

Introduction:

This document provides the description about the J2534 PassThru APIs. We support J2534 standard based APIs for our device, which are widely used in automotive domain.

User Data Types

1) struct RXStatus:

```
struct RXStatus
{
    unsigned long busStatus_u8;
    unsigned long txErrorCounter_u8;
    unsigned long rxErrorCounter_u8;
};
```

Data Members:

- **busStatus_u8:** This is a byte value which specifies the CAN bus status.
- **txErrorCounter_u8:** This is byte value which indicates the number of TX error frames.
- **rxErrorCounter_u8:** This is byte value which indicates the number of RX error frames.

2) struct PASSTHRU_MSG:

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

Data Members:

- **Timestamp:** This is a unsigned long variable which holds Timestamp of received Rx CAN frame.
- **Data:** This is a unsigned long variable which contains CAN data of received Rx CAN frame.

- **ExtraDataIndex:** This is a unsigned long variable which indicates the type of CAN frame, ExtraDataIndex =0 for standard frame and 1 for extended frame.
- **ProtocolID:** This is a unsigned long variable which contains message ID of received CAN frame.
- **DataSize:** This is a unsigned long variable which contains data length count (DataSize) of Rx/Tx CAN frame.
- **TxFlags:** This is a unsigned long variable which contains 1 if CAN frame is a RTR type.

Note: Channel Number is always 0 as there is only one CAN channel in SBUSCAN currently.

3) struct PassTHRU_CANFrame:

```
struct PASSTHRU_CANFrame
{
    PASSTHRU_MSG msg;
    RXStatus chipstate;
};
```

Data Members:

- **Msg:** This is a structure which contains all CAN data information.
- **chipState:** This is a structure which contains information about CAN channel state. This data member also contains txerrorFrame and rxerrorFrame count.

4) enum CAN_BAUD_RATE_E:

This enumeration lists all supported CAN baud rates by SBUSCAN.

Enumerated List:

- **CAN_BAUD_125 = 0:** This Enum value will indicate the CAN baud rate 125kbps.
- **CAN_BAUD_250 = 1:** This Enum value will indicate the CAN baud rate 250kbps.
- **CAN_BAUD_500 = 2:** This Enum value will indicate the CAN baud rate 500kbps.
- **CAN_BAUD_1000 = 3:** This Enum value will indicate the CAN baud rate 1000kbps.

APIs

Following table provides the list of APIs defined by SAE J2534 for CAN interface device.

SI #	Function	Description	Supported by SBUSCAN driver
1	PassThruConnect	Establish a connection with a protocol channel.	Yes
2	PassThruDisconnect	Terminate a connection with a protocol channel.	Yes
3	PassThruReadMsgs	Read message(s) from a protocol channel.	Yes
4	PassThruWriteMsgs	Write message(s) to a protocol channel.	Yes
5	PassThruStartPeriodicMsg	Start sending a message at a specified time interval on a protocol channel.	No
6	PassThruStopPeriodicMsg	Stop a periodic message.	No
7	PassThruStartMsgFilter	Start filtering incoming messages on a protocol channel.	Yes
8	PassThruStopMsgFilter	Stops filtering incoming messages on a protocol channel.	Yes
9	PassThruSetProgrammingVoltage	Set a programming voltage on a specific pin.	No
10	PassThruReadVersion	Reads the version information for the DLL and API.	No
11	PassThruGetLastError	Gets the text description of the last error.	No
12	PassThruIoctl	General I/O control functions for reading and writing protocol configuration parameters (e.g. initialization, baud rates, programming voltages, etc.).	No
13	PassThruOpen	This is used to select the SBUSCAN device	Yes
14	PassThruClose	This is used to close the device interface	Yes

Specification of each J2534 API:

1) PassThruOpen

long PassThruOpen (uint8_t pName[100][50], uint8_t *pCount)

This API is used to get list of serial number string of all devices whose VID and PID matches with the SBUSCAN. This is used when more than one SBUSCAN is connected to the same PC/Laptop.

API Parameter:

- **PName [10][20]:** This is 2-D array which contains list of serial numbers of devices whose VID and PID is matched with the SBUSCAN.
- **pCount:** This is a pointer which contains the number of serial numbers stored in **SBUSCAN_SERIAL_NUM** parameter.

Return:

Returns the status as Success or failure.

Preconditions:

None.

2) PassThruConnect

long PassThruConnect(uint8_t *DeviceID, unsigned long ProtocolID, unsigned long* pChannelID, CAN_BAUD_RATE_E Baudrate,unsigned long Flags)

This API is used to connect to a specific SUBCAN device. This API will search for target device in list of all connected USB devices, once found the USB handle is initialized, a receive call back is set and a start communication command is sent to SBUSCAN device.

API Parameters:

- ***DeviceID:** This is a pointer which stores Selected SBUSCAN device Serial number to interface with selected device.
- **ProtocolID – Redundant**
- **ChannelId – Redundant**
- **Baudrate – This is to select the appropriate baud rate**
- **Flags -**

Return:

Returns the status as Success or failure.

Preconditions:

Before calling this function call PassThruOpen () API.

3) PassThruDisconnect

long PassThruDisconnect(unsigned long ChannelID)

This API is used to disconnect the connected SBUSCAN device. When called a Stop Communication command is sent to SBUSCAN device and the communication is halted.

API Parameters:

DeviceId

Return:

Returns the status as Success or failure.

Preconditions:

PassThruConnect () API shall be called before calling this API.

4) PassThruReadMsgs

long PassThruReadMsgs(unsigned long ChannelID,pMsg pCallback, unsigned long *pNumMsgs, unsigned long Timeout)

This API is used to read CAN frames received from SBUSCAN.

API Parameters:

- **pCallback:** This parameter carries the address of the Api to be called on the reception of the message frame.
- **ChannelID:** This parameter is redundant.
- **pNumMsgs:** This parameter is redundant.
- **Timeout :** This parameter is redundant.

Return:

A long value will be returned which will describe the CAN Frames received successfully.

Preconditions:

PassThruConnect () API shall be called before calling this API.

5) PassThruStartMsgFilter

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

This API is used to filter the message received based on the mask and pattern type.

API Parameters:

- **ChannelID:** This parameter is redundant.
- **FilterType:** This parameter is used to allow the pattern type or block the pattern type

Note: Allowed Filter types are 1) PASS_FILTER 2)BLOCK FILTER

- **pMaskMsg :** This parameter specifies Mask id.
- **pPatternMsg:** This parameter specifies the Pateern id
- **pFlowControlMsg:** This parameter is redundant.
- **PMsgID:** This parameter is redundant

Return:

A long value will be returned which will describe the CAN Frames received successfully.

Preconditions:

None

PassThruConnect () API shall be called before calling this API.

6) PassThruStopMsgFilter

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

This API is used to stop the reception of a particular message id.

API Parameters:

- **ChannelID:** This parameter is redundant.
- **msgID:** This parameter represents the message id to be stopped.

Return:

A long value will be returned which will describe the CAN Frames received successfully.

Preconditions:

None

7) PassThruWriteMsgs

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

This API is used to transmit a CAN frame from SBUSCAN device. Upon calling this API a tx Data Frame is sent to SUBSCAN device. Details of CAN frames are described in API parameters.

API Parameters:

- **pMsg:** This parameter is a structure which contains all information needed to transmit a CAN frame
- **ChannelID:** This parameter is redundant.
- **pNumMsg:** This parameter is redundant.
- **Timeout :** This parameter is redundant.

A long value will be returned which will describe the CAN Frames received successfully.

Preconditions:

PassThruConnect () API shall be called before calling this API.

8) PassThruClose

long Close(unsigned long ChannelID)

This API is used to disconnect the connected SBUSCAN device. When called a Stop Communication command is sent to SBUSCAN device and USB device handler is uninitialized.

API Parameters:

DeviceId

Return:

Returns the status as Success or failure.

Preconditions:

PassThruConnect () API shall be called before calling this API.

SAMPLE TEST CODE

```
#include <iostream>
#include <fstream>
#include <stdint.h>
#include <stdio.h>
#include <unistd.h>
#include "j2534.h"

using namespace std;

uint8_t pName[100][50];
uint8_t pCount;
struct PASSTHRU_MSG txMsg;
pMsg pCallback;
CAN_BAUD_RATE_E Baudrate;
unsigned long Flags = 0; // Flags = 0[start] & 1[stop]
unsigned long* pNumMsgs;
unsigned long ChannelID;
unsigned long Timeout = 0;
PASSTHRU_MSG pMaskMsg;
PASSTHRU_MSG pPatternMsg;
unsigned long pMsgID;

/* Example function to observe recieved CAN Frames */

void Test_Print_CAN_Rx_Data(PASSTHRU_CANFrame CANFrameObj)
{
    printf("\n");
    printf("%x\t%x\t%x\t%x\t%x\t%x\t%x\t%x\t%x\t%x\n",
        CANFrameObj.msg.Timestamp,
        CANFrameObj.msg.ProtocolID,
        CANFrameObj.msg.DataSize,
        CANFrameObj.msg.Data[0],
        CANFrameObj.msg.Data[1],
        CANFrameObj.msg.Data[2],
        CANFrameObj.msg.Data[3],
        CANFrameObj.msg.Data[4],
        CANFrameObj.msg.Data[5],
        CANFrameObj.msg.Data[6],
        CANFrameObj.msg.Data[7]);

    printf("\n");
    CANFrameObj.msg.Data[0] = 0;
    CANFrameObj.msg.Data[1] = 0;
    CANFrameObj.msg.Data[2] = 0;
    CANFrameObj.msg.Data[3] = 0;
    CANFrameObj.msg.Data[4] = 0;
```

```
CANFrameObj.msg.Data[5] = 0;
CANFrameObj.msg.Data[6] = 0;
CANFrameObj.msg.Data[7] = 0;

}

/* Example function to transmit CAN Frames */

void Test_TransmitCANFrame()
{
    unsigned long Timeout;
    static uint8_t temp = 0;

    txMsg.ProtocolID = 0x26;
    txMsg.DataSize = 8;
    txMsg.Data[0] = temp++;
    txMsg.Data[1] = 2;
    txMsg.Data[2] = 3;
    txMsg.Data[3] = 4;
    txMsg.Data[4] = 5;
    txMsg.Data[5] = 6;
    txMsg.Data[6] = 7;
    txMsg.Data[7] = 8;
    PassThruWriteMsgs(ChannelID,txMsg,pNumMsgs,Timeout);
}

int main()
{
    ssize_t i;
    ssize_t DeviceID;

    unsigned long ProtocolID = 0;
    unsigned long* pChannelID;

    PassThruOpen(pName,&pCount);
    printf("%d Devs Connected to this machine\n\n",pCount);

    if(pCount == 0)
    {
        printf("List is Empty,Not possible to select device\n\n");
    }
    else
    {
        printf("Please selcet one\n\n");
    }

    for(i = 0; i < pCount;i++)
```

```
{
    printf("%d: %s\n\n",i +1,pName[i]);
}
//scanf("%d",&DeviceID);
DeviceID = 1;
printf("Selected Device is %s\n\n",pName[DeviceID-1]);

if(pName[DeviceID-1] != 0)
{
    PassThruConnect(pName[DeviceID-
1],ProtocolID,pChannelID,CAN_BAUD_500,Flags);
}
else
{
    printf("Device connection failed\n\n");
}

    pCallback = Test_Print_CAN_Rx_Data;
    pMaskMsg.ProtocolID = 0x7FC;
    pPatternMsg.ProtocolID = 0x7E0;

/*Setting the filter to receive the message with the message id as 0x7E0,
0x7E1, 0x7E2*/

    PassThruStartMsgFilter(0, PASS_FILTER, &pMaskMsg, &pPatternMsg,
nullptr, &pMsgID);
    /*Blocking the filter from receiving the Message with id 0x7E2*/
    PassThruStopMsgFilter(0, 0x7e2);
    PassThruReadMsgs(ChannelID,pCallback,pNumMsgs,Timeout);

    while (1)
    {
        Test_TransmitCANFrame();

        sleep(1);
    }

    sleep(1000);

    PassThruDisconnect(ChannelID);
    PassThruClose(DeviceID);
    return 0;
}
```