



Dearborn Group *Technology*

EASY API

Version: 1.0

USER'S MANUAL

©2005: Dearborn Group Inc.
27007 Hills Tech Court
Farmington Hills, MI 48331
Phone (248) 488-2080 • Fax (248) 488-2082
<http://www.dgtech.com>

This document is copyrighted by the Dearborn Group, Inc.

Permission is granted to copy any or all portions of this manual, provided that such copies are for use with the product provided by the Dearborn Group, and that the name "Dearborn Group, Inc." remain on all copies as on the original.

Table of Contents

1	<i>Abbreviations / Terms Used</i>	3
2	<i>References</i>	3
3	<i>INTRODUCTION</i>	4
3.1	Purpose	4
3.2	Scope	4
4	<i>Overview</i>	4
5	<i>Constraints</i>	4
6	<i>Device Setting</i>	4
7	<i>Easy API Functions</i>	5
7.1	Function Name: PassThruOpen	5
7.2	Function Name: PassThruClose	6
7.3	Function Name: PassThruConnect	7
7.4	Function Name: PassThruDisconnect	8
7.5	Function Name: PassThruReadMsgs	9
7.6	Function Name: PassThruWriteMsgs	10
7.7	Function Name: PassThruStartPeriodicMsg	11
7.8	Function Name: PassThruStopPeriodicMsg	12
7.9	Function Name: PassThruStartMsgFilter	13
7.10	Function Name: PassThruStopMsgFilter	15
7.11	Function Name: PassThruSetProgrammingVoltage	16
7.12	Function Name: PassThruReadVersion	17
7.13	Function Name: PassThruGetLastError	18
7.14	Function Name: PassThruIoctl	19
8	<i>Document Revision History</i>	20

1 Abbreviations / Terms Used

API	Application Programming Interface
DLL	Dynamic Link Library
Gryphon™	Dearborn Group TCP/IP to Automotive Network Interface Hardware
IP	Internet Protocol
SAE	Standard for Automotive Engineers

2 References

SAE J2534 Pass-Thru Vehicle Programming document – Revision 04.04
(This can be obtained from the SAE or the Dearborn Group, Inc. website.)

3 INTRODUCTION

3.1 Purpose

The purpose of this document is to provide a reference guide for APIs implemented in EasyAPI DLL.

3.2 Scope

The audiences for this document are the Software Development Engineers with knowledge of C/C++ language, a basic understanding of various Automotive Protocols and general familiarity with the Gryphon™ device. This document makes frequent references to “SAE J2534 Pass-Thru Vehicle Programming” document (Revision 04.04) throughout the manual and hence it is strongly recommended that a copy of this document be at hand all the time while developing an application. This document addresses the features implemented in the Easy API Release Version 4.00.

4 Overview

EasyAPI is a DLL that exports various APIs to communicate with Gryphon™. The DLL is based on WIN32 architecture and is meant to be loaded dynamically. The references to APIs are expected to be resolved at runtime by an application (Refer to SAE J2534 Section 9.2.2). The device specific settings are configured in the Registry of a PC running the DLL (Refer to Section 7). The communication is carried out over an Ethernet link and in turn with various protocols as requested. The DLL is called “j2534g32.dll” that resides under “System” or “System32” directory of the operating system.

5 Constraints

The DLL can be used to communicate with only one Gryphon™ at a time.

6 Device Setting

The device specific settings could be configured through a registry. Refer to SAE J2534 Section 9.2 for general description about the registry. The “VendorSpecificValues” of SAE J2534 Section 9.2 is used to set the device specific values. These values along with the other entries are as described in the table below.

DeviceID	DWORD	502
Vendor	String	“Dearborn Group Technology, Inc.”
Name	String	“Gryphon”
ProtocolsSupported	String	“CAN:1,ISO15765:1,J1850PWM:1,J1850VPW:1,ISO9141:1,ISO14230:1”
CAN	DWORD	2
ISO15765	DWORD	2
J1850PWM	DWORD	1
J1850VPW	DWORD	2
ISO9141	DWORD	1
ISO14230	DWORD	1
ConfigApplication	String	“C:\Gryphon2534\GryphonConfig.exe”
APIVersion	String	“04.04”
FunctionLibrary	String	“C:\WINDOWS\SYSTEM\j2534g32.dll”
IPAddress	String	The IP Address of the Gryphon that is used by “j2534g32.dll”. Ex: “192.168.1.1”
PortNum	DWORD	The port number that the Gryphon is connected on.
ProductVersion	String	The product version of the J2534 Gryphon DLL setup that was last installed.
TimeStampResolution	DWORD	The resolution on the timestamp from the Gryphon in

(microseconds)		microseconds.
Logging	DWORD	The flag that indicates whether or not to have logging during the DLL run.

Configure IP Address

Use the "IPAddress" key of the table above to configure the IP Address of the Gryphon.

7 Easy API Functions

7.1 Function Name: PassThruOpen

7.1.1 Description:

This function establishes connection to a Gryphon™ with an IP Address set in the registry and returns a newly created ID that is used as a handle to the Gryphon™. Refer to SAE J2534 Section 7.2.1.

7.1.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruOpen
{
    void *pName,
    unsigned long *pulDeviceID
}
```

7.1.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.1.2.

7.1.1.3 Return Values

Refer to SAE J2534 Section 7.2.1.3.

7.1.2 Example:

```
unsigned long ulErrorNumCode;
unsigned long ulDeviceID;
```

```
// Establish the connection to a Gryphon and returns the Device ID into ulDeviceID
// as a handle to the Gryphon.
ulErrorNumCode = PassThruOpen(&ulDeviceID);
```

7.2 Function Name: PassThruClose

7.2.1 Description:

This function closes the connection to a Gryphon™
Refer to SAE J2534 Section 7.2.2.

7.2.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruClose  
{  
    unsigned long ulDeviceID  
}
```

7.2.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.2.2.

7.2.1.3 Return Values

Refer to SAE J2534 Section 7.2.2.3.

7.2.2 Example:

```
unsigned long ulErrorNumCode;  
unsigned long ulDeviceID;
```

```
ulDeviceID = 0x01;           // Use the Device ID which is set to 1
```

```
ulErrorNumCode = PassThruClose(ulDeviceID);
```

7.3 Function Name: PassThruConnect

7.3.1 Description:

This function connects to a given protocol channel on Gryphon™ and returns a newly created ID that is used for all subsequent communication over that protocol and channel.
Refer to SAE J2534 Section 7.2.3.

7.3.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruConnect
{
    unsigned long ulDeviceID,
    unsigned long ulProtocolID,
    unsigned long ulFlags,
    unsigned long ulBaudRate,
    unsigned long *pulChannelID
}
```

7.3.1.2 Parameter Description

ulDeviceID:

Refer to SAE J2534 Section 7.2.3.2.

ulProtocolID:

Value(s)	Description	Definition
0x0001 – 0xFFFFFFFF	Refer to SAE J2534 Section 7.2.3.4	Refer to SAE J2534 Section 7.2.3.4

ulFlags:

Flag Bit(s)	Description	Value
31-24	Channel number of Gryphon™ .	0x00 to 0xFF
23-16	Refer to SAE J2534 Section 7.2.3.3	Refer to SAE J2534 Section 7.2.3.3

ulBaudRate:

Refer to SAE J2534 Section 7.2.3.2.

pulChannelID:

Refer to SAE J2534 Section 7.2.3.2.

7.3.1.3 Return Values

Refer to SAE J2534 Section 7.2.3.5.

7.3.2 Example:

```
unsigned long ulErrorNumCode;
```

```
unsigned long ulFlags, ulChannelID, ulProtocolID, ulDeviceID, ulBaudRate;
```

```
ulDeviceID = 0x01;           // Use the Device ID which is set to 1
ulProtocolID = 0x05;        // Set protocol to CAN.
ulFlags = 0x01000000;      // Set the most significant byte to 1 to connect to channel 1.
ulBaudRate = 500000;       // Set the CAN baud rate to 500,000
```

```
// Establish channel to communicate over CAN bus  
// If successful, it will return the channel ID into ulChannelID  
// for any further communication over this protocol and channel.  
ulErrorNumCode = PassThruConnect(ulDeviceID, ulProtocolID, ulFlags, ulBaudRate, &ulChannelID);
```

7.4 Function Name: PassThruDisconnect

7.4.1 Description:

This function is used to disconnect from a protocol channel.
Refer to SAE J2534 Section 7.2.4.

7.4.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruDisconnect  
{  
    unsigned long ulChannelID  
}
```

7.4.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.4.2.

7.4.1.3 Return Values

Refer to SAE J2534 Section 7.2.4.3.

7.4.2 Example:

```
unsigned long ulErrorNumCode;  
unsigned long ulChannelID;
```

```
// Disconnect from the protocol using the ulChannelID that was returned when connecting to  
// that protocol.  
ulErrorNumCode = PassThruDisconnect(ulChannelID);
```

7.5 Function Name: PassThruReadMsgs

7.5.1 Description:

This function is used to read messages.
Refer to SAE J2534 Section 7.2.5.

7.5.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruReadMsgs
{
    unsigned long ulChannelID,
    PASSTHRU_MSG *pstrucJ2534Msg,
    unsigned long *pulNumMsgs,
    unsigned long ulTimeout
}
```

7.5.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.5.2.

7.5.1.3 Return Values

Refer to SAE J2534 Section 7.2.5.3.

7.5.2 Example:

```
unsigned long    ulErrorNumCode;
unsigned long    ulChannelID, ulNumMsgs, ulTimeout;
PASSTHRU_MSG    *pstrucJ2534Msg;

ulNumMsgs = 5;           // Read 5 messages
ulTimeout = 1000;       // Allow 1000 ms to read in all 5 messages

// Allocate memory to read messages
pstrucJ2534Msg = (PASSTHRU_MSG *) malloc (sizeof(PASSTHRU_MSG) * ulNumMsgs);

// Read messages from the receive buffer using the ulChannelID that was returned when connecting
// to that protocol.
ulErrorNumCode = PassThruReadMsgs(ulChannelID, pstrucJ2534Msg,
    (unsigned long *)&ulNumMsgs, ulTimeout);

// Free the memory
free(pstrucJ2534Msg);
```

7.6 Function Name: PassThruWriteMsgs

7.6.1 Description:

This function is used to send messages.
Refer to SAE J2534 Section 7.2.6.

7.6.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruWriteMsgs
{
    unsigned long ulChannelID,
    PASSTHRU_MSG *pstrucJ2534Msg,
    unsigned long *pulNumMsgs,
    unsigned long ulTimeout
}
```

7.6.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.6.2.

7.6.1.3 Return Values

Refer to SAE J2534 Section 7.2.6.3.

7.6.2 Example:

```
unsigned long    ulErrorNumCode;
unsigned long    ulChannelID, ulNumMsgs, ulTimeout;
PASSTHRU_MSG    *pstrucJ2534Msg;

ulNumMsgs = 1;           // Write 1 message
ulTimeout = 1000;       // Allow 1000 ms to write 1 message (blocking)
                        // If non-blocking, set it to 0

// Allocating memory to write messages
pstrucJ2534Msg = (PASSTHRU_MSG *) malloc (sizeof(PASSTHRU_MSG) * ulNumMsgs);

// Setting the message structure
pstrucJ2534Msg ->ulProtocolID = 0x05;      // Set Protocol ID to CAN
pstrucJ2534Msg ->ulRxStatus = 0;
pstrucJ2534Msg ->ulTxFlags = 0;
pstrucJ2534Msg ->ulTimeStamp = 0;
pstrucJ2534Msg ->ulDataSize = 6;
pstrucJ2534Msg ->ulExtraDataIndex = 0;
pstrucJ2534Msg ->ucData = {0x04, 0x05, 0x0D, 0x11, 0x22, 0x33};

// Write messages to the transmit buffer using the ulChannelID that was returned when connecting to
// that protocol.
ulErrorNumCode = PassThruWriteMsgs(ulChannelID, pstrucJ2534Msg,
    (unsigned long *)&ulNumMsgs, ulTimeout);

// Free the memory
free(pstrucJ2534Msg);
```

7.7 Function Name: PassThruStartPeriodicMsg

7.7.1 Description:

This function starts sending periodic message.
Refer to SAE J2534 Section 7.2.7.

7.7.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruStartPeriodicMsg
{
    unsigned long ulChannelID,
    PASSTHRU_MSG *pstrucJ2534Msg,
    unsigned long *pulMsgID,
    unsigned long ulTimeInterval
}
```

7.7.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.7.2.

7.7.1.3 Return Values

Refer to SAE J2534 Section 7.2.7.3.

7.7.2 Example:

```
unsigned long    ulErrorNumCode;
unsigned long    ulChannelID, ulMsgID, ulTimeInterval;
PASSTHRU_MSG    *pstrucJ2534Msg;

// Allocating memory to start periodic message
pstrucJ2534Msg = (PASSTHRU_MSG *) malloc (sizeof(PASSTHRU_MSG));

// Setting the message structure
pstrucJ2534Msg ->ulProtocolID = 0x05;    // Set Protocol ID to CAN
pstrucJ2534Msg ->ulRxStatus = 0;
pstrucJ2534Msg ->ulTxFlags = 0;
pstrucJ2534Msg ->ulTimeStamp = 0;
pstrucJ2534Msg ->ulDataSize = 6;
pstrucJ2534Msg ->ulExtraDataIndex = 0;
pstrucJ2534Msg ->ucData = {0x04, 0x05, 0x0D, 0x11, 0x22, 0x33};

ulTimeInterval = 1000;    // Allow 1000 ms interval for periodic messages

// Start periodic message using the ulChannelID that was returned when connected.
// When the periodic message is started, ulMsgID will be assigned a value,
ulErrorNumCode = PassThruStartPeriodicMsg(ulChannelID, pstrucJ2534Msg,
    &ulMsgID, ulTimeInterval);

// Free the memory
free(pstrucJ2534Msg);
```

7.8 Function Name: PassThruStopPeriodicMsg

7.8.1 Description:

This function stops sending periodic message.
Refer to SAE J2534 Section 7.2.8.

7.8.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruStopPeriodicMsg
{
    unsigned long ulChannelID,
    unsigned long ulMsgID
}
```

7.8.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.8.2.

7.8.1.3 Return Values

Refer to SAE J2534 Section 7.2.8.3.

7.8.2 Example:

```
unsigned long ulErrorNumCode;
unsigned long ulChannelID, ulMsgID;
```

```
// This is to stop a periodic message using the ulChannelID that was returned when connecting to
// that protocol and ulMsgID that was return when starting that periodic message.
ulErrorNumCode = PassThruStopPeriodicMsg (ulChannelID, ulMsgID);
```

7.9 Function Name: PassThruStartMsgFilter

7.9.1 Description:

This function starts filtering incoming messages.
Refer to SAE J2534 Section 7.2.9.

7.9.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruStartMsgFilter
{
    unsigned long ulChannelID,
    unsigned long ulFilterType,
    PASSTHRU_MSG *pstrucJ2534MaskMsg,
    PASSTHRU_MSG *pstrucJ2534PatternMsg,
    PASSTHRU_MSG *pstrucJ2534FlowControlMsg,
    unsigned long *pulFilterID
}
```

7.9.1.2 Parameter Description

ulChannelID:

Refer to SAE J2534 Section 7.2.9.2.

ulFilterType:

Refer to SAE J2534 Section 7.2.9.3.

pstrucJ2534MaskMsg:

Refer to SAE J2534 Section 7.2.9.2.

pstrucJ2534PatternMsg:

Refer to SAE J2534 Section 7.2.9.2.

pstrucJ2534FlowControlMsg:

Refer to SAE J2534 Section 7.2.9.2.

pulFilterID:

Refer to SAE J2534 Section 7.2.9.2.

7.9.1.3 Return Values

Refer to SAE J2534 Section 7.2.9.4.

7.9.2 Example:

```
unsigned long    ulErrorNumCode;
unsigned long    ulChannelID, ulFilterID;
unsigned long    ulFilterType;

// Declaring message filter pointers
PASSTHRU_MSG *pstrucMaskMsgJ2534, // Mask message
              *pstrucPatternMsgJ2534, // Pattern message
              *pstrucFlowControlMsgJ2534; // Flow control message

// Set filter type
```

```

ulFilterType = 1;          // 1 – PASS_FILTER, 2 – BLOCK_FILTER,
                          // 3 – FLOW_CONTROL_FILTER (used in ISO15765)

// Allocating memory to start filter message
pstrucPatternMsgJ2534 = (PASSTHRU_MSG *) malloc (sizeof(PASSTHRU_MSG));
pstrucMaskMsgJ2534 = (PASSTHRU_MSG *) malloc (sizeof(PASSTHRU_MSG));
pstrucFlowControlMsgJ2534 = NULL;    // Used in ISO15765 only

// Setting the pattern message structure
pstrucPatternMsgJ2534 ->ulProtocolID = 0x01;    // Set Protocol ID to J1850VPW
pstrucPatternMsgJ2534 ->ulRxStatus = 0;
pstrucPatternMsgJ2534 ->ulTxFlags = 0;
pstrucPatternMsgJ2534 ->ulTimeStamp = 0;
pstrucPatternMsgJ2534 ->ulDataSize = 6;
pstrucPatternMsgJ2534 ->ulExtraDataIndex = 0;
pstrucPatternMsgJ2534 ->ucData = {0x04, 0x05, 0x0D, 0x11, 0x22, 0x33};

// Setting the mask message structure
pstrucMaskMsgJ2534 ->ulProtocolID = 0x01;    // Set Protocol ID to J1850VPW
pstrucMaskMsgJ2534 ->ulRxStatus = 0;
pstrucMaskMsgJ2534 ->ulTxFlags = 0;
pstrucMaskMsgJ2534 ->ulTimeStamp = 0;
pstrucMaskMsgJ2534 ->ulDataSize = 6;
pstrucMaskMsgJ2534 ->ulExtraDataIndex = 0;
pstrucMaskMsgJ2534 ->ucData = {0xFF, 0FF, 0xFF, 0xFF, 0xFF, 0x00};

// The pMaskMsgJ2534 ANDs with the pPatternMsgJ2534 to mask message to each
// incoming message to mask any unimportant bits.
// The enumFilterType determines how each filtered-through message will be treated.
// In this case with the enumFilterType to be PASS_FILTER, any messages that is
// {0x04, 0x05, 0x0D, 0x11, 0x22, 0x33} && {0xFF, 0FF, 0xFF, 0xFF, 0xFF, 0x00} to be
// {0x04, 0x05, 0x0D, 0x11, 0x22, 0xXX} will be passed (X – any byte value).
// Others will be blocked.

// Start message filter, assuming there is connection to J1850VPW with ulChannelID returned.
// When the message filter is started, ulFilterID will be assigned a value
ulErrorNumCode = PassThruStartMsgFilter(ulChannelID, ulFilterType,
    pstrucMaskMsgJ2534, pstrucPatternMsgJ2534, pstrucFlowControlMsgJ2534, &ulFilterID);

// Free the memory
free(pstrucMaskMsgJ2534);
free(pstrucPatternMsgJ2534);

```

7.10 Function Name: PassThruStopMsgFilter

7.10.1 Description:

This function stops filtering incoming messages.
Refer to SAE J2534 Section 7.2.10.

7.10.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruStopMsgFilter
{
    unsigned long ulChannelID,
    unsigned long ulFilterID
}
```

7.10.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.10.2.

7.10.1.3 Return Values

Refer to SAE J2534 Section 7.2.10.3.

7.10.2 Example:

```
unsigned long ulErrorNumCode;
unsigned long ulChannelID, ulFilterID;
```

```
// This is to stop a message filter using the ulChannelID that was returned when connecting to
// that protocol and using the ulFilterID that was return when setting up the filter
ulErrorNumCode = PassThruStopMsgFilter (ulChannelID, ulFilterID);
```

7.11 *Function Name:* **PassThruSetProgrammingVoltage**

7.11.1 *Description:*

This function sets a programming voltage on a pin.
Refer to SAE J2534 Section 7.2.11.

7.11.1.1 *C/C++ Prototype*

```
extern "C" long WINAPI PassThruSetProgrammingVoltage
{
    unsigned long ulDeviceID,
    unsigned long ulPinNumber,
    unsigned long ulVoltage
}
```

7.11.1.2 *Parameter Description*

ulDeviceID:

Refer to SAE J2534 Section 7.2.11.2.

ulPinNumber:

Refer to SAE J2534 Section 7.2.11.2.

ulVoltage:

Refer to SAE J2534 Section 7.2.11.3.

7.11.1.3 *Return Values*

Refer to SAE J2534 Section 7.2.11.4.

7.11.2 *Example:*

```
unsigned long ulErrorNumCode;
unsigned long ulPinNumber, ulVoltage;
unsigned long ulDeviceID;
```

```
ulDeviceID = 0x01;           // Use the Device ID which is set to 1
```

```
// This is to set 5 volts on Pin 12.
```

```
ulVoltage = 5000; // ulVoltage is in millivolts
```

```
ulPinNumber = 12;
```

```
ulErrorNumCode = PassThruSetProgrammingVoltage (ulDeviceID, ulPinNumber, ulVoltage);
```

7.12 **Function Name:** PassThruReadVersion

7.12.1 **Description:**

This function reads in the firmware, the DLL, and the API versions to the user.

Refer to SAE J2534 Section 7.2.12.

7.12.1.1 **C/C++ Prototype**

```
extern "C" long WINAPI PassThruReadVersion
{
    unsigned long ulDeviceID,
    char *pchFirmwareVersion,
    char *pchDllVersion,
    char *pchApiVersion
}
```

7.12.1.2 **Parameter Description**

Refer to SAE J2534 Section 7.2.12.2.

7.12.1.3 **Return Values**

Refer to SAE J2534 Section 7.2.12.3.

7.12.2 **Example:**

```
unsigned long ulErrorNumCode;
char          chFirmwareVersion[80], chDllVersion[80], chApiVersion[80];
// Each char variables has a max. length of 80 char.
unsigned long ulDeviceID;

ulDeviceID = 0x01;           // Use the Device ID which is set to 1

// This is to read the versions to the user.
ulErrorNumCode = PassThruReadVersion (ulDeviceID, chFirmwareVersion, chDllVersion,
                                     chApiVersion);
```

7.13 Function Name: PassThruGetLastError

7.13.1 Description:

This function returns the description of the last error detected to the user.

Refer to SAE J2534 Section 7.2.13.

7.13.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruGetLastError
{
    char *pchErrorDescription
}
```

7.13.1.2 Parameter Description

Refer to SAE J2534 Section 7.2.13.2.

7.13.1.3 Return Values

Refer to SAE J2534 Section 7.2.13.3.

7.13.2 Example:

```
unsigned long ulErrorNumCode;
char chErrorDescription[80]; // This has a max. length of 80 char.
```

```
// This is to get the last error string to the user.
```

```
ulErrorNumCode = PassThruGetLastError (chErrorDescription);
```

7.14 Function Name: PassThruIoctl

7.14.1 Description:

This function reads and writes the IOCTL parameters to and from the device.
Refer to SAE J2534 Section 7.2.14.

7.14.1.1 C/C++ Prototype

```
extern "C" long WINAPI PassThruIoctl
{
    unsigned long ulChannelID,
    unsigned long ulIoctlID,
    void *pInput,
    void *pOutput
}
```

7.14.1.2 Parameter Description

ulChannelID:

Refer to SAE J2534 Section 7.2.14.2.

ulIoctlID:

Value(s)	Description	Definition
0x01 – 0xFFFF	Refer to SAE J2534 Section 7.2.14.3	Refer to SAE J2534 Section 7.2.14.3

pInput:

Refer to SAE J2534 Section 7.3.

pOutput:

Refer to SAE J2534 Section 7.3.

7.14.1.3 Return Values

Refer to SAE J2534 Section 7.2.14.4.

7.14.2 Example:

```
unsigned long    ulErrorNumCode;
unsigned long    ulChannelID, ulIoctlID;
SCONFIG_LIST    strucSconfigList;

// This is to set the baud rate to 250,000.
strucSconfigList.pConfigPtr = (SCONFIG *) malloc (sizeof (SCONFIG));
strucSconfigList.ulNumOfParams = 1;
strucSconfigList.pConfigPtr->ulParameter = 0x01; // This is for DATA_RATE parameter
strucSconfigList.pConfigPtr->ulValue = 250000; // This is to set the baud rate to 250,000
ulIoctlID = 0x02; // This is for SET_CONFIG Ioctl

// This is to set the IOCTL config using the ulChannelID that was returned when connecting to
// that protocol.
ulErrorNumCode = PassThruIoctl (ulChannelID, ulIoctlID, &strucSconfigList, NULL);

// Free the memory
```

free(strucSconfigList. pConfigPtr);

8 Document Revision History

Document Version	Date	Owner	Description
0.1	November 26, 2003	Sanjay	Initial Revision
0.2	December 5, 2003	Ming	Changed all enumerated variable types and its values to unsigned long. All example codes are in italic. All variable declarations in the code are lined up. Removed all pound defines and used the number instead. Section 2, there is website information in obtaining the reference document. Section 6, the "ConfigApplication" value now has the full path.
1.0	March 18,2005	Sanjay	Section 7, Added two new functions: PassThruStart (7.1) and PassThruStop (7.2) and renumbered the remaining functions.