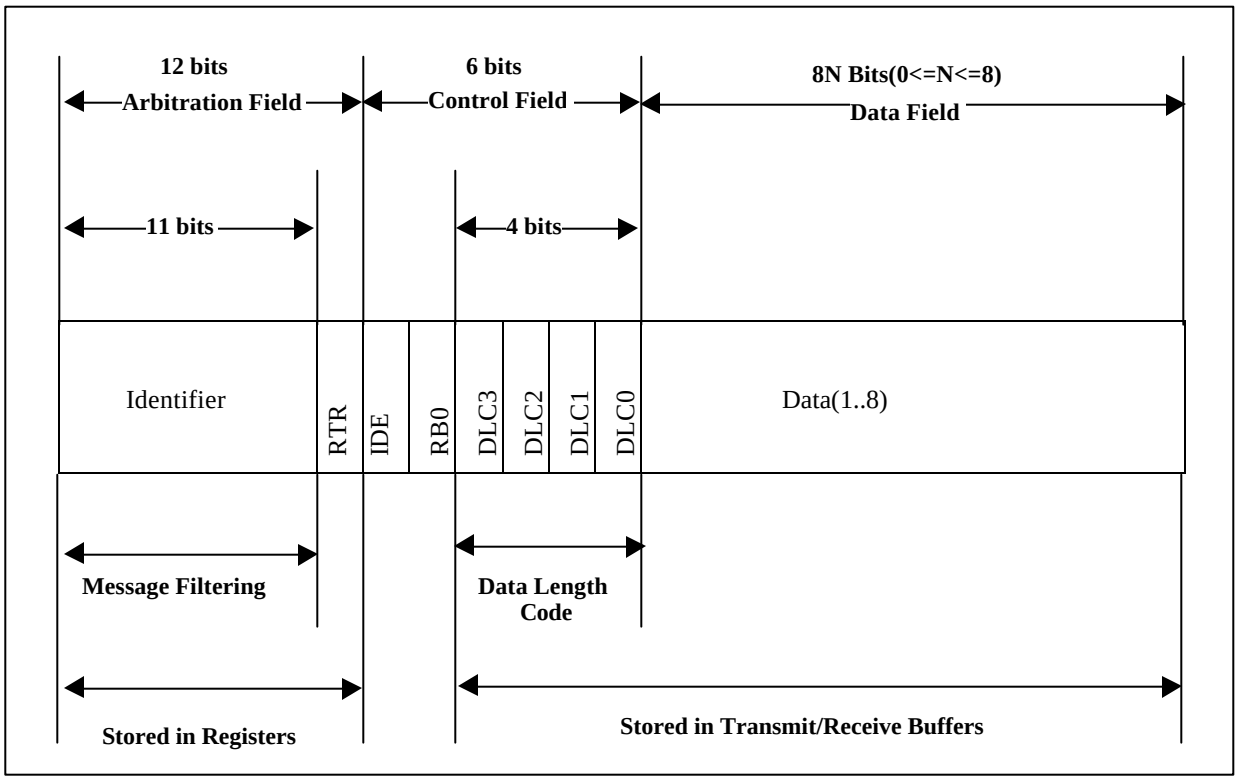


Figure 18 Standard CAN Data Frame, 11-bit Identifier

The 11-bit Identifier is a functional address that is also encoded for message priority.

The Remote Transmission Request(RTR) bit is used to distinguish a data frame from a remote request frame. The remote frame contains no data and asks the receiving node to send the data frame associated with the remote frame identifier.

The Identifier Extended(IDE) bit is used to distinguish a Standard(11-bit Identifier) data frame from an Extended(29-bit Identifier) data frame.

The Reserved Bit Zero(RB0) must be 0.

The Data Length Code(DLC3..DLC0) specifies the number of data bytes(8 maximum) contained in the message.

Copyright 2003 Drew Technologies, Inc.

The Extended CAN frame structure contains an Arbitration field(includes 29-bit Identifier), Control field(includes the Data Length Code) and Data field(up to eight bytes). The Extended CAN frame is depicted in Figure 19.

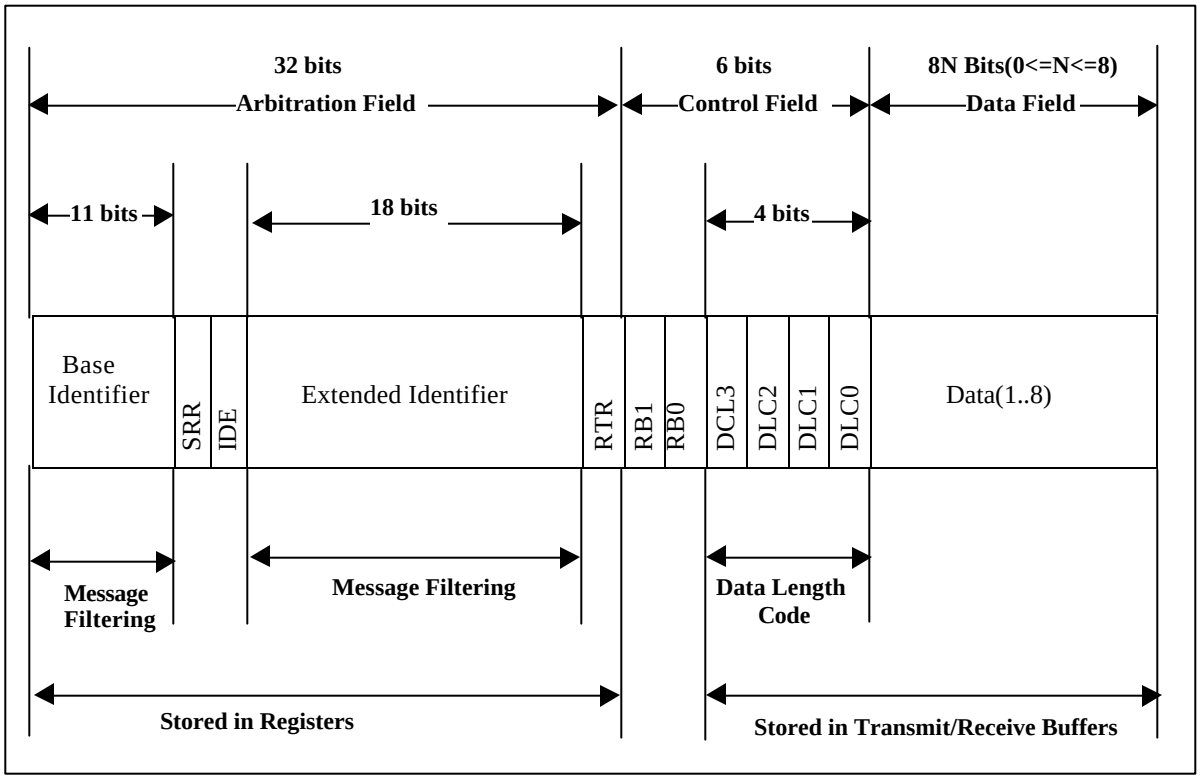


Figure 19 Extended CAN Data Frame, 29-bit Identifier

The 11-bit Base Identifier is a functional address that is also encoded for message priority.

The Substitute Remote Request(SRR) bit is always transmitted as a recessive bit so the bus arbitration will ensure that the Standard Data Frame will always have priority over the Extended Data Frame if both messages carry the same base identifier(11-bit Identifier).

The Identifier Extended(IDE) bit is used to distinguish a Standard(11-bit Identifier) data frame from an Extended(29-bit Identifier) data frame.

The 18-bit Extended Identifier is used to provide compatibility with other serial communications protocols used in the vehicle network. It contains the source address of the transmitting peer and the target address of the receiving peer.

The Remote Transmission Request(RTR) bit is used to distinguish a data frame from a remote request frame. The remote frame contains no data and asks the receiving node to send the data frame associated with the remote frame identifier.

The Reserved Bits(RB1,RB0) must be 0.

The Data Length Code(DLC3..DLC0) specifies the number of data bytes(8 maximum) contained in the message.

Copyright 2003 Drew Technologies, Inc.

ISO15765-2 Message Frame Structure

The ISO15765-2 protocol uses the CAN frame structure but overlays one or more CAN data bytes with a Protocol Control Information(PCI) byte and depending on the Message Type additional PCI specific parameter(s). If normal addressing is configured the PCI byte overlays the first CAN data byte, refer to Figure 20. If extended addressing is configured the extended address byte overlays the first CAN data byte and the PCI byte overlays the second CAN data byte, refer to Figure 21. Because of PCI information overhead the ISO15765-2 frame accommodates less user data than the Standard CAN frame.

PCI Frame Name	**CAN Frame Structure**									
	Arbitration	**Control**	**Data (Bytes,1..8)** 1	2	3	4	5	6	7	8
SingleFrame(SF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**PCI**							
FirstFrame(FF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**PCI**	**FL**						
ConsecutiveFrame(CF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**PCI**							
FlowControl(FC)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**PCI**	**BS**	ST min					

Figure 20 ISO15765-2 Message Types mapped to Standard CAN Frame, Normal Addressing

PCI Frame Name	**CAN Frame Structure**									
	Arbitration	**Control**	**Data (Bytes,1..8)** 1	2	3	4	5	6	7	8
SingleFrame(SF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**Ext Addr**	PCI						
FirstFrame(FF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**Ext Addr**	PCI	FL					
ConsecutiveFrame(CF)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**Ext Addr**	PCI						
FlowControl(FC)	**11-bit ID, RTR**	**IDE, RB0, Data Length**	**Ext Addr**	PCI	BS	ST min				

Figure 21 ISO15765-2 Message Types mapped to Standard CAN Frame, Extended Addressing

Copyright 2003 Drew Technologies, Inc.

ISO15765-2 PCI Message Description

The PCI byte encodes a Message Type value in the upper four bits and a message specific parameter in the lower four bits. The defined PCI message types are: SingleFrame, FirstFrame, ConsecutiveFrame and FlowControl.

The SingleFrame message is used for data transfers where the data fits entirely into the data field. This message type is used for unsegmented message transfers. The Data Length bits specifies the number of bytes contained in the message. The SingleFrame transactions can use both physical(peer to peer) and functional(peer to many) addressing(CAN Identifiers).

A message bigger than a single CAN frame will be transferred as sequence of multiple CAN frames and this process is referred to as a segmented message transfer. The node initiating the segmented transaction is the transmitting peer and the node receiving and reassembling the CAN frame segments is the receiving peer. The FirstFrame, ConsecutiveFrame and FlowControl messages are used exclusively for segmented message transfers.

The FirstFrame message is sent by the transmitting peer and denotes the start of a segmented message transfer. The Frame Length is the length of the entire message and the maximum allowable value is 4095 bytes. The FirstFrame carries the first 5(extended address mode) or 6(normal address mode) bytes of the message data. Segmented message transactions can only use physical(peer to peer) addressing mode.

The transmitting peer uses the ConsecutiveFrame message for the second through last data frames of a segmented message transfer. The Sequence Number is used by the receiving peer for ordering the frame segments during the reassembly process. The ConsecutiveFrame carries 6(extended address mode) or 7(normal address mode) bytes of the message data(except for the last frame when the original message size is not an exact multiple of 6 or 7).

The FlowControl message is used by the receiving peer to start, stop, abort and configure the transfer rate by specifying parameters, which shape the ConsecutiveFrame burst. It is sent in response to FirstFrame or last ConsecutiveFrame message in a burst. The Flow Status indicates whether the transmitting peer can proceed with the message transfer. The Flow Status values are: ContinueToSend, Wait and OverFlow. The ContinueToSend status informs the transmitting peer that the receiving peer can handle another BlockSize burst of ConsecutiveFrames. The Wait status informs the transmitting peer to pause until the next FlowControl message. The OverFlow status informs the transmitting peer that the segmented frame length exceeds the receivers buffering capacity. The OverFlow status is only sent as a response to the FirstFrame message.

The BlockSize parameter is the number of ConsecutiveFrames that can be sent in a burst before waiting for the next FlowControl message. The receiving peer sends the FlowControl message after the last ConsecutiveFrame of a transfer burst. The SeparationTime is the minimum time to pause between transmitting ConsecutiveFrames.

Copyright 2003 Drew Technologies, Inc.

PCI Frame Name	PCI Byte structure								Data Bytes		
	Message Type encoding bits				Message specific encoding bits						
	7	6	5	4	3	2	1	0	2	3	4..8
SingleFrame(SF)	0	0	0	0	Data Length				Data(2..8)		
FirstFrame(FF)	0	0	0	1	Extended Frame Length				**Frame Length**	Data(3..8)	
ConsecutiveFrame(CF)	0	0	1	0	Sequence Number				Data(2..8)		
FlowControl(FC)	0	0	1	1	Flow Status				BS	**STmin**	-

Figure 22 ISO15765-2 Protocol Control Information encoding, Normal Addressing

ISO15765-2 Multiple Frame Transmission using segmentation

An example message flow for a generic segmented message transfer is depicted in Figure 23. The original 35-byte message consumed one FirstFrame and five ConsecutiveFrame messages. Assuming normal addressing mode, the FirstFrame carries 6 data bytes, the next four ConsecutiveFrames carry 7 data bytes and the last ConsecutiveFrame carries 1 data byte. The receiving peer's first and second FlowControl messages set the Flow Status(FS) to ContinueToSend(CTS), the BlockSize(BS) to 3 and SeparationTime minimum to 5 milliseconds. Adhering to the receiving peer's message assembly capabilities, the transmitting peer divided the 35-byte message into two Consecutive Frame bursts.

Copyright 2003 Drew Technologies, Inc.

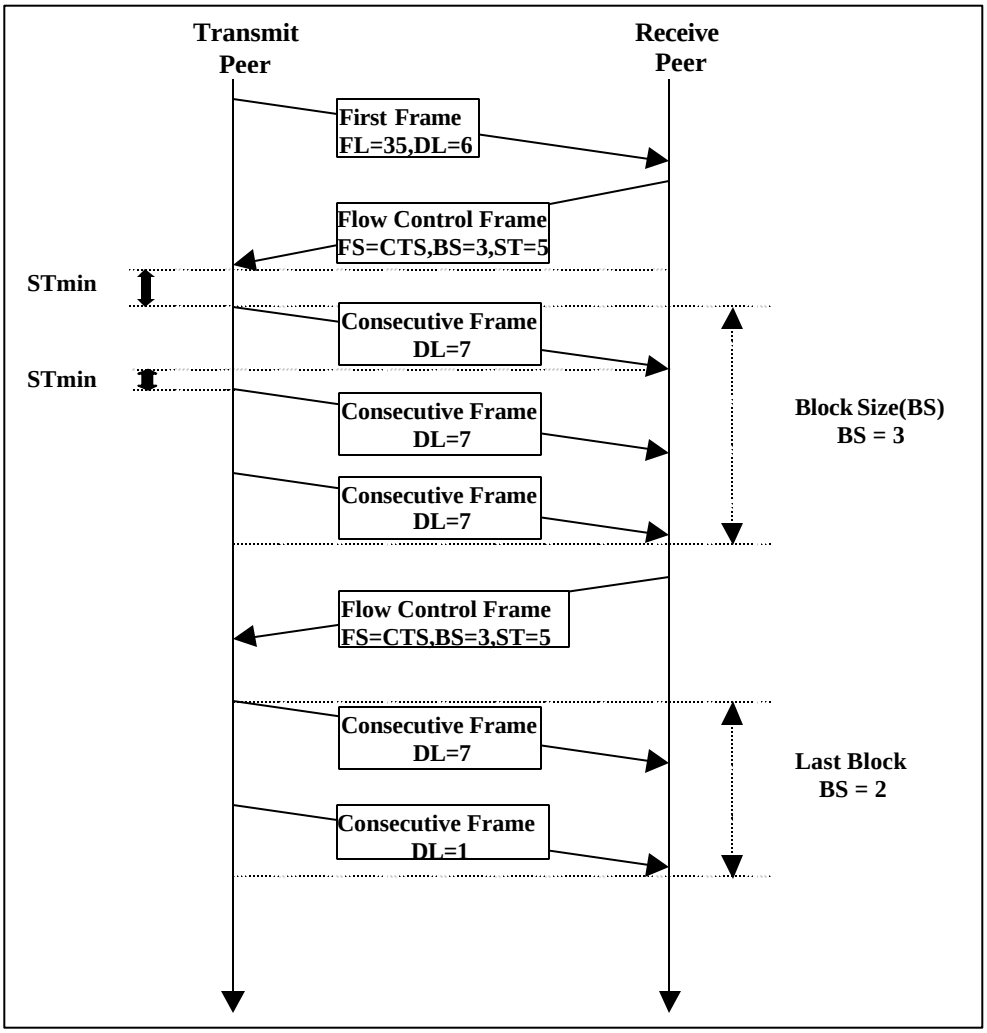


Figure 23 ISO15765-2 Flow Control Mechanism

ISO15765-2 segmented transfer in a CAN network using a Flow Control filter

An example message flow for a segmented message transfer where the UserApplication plays the role of the transmitting peer is depicted in Figure 24.

The segmented transfer uses physical addressing to identify both the transmitting and receiving peer. The UserApplication must have detailed knowledge of the CAN Identifiers assigned to the ECUs within the CAN network. The ECUs are assigned predefined physical CAN Identifiers when the CAN network is created. In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address in responding to messages from external devices. This is because the 11-bit CAN Identifier does not encode the physical address of the receiving or transmitting peer. In the case of 29-bit CAN Identifiers, the message(FirstFrame, FlowControl or ConsecutiveFrame) Identifier itself contains the physical addresses of both the transmitting and receiving peers.

Copyright 2003 Drew Technologies, Inc.

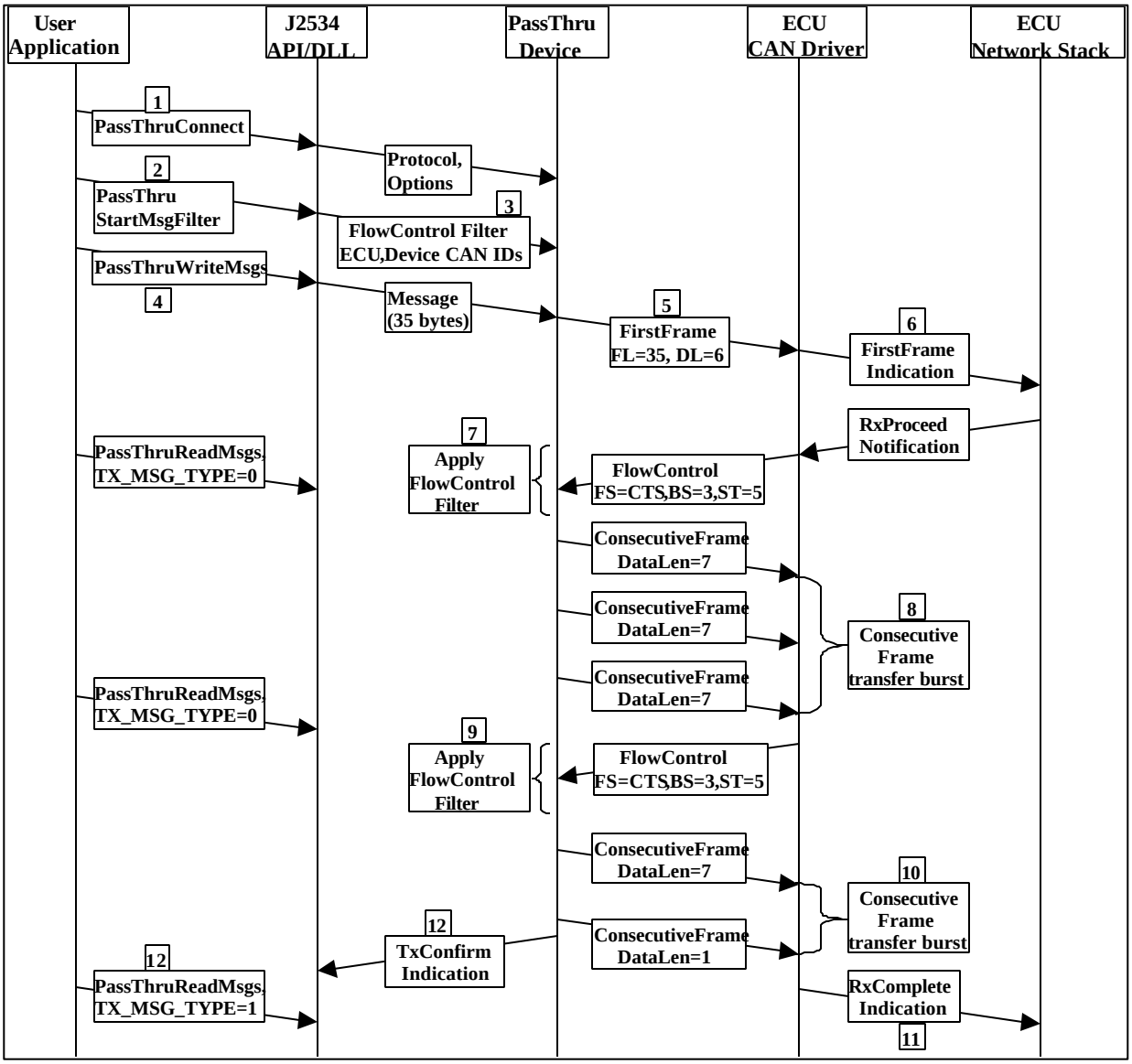


Figure 24 UserApplication initiated segmented transfer using a FlowControl filter

1. The UserApplication establishes an ISO15765 communication channel to the vehicle network using the specified protocol options(11-bit CAN Identifier, no extended address).
2. The UserApplication selects the physical CAN Identifiers for the target ECU and the PassThru device. The UserApplication builds a FlowControl message filter and sends it to the PassThru device.
3. The PassThru device writes the ECU CAN Identifier and Mask values to its CAN transceiver message acceptance filter and mask registers. The PassThru device CAN Identifier is written to its internal ISO15765 FlowControl table.
4. The UserApplication builds a request message addressed to the target ECU and sends it to the PassThru device. The request message is larger than a single CAN frame.
5. The PassThru device extracts the target ECU CAN Identifier from the UserApplication message. The PassThru device builds a FirstFrame message using the extracted CAN Identifier as the target ECU physical address. The PassThru device transmits the FirstFrame message onto the CAN bus.
6. The target ECU receives the FirstFrame message and delivers the FirstFrame indication to the ECU Network Stack. The ECU Network Stack notifies the ECU driver that it is ready to service a PassThru device request. The ECU driver makes note of the segmented message length(Frame Length in

Copyright 2003 Drew Technologies, Inc.

ISO15765-2 header). The ECU builds a FlowControl message, which contains parameters that will regulate the PassThru device first consecutive frame transfer burst. In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address for the PassThru device. In the case of 29-bit Identifiers, the FlowControl Identifier itself contains the physical address of the PassThru device.

7. The PassThru device CAN transceiver message acceptance filter and mask registers allow the FlowControl message to enter the transceivers receive register. The PassThru device decodes the FlowControl message and uses the BlockSize and SeparationTime values for shaping and timing its first consecutive frame burst(3 ConsecutiveFrames spaced at least SeparationTime apart). The PassThru device transmits 3 ConsecutiveFrame messages onto the CAN bus.
8. The target ECU receives the 3 consecutive frames, reassembles the data by removing the CAN and ISO15765-2 header information and updates number of message bytes received. The target ECU builds a FlowControl message, which contains parameters that will regulate the PassThru device second consecutive frame transfer burst. In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address for the PassThru device. In the case of 29-bit Identifiers, the ConsecutiveFrame Identifier itself contains the physical address of the PassThru device.
9. The PassThru device CAN transceiver message acceptance filter and mask registers allow the FlowControl message to enter the transceivers receive register. The PassThru device decodes the FlowControl message and uses the BlockSize and SeparationTime values for shaping and timing its second consecutive frame burst(2 ConsecutiveFrames spaced at least SeparationTime apart). The PassThru device transmits 2 ConsecutiveFrame messages onto the CAN bus.
10. The target ECU receives the 2 consecutive frames, reassembles the data by removing the CAN and ISO15765-2 header information and updates number of message bytes received. The target ECU determines that the number of message bytes specified in the FirstFrame message have all been received. The ECU CAN Driver delivers a receive frame complete indication to the ECU Network Stack.
11. The PassThru device delivers a transmit confirm indication to the J2534 API/DLL after its CAN transceiver signals that the message was successfully transmitted. The UserApplication must enable LOOPBACK mode(PassThruIoctl SET_CONFIG, LOOPBACK) in order to obtain a transmit confirm indication that contains the original transmit frame.
12. The UserApplication receives a PASSTHRU_MSG structure containing no data but with the RxStatus TX_MSG_TYPE bit set. The UserApplication has been periodically polling the J2534 API/DLL waiting for the transmit confirm indication before proceeding with another message transfer.

Copyright 2003 Drew Technologies, Inc.

An example message flow for a segmented message transfer where the UserApplication plays the role of the receiving peer is depicted in Figure 25.

The segmented transfer uses physical addressing to identify both the transmitting and receiving peer. The UserApplication must have detailed knowledge of the CAN Identifiers assigned to the ECUs within the CAN network. The ECUs are assigned predefined physical CAN Identifiers when the CAN network is created. In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address in responding to messages from external devices. This is because the 11-bit CAN Identifier does not encode the physical address of the receiving or transmitting peer. In the case of 29-bit CAN Identifiers, the message(FirstFrame, FlowControl or ConsecutiveFrame) Identifier itself contains the physical addresses of both the transmitting and receiving peers.

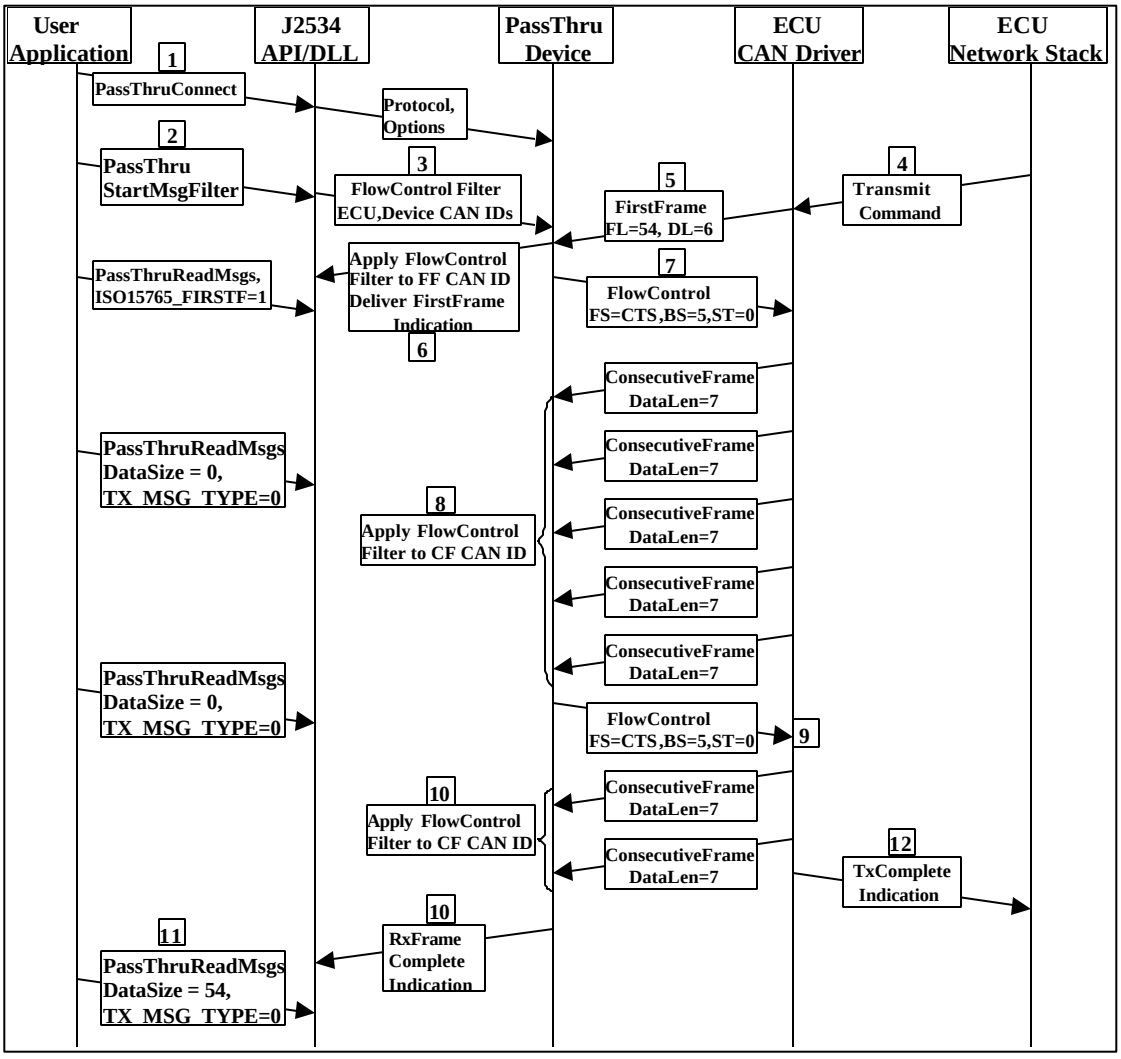


Figure 25 ECU initiated segmented transfer using a Flow Control filter

1. The UserApplication establishes an ISO15765 communication channel to the vehicle network using the specified protocol options(11-bit CAN Identifier, no extended address).
2. The UserApplication selects the physical CAN Identifiers for the target ECU and the PassThru device. The UserApplication builds a FlowControl message filter and sends it to the PassThru device.

Copyright 2003 Drew Technologies, Inc.

3. The PassThru device writes the ECU CAN Identifier and Mask values to its CAN transceiver message acceptance filter and mask registers. The PassThru device CAN Identifier is written to its internal ISO15765 FlowControl table.
4. The ECU Network Stack builds a response message addressed to the PassThru device and delivers a transmit request to the ECU CAN driver. The response message is larger than a single CAN frame.
5. The ECU driver builds a FirstFrame message using the PassThru device CAN Identifier as the physical address. In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address for the PassThru device. In the case of 29-bit Identifiers, the FirstFrame or ConsecutiveFrame Identifier(from a previous request transaction) contains the physical address of the PassThru device. The ECU driver transmits the FirstFrame message onto the CAN bus.
6. The PassThru device CAN transceiver message acceptance filter and mask registers allow the FirstFrame message to enter the transceivers receive register. The PassThru device decodes the FirstFrame message and makes note of the segmented message length(Frame Length in ISO15765-2 header). The PassThru device delivers the FirstFrame indication to the J2534 API/DLL. The J2534 API/DLL notifies the UserApplication that a receive frame from a known ECU is being reassembled. The UserApplication receives a PASSTHRU_MSG structure containing no data but with the RxStatus ISO15765_FIRST_FRAME bit set. The PassThru device builds a FlowControl message, which contains parameters that will regulate the ECU drivers, first consecutive frame transfer burst.
7. The ECU CAN transceiver message acceptance filter and mask registers allow the FlowControl message to enter the transceivers receive register. The ECU driver decodes the FlowControl message and uses the BlockSize and SeparationTime values for shaping and timing its first consecutive frame burst(3 ConsecutiveFrames spaced at least SeparationTime apart). In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address for the PassThru device. In the case of 29-bit Identifiers, the FlowControl Identifier contains the physical address of the PassThru device. The ECU driver transmits the 3 ConsecutiveFrame messages onto the CAN bus.
8. The PassThru device CAN transceiver message acceptance filter and mask registers allow the ConsecutiveFrame messages to enter the transceivers receive register. The PassThru device receives the 3 consecutive frames, reassembles the data by removing the CAN and ISO15765-2 header information and updates number of message bytes received. The PassThru device builds a FlowControl message, which contains parameters that will regulate the ECU driver second consecutive frame transfer burst.
9. The ECU CAN transceiver message acceptance filter and mask registers allow the FlowControl messages to enter the transceivers receive register. The PassThru device decodes the FlowControl message and uses the BlockSize and SeparationTime values for shaping and timing its second consecutive frame burst(2 ConsecutiveFrames spaced at least SeparationTime apart). In the case of 11-bit CAN Identifiers, the ECU uses a predetermined physical address for the PassThru device. In the case of 29-bit Identifiers, the FlowControl Identifier contains the physical address of the PassThru device. The ECU driver transmits the 2 ConsecutiveFrame messages onto the CAN bus.
10. The PassThru device receives the 2 consecutive frames, reassembles the data by removing the CAN and ISO15765-2 header information and updates number of message bytes received. The PassThru device determines that the number of message bytes specified in the FirstFrame message have all been received. The PassThru device delivers a receive frame complete indication to the J2534 API/DLL.
11. The UserApplication receives a PASSTHRU_MSG structure containing the response data and with the RxStatus TX_MSG_TYPE bit set. The UserApplication has been periodically polling the J2534 API/DLL waiting for the receive complete indication before proceeding with another message transfer.
12. The ECU driver delivers a transmit confirm indication to the ECU Network Stack after its CAN transceiver signals that the message was successfully transmitted.

Copyright 2003 Drew Technologies, Inc.

Flow Control Program Example

The code sample below demonstrates how to setup a Flow Control filter using PassThruStartMsgFilter function parameters and how to read multi-segmented messages. On successful completion the FilterID parameter will contain the filter handle that is required for the PassThruStopMsgFilter function call.

```
typedef struct {
   unsigned long ProtocolID        /* vehicle network protocol */
   unsigned long RxStatus;         /* receive message status */
   unsigned long TxFlags;          /* transmit message flags */
   unsigned long Timestamp;        /* receive message timestamp(in microseconds) */
   unsigned long DataSize;         /* byte size of message payload in the Data array */
   unsigned long ExtraDataIndex;   /* start of extra data(i.e. CRC, checksum, etc) in Data array */
   unsigned char Data[4128];       /* message payload or data */
} PASSTHRU_MSG;

typedef struct {
   unsigned long Parameter;        /* Name of configuration parameter */
   unsigned long Value;            /* Value of configuration parameter */
} SCONFIG;

typedef struct {
   unsigned long NumOfParams;      /* sizeof SCONFIG array */
   SCONFIG *ConfigPtr;             /* array containing configuration item(s) */
} SCONFIG_LIST;

unsigned long status;
unsigned long ChannelID;           /* Logical channel identifier returned by PassThruConnect */
unsigned long FilterID;
unsigned long NumCanMsg;
PASSTHRU_MSG MaskMsg;
PASSTHRU_MSG PatternMsg;
PASSTHRU_MSG FilterMsg;
PASSTHRU_MSG CanMsg;
SCONFIG CfgItem[2];
SCONFIG_LIST Input;
int FrameWait;
char errstr[256];

status = PassThruConnect(ISO15765, 0x00000000, &ChannelID);

/*
** Setup the PassThru device receive message assembly capabilities.
** BlockSize is the number ConsecutiveFrames that can be received in a burst.
** SeparationTime is the minimum time to pause between transmitting ConsecutiveFrames.
*/
CfgItem[0].Parameter = ISO15765_BS;
CfgItem[0].Value = 0x20;                    /* BlockSize is 32 frames */
CfgItem[1].Parameter = ISO15765_STMIN;
CfgItem[1].Value = 0x0x01;                  /* SeparationTime is 1 millisecond */

Input.NumOfParams = 2;                      /* Configuration list has 2 items */
Input.ConfigPtr = &CfgItem;

/*
```

Copyright 2003 Drew Technologies, Inc.

```
** The PassThru device is asked to use the UserApplication selected BlockSize and
** SeparationTime values instead of the J2534 specified defaults.
*/
status = PassThruIoctl(ChannelID, SET_CONFIG, (void *)&Input, (void *)&NULL);

/*
** Set RxStaus, TxFlags, TimeStamp and ExtraDataIndex to zero.
** TxFlags = 0x00000000 implies:
**              No blocking on Tx complete, 11-bit CAN Identifier, normal addressing,
**              no zero padding of FlowControl message.
** The ProtocolID, DataSize and Data array are set below.
*/
memset(&MaskMsg, 0, sizeof(MaskMsg));

/*
** This example assumes 11-bit CAN Identifiers are used by the ECUs connected to the bus.
** Set the Mask to look at all 11 bits of the Can Identifier portion of a receive frame.
** MaskMsg.Data[0] and MaskMsg[0].Data[1] set to zero by previous memset.
*/
MaskMsg.ProtocolID = ISO15765;
MaskMsg.Data[2] = 0x07;
MaskMsg.Data[3] = 0xFF;
MaskMsg.DataSize = 4;     /* Mask message contains 4 bytes */

/*
** Set RxStaus, TxFlags, TimeStamp and ExtraDataIndex to zero.
** TxFlags = 0x00000000 implies:
**              No blocking on Tx complete, 11-bit CAN Identifier, normal addressing,
**              no zero padding of FlowControl message.
** The ProtocolID, DataSize and Data array are set below.
*/
memset(&PatternMsg, 0, sizeof(PatternMsg));

/*
** The Can Identifier is a 11-bit OBD CAN Identifier used for physically addressed response messages
** originating from ECU #1 and destined to the PassThru device.
** PatternMsg.Data[0] and PatternMsg[0].Data[1] set to zero by previous memset.
*/
PatternMsg.ProtocolID = ISO15765;
PatternMsg.Data[2] = 0x07;
PatternMsg.Data[3] = 0xE0;
PatternMsg.DataSize = 4;   /* Pattern message contains 4 bytes */

/*
** Set RxStaus, TxFlags, TimeStamp and ExtraDataIndex to zero.
** TxFlags = 0x00000000 implies:
**              No blocking on Tx complete, 11-bit CAN Identifier, normal addressing,
**              no zero padding of FlowControl message.
** The ProtocolID, DataSize and Data array are set below.
*/
memset(&FilterMsg, 0, sizeof(FilterMsg));

/*
** The Can Identifier is a 11-bit OBD CAN Identifier used for physically addressed request messages
** originating from the PassThru device and destined to ECU #1.
** FilterMsg.Data[0] and FilterMsg[0].Data[1] set to zero by previous memset.
```

```
*/
FilterMsg.ProtocolID = ISO15765;
FilterMsg.Data[2] = 0x07;
FilterMsg.Data[3] = 0xE8;
FilterMsg.DataSize = 4;     /* Filter message contains 4 bytes */

status = PassThruStartMsgFilter(ChannelID, FLOW_CONTROL_FILTER, &MaskMsg, &PatternMsg,
                                        &FilterMsg, &FilterID,);

if (status != STATUS_NOERROR)
{
        /*
        ** PassThruStartMsgFilter failed! Get descriptive error string.
        */
        PassThruGetLastError(&errstr[0]);

        /*
        ** Display Error dialog box and/or write error description to Log file.
        */
}

memset(&CanMsg, 0, sizeof(CanMsg));

/*
** Read multi-segmented ISO15765 messages and handle ISO15765_FIRST_FRAME indication.
*/
while(1)
{
        /*
        ** Post read request to PassThru device's ISO15765 receive queue.
        */
        NumCanMsg = 1;
        status = PassThruReadMsgs(ChannelID, &CanMsg, &NumCanMsg, 0);

        /*
        ** Special processing highlighted for ISO15765 FirstFrame Indication.
        */
        if ((status == STATUS_NOERROR) && (CanMsg.RxStatus & ISO15765_FIRST_FRAME))
        {
                /*
                ** No Error Indication provided after FirstFrame Indication for an aborted
                ** multi-segmented frame transfer. Give up frame watch after wait interval expires.
                */
                FrameWait = 16;

                while(FrameWait-- > 0)
                {
                        /*
                        ** Post read request for expected multi-segmented CAN frame.
                        */
                        NumCanMsg = 1;
                        status = PassThruReadMsgs(ChannelID, &CanMsg, &NumCanMsg, 0);

                        if (status == ERR_BUFFER_EMPTY)
                        {
                                /*
```

```
                        ** Wait 1 ms. before posting another receive queue read request.
                        */
                        Sleep(1);
                        continue;
                }
                else if (status == STATUS_NOERROR)
                {
                        /*
                        ** Notify CAN packet handler of multi-segmented CAN frame arrival.
                        */
                        MyCanPktHandler(&CanMsg);
                }
                else
                {
                        /*
                        ** Notify receive packet error handler about unexpected problem.
                        */
                        MyReceivePktErrorHandler(status, &CanMsg);
                }
                /*
                ** Either unexpected error or processed valid multi-segmented CAN frame.
                */
                break;
        }

        if (FrameWait <= 0)
        {
                /*
                ** Notify receive indication error handler about missing frame.
                */
                MyReceiveIndErrorHandler();
        }
    }
    else if ((status == STATUS_NOERROR) && (CanMsg.Datasize > 0))
    {
        /*
        ** Notify CAN packet handler of non-segmented CAN frame arrival.
        */
        MyCanPktHandler(&CanMsg);
    }
    else if (status != ERR_BUFFER_EMPTY)
    {
        /*
        ** Notify receive packet error handler about unexpected problem.
        */
        MyReceivePktErrorHandler(status, &CanMsg);
    }

    /*
    ** Wait 5 milliseconds before posting another receive queue read request.
    */
    Sleep(5);
}
```

References

SAE and ISO documents of interest are listed below.

[1] SAE J1850 Class B Data Communications Network Interface, specifies requirements for a vehicle data communications network.
[2] SAE J2534 Recommended Practice for Pass-Thru Vehicle Programming
[3] SAE J2178/1 Class B Data Communication Network Messages: Detailed Header Formats and Physical Address assignments
[4] SAE J2178/2 Class B Data Communication Network Messages: Data Parameter Definitions
[5] SAE J2178/3 Class B Data Communication Network Messages: Frame Ids for Single Byte Forms of Headers
[6] SAE J2178/4 Class B Data Communication Network Messages: Message Definitions for Three Byte Headers
[7] SAE J1939 Truck and Bus Control and Communications Network
[8] SAE J1962 Diagnostic Connector defines a 16-pin standard connector.
[9] SAE 2610 DaimlerChrysler Information Report for Serial Data Communications Interface(SCI).
[10] ISO 7637-1 Road Vehicles – Electrical disturbance by conduction and coupling – Passenger cars and light commercial vehicles with nominal 12 V supply voltage
[11] ISO 9141-2 Road Vehicles – Diagnostic systems – CARB requirements for interchange of digital information
[12] ISO 11898 Road Vehicles – Interchange of digital information – Controller area network(CAN) for high speed communication
[13] ISO 14230-4 Road Vehicles – Diagnostic systems – Keyword protocol 2000 – Requirements for emission-related systems
[14] ISO 15765-2 Road Vehicles – Diagnostics on controller area networks(CAN) – Network Layer Services
[15] ISO 15765-4 Road Vehicles – Diagnostics on controller area networks(CAN) – Requirements for emission-related systems

S.A.E
400 Commonwealth Drive
Warrendale, PA 15096-0001 U.S.A.
(724) 776-4841
FAX (724) 776-0790
www.sae.org

ANSI
25 West 43rd Street
New York, NY 10036-8002
(212) 642-4900
(212) 398-0023
www.ansi.org

Copyright 2003 Drew Technologies, Inc.