

CAN Gateway – application note

INDEX

INDEX	2
Revision History	3
Abbreviation	3
1 Introduction	4
2 Using the HDJ2534 library	4
2.1 Introduction to J2534 API	5
2.2 Opening the device	6
2.3 Closing the device	7
2.4 Connecting to a physical channel	8
2.5 Disconnecting from a physical channel	10
2.6 Activating a message filter	11
2.7 Disabling a message filters	14
2.8 Sending of CAN messages	15
2.9 Receiving of CAN messages	17
2.10 Activating a periodic message	19
2.11 Disabling a periodic message	20
2.12 Retrieving text description of the last error	21
2.13 Executing IOCTL commands	22
2.14 Getting additional information about status of the device	24
2.15 Resetting the device	25
3 The HDJ2534 library in multithreaded applications	26
4 An example code for Linux OS	26
4.1 Prerequisites	26
4.2 Configuration instructions for the example	26
4.3 Description of the example	26
5 Related documents	28

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

Revision History

Created			Approved		Description
Rev	Date	By	Date	By	
0.1	14.02.2013	Łukasz Madej			- Draft version of the document with description of example project has been created
0.2	06.03.2014	Łukasz Madej			- The document has been turned into complete application note
1	13.03.2014	Łukasz Madej	14.03.2014	Joakim Emanuelsson Jakub Kwiatkowski	- Description of <i>PassThruGetStatus()</i> function has been added - Description of <i>PassThruReset()</i> function has been added - Doxygen documentation has been improved

Abbreviation

Abbreviation	Description
API	Application Programming Interface
CAN	Controller Area network
Hatteland	Hatteland Display AS
HDJ2534	Hatteland Display API library to accessing CAN Gateway from user application
OS	Operating System

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

1 Introduction

The CAN Module produced by Hatteland Display is delivered with standard SAE J2534 API [R1]. This document familiarizes with the API and introduces some examples. Furthermore an example C++ project for Linux OS which uses the API is provided with this document. The project shows how to initialize CAN device and how to properly exchange simple messages.

ATTENTION: The example should not be used in a direct form as a base of industrial applications.

The application note consists of:

- This document,
- Example application v1.2 (cangw_example.zip),
- Doxygen documentation for the example.

2 Using the HDJ2534 library

In Figure 1 role of the HDJ2534 library in the system with CAN Gateway is shown. The library is used as interface to access CAN Gateway device. Further parts of this chapter describe functions exported by the API.

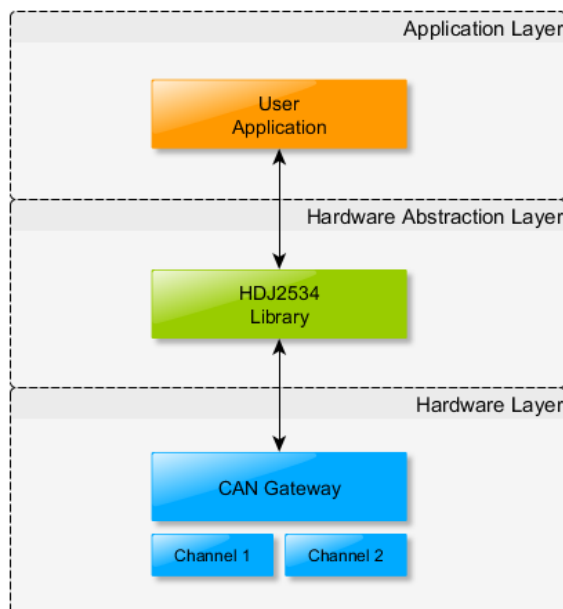


Figure 1. Role of the HDJ2534 library in a system with CAN Gateway

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.1 Introduction to J2534 API

Library for accessing CAN Gateway uses J2534 API [R1] as its interface. In Table 1 a complete set of functions of the API is shown.

Table 1. J2534 API's function set

J2534 function	Brief description
<i>PassThruOpen()</i>	Opens CAN Gateway device
<i>PassThruClose()</i>	Closes CAN Gateway device
<i>PassThruConnect()</i>	Connects a logical channel with specified protocol to a physical channel
<i>PassThruDisconnect()</i>	Disconnects a logical channel from a physical channel
<i>PassThruReadMsgs()</i>	Reads specified number of messages from a logical channel
<i>PassThruWriteMsgs()</i>	Writes specified number of messages to a logical channel
<i>PassThruStartPeriodicMsg()</i>	Activates repetitively transmitted message
<i>PassThruStopPeriodicMsg()</i>	Removes periodic message from logical channel
<i>PassThruStartMsgFilter()</i>	Adds pass or block message filter to selected logical channel
<i>PassThruStopMsgFilter()</i>	Removes message filter from selected logical channel
<i>PassThruGetLastError()</i>	Returns the latest API error as a text message
<i>PassThruIoctl()</i>	Executes general purpose I/O control command
<i>PassThruGetStatus()</i>	Obtains actual value of CAN Gateway's status register (should not be used in standard implementations)
<i>PassThruReset()</i>	Forces CAN Gateway's reset (should not be used in standard implementations)

In order to exchange CAN messages a minimum set of J2534 functions must be called in following order:

- 1) Open CAN device with *PassThruOpen()* function (refer to chapter 2.2),
- 2) Connect a logical channel to a physical channel with *PassThruConnect()* function (refer to chapter 2.4),
- 3) In order to receive messages, add pass message filter to a logical channel by calling *PassThruStartMsgFilter()* (refer to chapter 2.6),
- 4) Exchange messages with *PassThruWriteMsgs()* and *PassThruReadMsgs()* functions (refer to chapter 2.8 and 2.9 respectively),
- 5) Disconnect a logical channel from a physical channel with *PassThruDisconnect()* function (refer to chapter 2.5),
- 6) Close CAN device by calling *PassThruClose()* function (refer to chapter 2.3).

Above list shows minimal implementation needed to assure proper data exchange via CAN Gateway device. For more detailed description refer to further parts of this document.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.2 Opening the device

Before any transaction with CAN Device, the device has to be opened with *PassThruOpen()* function. Otherwise every API function will return an error.

An example call of a *PassThruOpen(void * pName, unsigned long * pDeviceID)* function is shown in Listing 1. Arguments passed to the function are summarized in Table 2.

Table 2. Summary of *PassThruOpen()* function's arguments

Argument	Description
<i>void * pName</i>	Shall always be <i>NULL</i> .
<i>unsigned long * pDeviceID</i>	Pointer to a variable to store an ID of the device calculated by the function. This ID is needed during further communication with CAN Gateway.

Listing 1. An example usage of *PassThruOpen()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idDevice; // ID of the CAN device

// Opens CAN Gateway device
J2534_ERROR_CODE passThruErr = PassThruOpen(NULL, &idDevice);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.3 Closing the device

Before exiting, an application shall close previously opened connection with CAN Gateway with *PassThruClose()* function.

An example call of a *PassThruClose(unsigned long DeviceID)* function is shown in Listing 2. Argument passed to the function is summarized in Table 3.

Table 3. Summary of *PassThruClose()* function's argument

Argument	Description
<i>unsigned long DeviceID</i>	ID of the CAN device calculated by <i>PassThruOpen()</i> function.

Listing 2. An example usage of *PassThruClose()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idDevice; // ID of the CAN device calculated by PassThruOpen() function

// Closes a CAN Gateway device
J2534_ERROR_CODE passThruErr = PassThruClose(idDevice);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- Before calling *PassThruClose()* function all active logical channels should be closed with *PassThruDisconnect()* function (refer to chapter 2.5).

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.4 Connecting to a physical channel

In order to send or receive CAN messages a logical channel have to be initialized and connected to a physical channel of CAN Gateway (physical port CAN1 or CAN2). For this purpose a *PassThruConnect()* function shall be used.

An example call of a *PassThruConnect(unsigned long DeviceID, J2534_PROTOCOL ProtocolID, unsigned long Flags, unsigned long BaudRate, unsigned long * pChannelID)* function is shown in Listing 3. Arguments passed to the function are summarized in Table 4. For more detailed description of the function refer to [R1].

Table 4. Summary of *PassThruConnect()* function's arguments

Argument	Description
<i>unsigned long DeviceID</i>	ID of the CAN device calculated by <i>PassThruOpen()</i> function.
<i>J2534_PROTOCOL ProtocolID</i>	Protocol to be used for communication over a logical channel. Current version of HDJ2534 library supports RAW CAN protocol only (CAN).
<i>unsigned long Flags</i>	Passes values of additional flags to the function. If logical channel connects to physical channel 1, a value equal 0 shall be used. If logical channel connects to physical channel 2, a value equal PHYSICAL_CHANNEL shall be used.
<i>unsigned long BaudRate</i>	Selects baud rate for physical channel. Available baud rate values are: 125000 (125 kbps), 250000 (250 kbps), 500000 (500 kbps), 1000000 (1 Mbps).
<i>unsigned long * pChannelID</i>	Pointer to a variable to store an ID of the logical channel to be used during further communication with the CAN Gateway.

Listing 3. An example usage of *PassThruConnect()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idDevice; // ID of the CAN device calculated by PassThruOpen() function
unsigned long idCh1; // ID of the logical channel 1
unsigned long idCh2; // ID of the logical channel 2
static const unsigned long baudrateCh1 = 125000; // Baud rate for channel = 125 kbps
static const unsigned long baudrateCh2 = 1000000; // Baud rate for channel = 1 Mbps
static const unsigned long flagsCh1 = 0; // Value of flags for physical channel 1
static const unsigned long flagsCh2 = PHYSICAL_CHANNEL; // Value of flags for physical channel 2
J2534_ERROR_CODE passThruErr = STATUS_NOERROR; // Result of PassThru functions

// Connects a RAW CAN logical channel 1 to a physical channel 1
passThruErr = PassThruConnect(idDevice, CAN, flagsCh1, baudrateCh1, &idCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
// Connects a RAW CAN logical channel 2 to a physical channel 2
passThruErr = PassThruConnect(idDevice, CAN, flagsCh2, baudrateCh2, &idCh2);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

Additional notes:

- If two logical channels connect to the same physical channel (e.g. physical channel 1), each logical channel receives the same messages sent to this physical channel.
- If one logical channel connects to a physical channel with specified baud rate and the other logical channel connects to the same physical channel with different baud rate, a physical channel changes its baud rate to passed with last call of *PassThruConnect()* function.
- There is a limit of 10 logical channels for both physical channel.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.5 Disconnecting from a physical channel

If application closes it shall, before calling *PassThruClose()* function, disconnect all connected logical channels with *PassThruDisconnect()* function.

An example call of a *PassThruDisconnect(unsigned long ChannelID)* function is shown in Listing 4. Argument passed to the function is summarized in Table 5. For more detailed description of the function refer to [R1].

Table 5. Summary of *PassThruDisconnect()* function's argument

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel to be disconnected calculated by <i>PassThruConnect()</i> function.

Listing 4. An example usage of *PassThruDisconnect()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function

// Disconnects a logical channel 1 from a physical channel
J2534_ERROR_CODE passThruErr = PassThruDisconnect(idCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- If logical channel disconnects, all message filters related to this channel are disabled automatically (refer to chapter 2.7).
- If logical channel disconnects, all periodic messages related to this channel are disabled automatically (refer to chapter 2.11).

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.6 Activating a message filter

In order to receive messages via previously connected logical channel (refer to chapter 2.4) a message filter has to be activated on this channel with *PassThruStartMsgFilter()* function. There are two types of filters: pass filter and block filter. A pass filter will pass messages matching to the filter and a block filter will block messages matching to the filter. According to [R1] message filters works in a following way:

- 1) Received message is masked by a PassThru filter mask (logical AND operation),
- 2) Result of AND operation is compared with PassThru filter pattern,
- 3) If result of AND operation differs from PassThru filter pattern the filter passes or blocks received message (filter action depends of filter type – pass filter or block filter).

Therefore a pseudo truth table for pass filter and block filter is as shown in table below.

Table 6. Pseudo truth table for pass filter and block filter

Filter Mask Bit (FMB)	Received Message Bit (RMB)	FMB AND RMP	Filter Pattern Bit (FPB)	Pass filter action	Block filter action
0	0	0	0	Pass	Block
0	1	0	0	Pass	Block
0	0	0	1	Block	Pass
0	1	0	1	Block	Pass
1	0	0	0	Pass	Block
1	1	1	0	Block	Pass
1	0	0	1	Block	Pass
1	1	1	1	Pass	Block

Since message filters are implemented in the CAN hardware, they do not affect user application's performance.

An example calls of a *PassThruStartMsgFilter(unsigned long ChannelID, J2534_FILTER FilterType, PASSTHRU_MSG * pMaskMsg, PASSTHRU_MSG * pPatternMsg, PASSTHRU_MSG * pFlowControlMsg, unsigned long * pFilterID)* function are shown in Listing 5 and Listing 6. Arguments passed to the function are summarized in Table 7. Summary of *PassThruStartMsgFilter()* function's arguments. For more detailed description of the function refer to [R1].

Table 7. Summary of *PassThruStartMsgFilter()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel to be affected by the filter calculated by <i>PassThruConnect()</i> function.
<i>J2534_FILTER FilterType</i>	Type of message filter. Shall be equal PASS_FILTER or BLOCK_FILTER.
<i>PASSTHRU_MSG * pMaskMsg</i>	Pointer to a message structure which contains mask data. The mask message is applied by the CAN device to every receive message. 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.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

Argument	Description
<i>PASSTHRU_MSG * pPatternMsg</i>	<p>Pointer to a message structure which contains the user application's pattern data. The CAN device compares the pattern message to the receive message after it is ANDed with the mask message.</p> <p>For a PASS_FILTER if the pattern message matches the result of received message ANDed with the mask message, then the CAN device adds that message to its receive message queue. Otherwise that receive message will be ignored.</p> <p>For a BLOCK_FILTER if the Pattern message matches the result of received message ANDed with the mask message, then the CAN device ignores that message. Otherwise that receive message will be added to the CAN device receive message queue.</p>
<i>PASSTHRU_MSG * pFlowControlMsg</i>	Shall always be <i>NULL</i> .
<i>unsigned long * pFilterID</i>	Pointer to a variable to store an ID of the filter to be used during further communication with the CAN Gateway.

Listing 5. An example usage of *PassThruStartMsgFilter()* function to pass all messages

```
#include <cstring>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idFilterCh1; // ID of message filter for logical channel 1
PASSTHRU_MSG maskPassThruMsg; // Message structure with filter mask data
PASSTHRU_MSG patternPassThruMsg; // Message structure with filter pattern data
J2534_ConnectFlags filterFlags; // Additional flags for message structures

memset(&maskPassThruMsg, 0, sizeof(PASSTHRU_MSG));
memset(&patternPassThruMsg, 0, sizeof(PASSTHRU_MSG));
maskPassThruMsg.ProtocolID = CAN; // // Masks protocol ID
patternPassThruMsg.ProtocolID = CAN; // Will pass RAW CAN messages only
filterFlags.bits.Can29BitId = 0; // Do not mask ID type (11-bit / 29-bit)
maskPassThruMsg.TxFlags = filterFlags.value;
filterFlags.bits.Can29BitId = 0; // Will pass messages with every ID type (11-bit and 29-bit)
patternPassThruMsg.TxFlags = filterFlags.value;
// Activates the pass filter
J2534_ERROR_CODE passThruErr = PassThruStartMsgFilter(idCh1, PASS_FILTER, &maskPassThruMsg,
&patternPassThruMsg, NULL, &idFilterCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Listing 6. An example usage of *PassThruStartMsgFilter()* function to pass messages with 11-bit ID only

```
#include <cstring>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idFilterCh1; // ID of message filter for logical channel 1
PASSTHRU_MSG maskPassThruMsg; // Message structure with filter mask data
PASSTHRU_MSG patternPassThruMsg; // Message structure with filter pattern data
J2534_ConnectFlags filterFlags; // Additional flags for message structures

memset(&maskPassThruMsg, 0, sizeof(PASSTHRU_MSG));
memset(&patternPassThruMsg, 0, sizeof(PASSTHRU_MSG));
maskPassThruMsg.ProtocolID = CAN; // Masks protocol ID
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

```
patternPassThruMsg.ProtocolID = CAN; // Will pass RAW CAN messages only
filterFlags.bits.Can29BitId = 1; // Masks ID type (11-bit / 29-bit)
maskPassThruMsg.TxFlags = filterFlags.value;
filterFlags.bits.Can29BitId = 0; // Will pass messages with 11-bit ID type only
patternPassThruMsg.TxFlags = filterFlags.value;
// Activates the pass filter
J2534_ERROR_CODE passThruErr = PassThruStartMsgFilter(idCh1, PASS_FILTER, &maskPassThruMsg,
&patternPassThruMsg, NULL, &idFilterCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- There is a limit of 10 message filters for each logical channel.
- Message filters do not filter data fields of CAN messages (bytes from 0 to 11 of CAN data frame).

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.7 Disabling a message filters

Previously activated message filter can be disabled with *PassThruStopMsgFilter()* function.

An example call of a *PassThruStopMsgFilter(unsigned long ChannelID, unsigned long FilterID)* function is shown in Listing 7. Arguments passed to the function are summarized in Table 8. For more detailed description of the function refer to [R1].

Table 8. Summary of *PassThruStopMsgFilter()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel with filter to be disabled calculated by <i>PassThruConnect()</i> function.
<i>unsigned long FilterID</i>	ID of message filter to be disabled calculated by <i>PassThruStartMsgFilter()</i> function

Listing 7. An example usage of *PassThruStopMsgFilter()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idFilterCh1; // ID of message filter calculated by PassThruStartMsgFilter() function

// Disables a message filter
J2534_ERROR_CODE passThruErr = PassThruStopMsgFilter(idCh1, idFilterCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- If logical channel disconnects, all message filters related to this channel are disabled automatically.
- All message filters related to selected channel can be disabled with an IOCTL command (refer to chapter 2.13).

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.8 Sending of CAN messages

The HDJ2534 library sends messages with *PassThruWriteMsgs()* function. There is a possibility to send few messages within one call of the function. Furthermore the function provides timeout mechanism (if message is not sent via CAN network during timeout interval, an error occurs).

An example call of a *PassThruWriteMsgs(unsigned long ChannelID, PASSTHRU_MSG * pMsg, unsigned long * pNumMsgs, unsigned long Timeout)* function is shown in below listings. Arguments passed to the function are summarized in Table 9. For more detailed description of the function refer to [R1].

Table 9. Summary of *PassThruWriteMsgs()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel to be used for sending of the messages calculated by <i>PassThruConnect()</i> function.
<i>PASSTHRU_MSG * pMsg</i>	Pointer to a message(s) generated by user application.
<i>unsigned long * pNumMsgs</i>	Pointer to a variable which contains number of PASSTHRU_MSG structures to send. On function completion this variable will contain the actual number of messages sent to CAN network. The transmitted number of messages may be less than the number requested by the application.
<i>unsigned long Timeout</i>	Timeout interval (in milliseconds) to wait for transmit completion. A value of zero instructs the library to queue as many transmit messages as possible and return immediately. A nonzero timeout value instructs the library to wait for the timeout interval to expire before returning. The library will not wait the entire timeout interval if an error occurs or specified number of messages is sent.

Listing 8. An example usage of *PassThruWriteMsgs()* function to send one message with inactive timeout

```
#include <cstring>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long msgNum; // Number of messages to send
unsigned int canId; // ID of CAN message
PASSTHRU_MSG txPassThruMsg; // Message structure
J2534_TxFlags txFlags; // Additional flags for message

// Generates a CAN message
memset(&txPassThruMsg, 0, sizeof(PASSTHRU_MSG));
txFlags.bits.Can29BitId = 1; // 1 for 29-bit CAN ID, 0 for 11-bit CAN ID
txPassThruMsg.TxFlags = txFlags.value;
txPassThruMsg.ProtocolID = CAN; // Raw Can protocol will be used
idCan = 0x0000ABCD;
txPassThruMsg.Data[0] = static_cast<unsigned char>(idCan >> 24);
txPassThruMsg.Data[1] = static_cast<unsigned char>(idCan >> 16);
txPassThruMsg.Data[2] = static_cast<unsigned char>(idCan >> 8);
txPassThruMsg.Data[3] = static_cast<unsigned char>(idCan >> 0);
txPassThruMsg.Data[4] = 'M';
txPassThruMsg.Data[5] = 'S';
txPassThruMsg.Data[6] = 'G';
txPassThruMsg.DataSize = 7; // 4 bytes for a CAN ID + 3 bytes for a message
msgNum = 1; // One message will be sent
// Sends message to CAN Gateway
J2534_ERROR_CODE passThruErr = PassThruWriteMsgs(idCh1, &txPassThruMsg, &msgNum, 0);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

```
}
```

Listing 9. An example usage of *PassThruWriteMsgs()* function to send 3 messages with active timeout

```
#include <iostream>
#include <cstring>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long msgNum; // Number of messages to send
unsigned int canId; // ID of CAN message
PASSTHRU_MSG txPassThruMsg[3]; // Array of message structures (three messages)
J2534_TxFlags txFlags; // Additional flags for message
static const unsigned long timeout = 100; // Timeout for message send = 100 ms

// Generates a CAN message
memset(&txPassThruMsg, 0, sizeof(PASSTHRU_MSG));
txFlags.bits.Can29BitId = 1; // 1 for 29-bit CAN ID, 0 for 11-bit CAN ID
txPassThruMsg[0].TxFlags = txFlags.value;
txPassThruMsg[0].ProtocolID = CAN; // Raw Can protocol will be used
idCan = 0x0000ABCD;
txPassThruMsg[0].Data[0] = static_cast<unsigned char>(idCan >> 24);
txPassThruMsg[0].Data[1] = static_cast<unsigned char>(idCan >> 16);
txPassThruMsg[0].Data[2] = static_cast<unsigned char>(idCan >> 8);
txPassThruMsg[0].Data[3] = static_cast<unsigned char>(idCan >> 0);
txPassThruMsg[0].Data[4] = 'M';
txPassThruMsg[0].Data[5] = 'S';
txPassThruMsg[0].Data[6] = 'G';
txPassThruMsg[0].Data[7] = '0';
txPassThruMsg[0].DataSize = 8; // 4 bytes for a CAN ID + 4 bytes for a message

txPassThruMsg[1] = txPassThruMsg[0]; // Message 1 is a copy of message 0
txPassThruMsg[2] = txPassThruMsg[0]; // Message 2 is a copy of message 0
txPassThruMsg[1].Data[7] = '1'; // Updates message number for message 1
txPassThruMsg[2].Data[7] = '2'; // Updates message number for message 2

msgNum = 3; // Three message will be sent
// Sends messages to CAN Gateway
J2534_ERROR_CODE passThruErr = PassThruWriteMsgs(idCh1, &txPassThruMsg, &msgNum, timeout);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
    std::cout << "Sent messages: " << msgNum << std::endl;
    if (ERR_TIMEOUT == passThruErr)
    {
        std::cout << "Timeout occurred!" << std::endl;
    }
}
}
```

Additional notes:

- The library will not wait the entire timeout interval if an error occurs or the specified number of messages is sent.
- *PassThruWriteMsgs()* function can blocks application's context for timeout interval.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.9 Receiving of CAN messages

The HDJ2534 library receives messages with *PassThruReadMsgs()* function. There is a possibility to receive few messages within one call of the function. Furthermore the function provides timeout mechanism (if specified number of messages is not received from network during timeout interval, an error occurs).

An example call of a *PassThruReadMsgs(unsigned long ChannelID, PASSTHRU_MSG * pMsg, unsigned long * pNumMsgs, unsigned long Timeout)* function is shown in below listings. Arguments passed to the function are summarized in Table 10. For more detailed description of the function refer to [R1].

Table 10. Summary of *PassThruReadMsgs()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel to be used for receiving of the messages calculated by <i>PassThruConnect()</i> function.
<i>PASSTHRU_MSG * pMsg</i>	Pointer to a message structure where the library will write received message(s).
<i>unsigned long * pNumMsgs</i>	Pointer to a variable which contains number of PASSTHRU_MSG structures allocated for received messages. It is the maximum number of messages that can be received within one call of the function. On function completion this variable will contain the actual number of messages received from CAN network. The received number of messages may be less than the number requested by the application.
<i>unsigned long Timeout</i>	Timeout interval (in milliseconds) to wait for read completion. A value of zero instructs the library to read received messages from CAN Gateway receive buffer and return immediately. A nonzero timeout value instructs the library to return after the timeout interval has expired. The library will not wait the entire timeout interval if an error occurs or the specified number of messages is read.

Listing 10. An example usage of *PassThruReadMsgs()* function to receive up to 5 messages with inactive timeout

```

include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long msgNum; // Number of messages to receive
static const unsigned int maxMsgNum = 5; // Maximum number of messages to receive
PASSTHRU_MSG txPassThruMsg[maxMsgNum]; // Array of structures for received messages

msgNum = maxMsgNum; // Up to 5 messages will be received within one read
// Receives messages from CAN Gateway
J2534_ERROR_CODE passThruErr = PassThruWriteMsgs(idCh1, &txPassThruMsg, &msgNum, 0);
if (STATUS_NOERROR == passThruErr)
{
    // Process received messages
    std::cout << "Received messages: " << msgNum << std::endl;
}
if (ERR_BUFFER_EMPTY == passThruErr)
{
    // No messages received
}
else
{
    // User defined error handling
}

```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

Listing 11. An example usage of *PassThruReadMsgs()* function to receive up to 5 messages with active timeout

```
include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long msgNum; // Number of messages to receive
static const unsigned int maxMsgNum = 5; // Maximum number of messages to receive
PASSTHRU_MSG txPassThruMsg[maxMsgNum]; // Array of structures for received messages
static const unsigned long timeout = 100; // Timeout for message receive = 100 ms

msgNum = maxMsgNum; // Up to 5 messages will be received within one read
// Receives messages from CAN Gateway
J2534_ERROR_CODE passThruErr = PassThruWriteMsgs(idCh1, &txPassThruMsg, &msgNum, timeout);
if ((STATUS_NOERROR == passThruErr) || (ERR_TIMEOUT == passThruErr))
{
    // Process received messages
    std::cout << "Received messages: " << msgNum << std::endl;
    if (ERR_TIMEOUT == passThruErr)
    {
        // At least one message received but not maxMsgNum
        std::cout << "Timeout occurred!" << std::endl;
    }
}
if (ERR_BUFFER_EMPTY == passThruErr)
{
    // No messages received during timeout interval
}
else
{
    // User defined error handling
}
```

Additional notes:

- Error ERR_BUFFER_EMPTY is returned if no message is received within given timeout.
- Error ERR_TIMEOUT is returned if CAN Gateway is not able to read specified number of messages (maxMsgNum in above listing) but some number of messages is received.
- *PassThruReadMsgs()* function can blocks application's context for timeout interval.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.10 Activating a periodic message

The CAN Gateway hardware is able to send messages with specified period without commitment of an user application. These periodic messages can be started with *PassThruStartPeriodicMsg()* function.

An example call of a *PassThruStartPeriodicMsg(unsigned long ChannelID, PASSTHRU_MSG * pMsg, unsigned long * pMsgID, unsigned long TimeInterval)* function is shown in Listing 12. Arguments passed to the function are summarized in Table 11. For more detailed description of the function refer to [R1].

Table 11. Summary of *PassThruStartPeriodicMsg()* function

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel to be used for periodic message calculated by <i>PassThruConnect()</i> function.
<i>PASSTHRU_MSG * pMsg</i>	Pointer to a message structure with user application's periodic message.
<i>unsigned long * pMsgID</i>	Pointer to a variable to store an ID of the periodic message to be used during further communication with the CAN Gateway.
<i>unsigned long TimeInterval</i>	Time interval (in miliseconds) at which the periodic message will be repetitively transmitted. Allowed range is from 3 to 65535 ms.

Listing 12. An example usage of *PassThruStartPeriodicMsg()* function

```
#include <cstring>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idMsg; // ID of periodic message
PASSTHRU_MSG periodicMsg; // Structure for periodic message
J2534_TxFlags periodicFlags; // Additional flags for periodic message
static const unsigned long msgInterval = 250; // Interval for periodic message = 250 ms
static const unsigned int canId = 1234; // ID of CAN message

memset(&periodicMsg, 0, sizeof(PASSTHRU_MSG));
periodicFlags.bits.Can29BitId = 1; // 29-bit ID message
periodicMsg.ProtocolID = CAN;
periodicMsg.TxFlags = periodicFlags.value;
periodicMsg.Data[0] = static_cast<unsigned char>(canId >> 24);
periodicMsg.Data[1] = static_cast<unsigned char>(canId >> 16);
periodicMsg.Data[2] = static_cast<unsigned char>(canId >> 8);
periodicMsg.Data[3] = static_cast<unsigned char>(canId >> 0);
periodicMsg.Data[4] = 'T';
periodicMsg.Data[5] = 'E';
periodicMsg.Data[6] = 'S';
periodicMsg.Data[7] = 'T';
periodicMsg.DataSize = 8; // 4 bytes of canId + 4 bytes of data

// Activates a periodic message
J2534_ERROR_CODE passThruErr = PassThruStartPeriodicMsg(idCh1, &periodicMsg, &idMsg, msgInterval);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- There is a limit of 10 messages for each logical channel.

2.11 Disabling a periodic message

Previously activated periodic message can be disabled with *PassThruStopPeriodicMsg()* function.

An example call of a *PassThruStopPeriodicMsg(unsigned long ChannelID, unsigned long MsgID)* function is shown in Listing 13. Arguments passed to the function are summarized in Table 12. For more detailed description of the function refer to [R1].

Table 12. Summary of *PassThruStopPeriodicMsg()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel with periodic message to be disabled calculated by <i>PassThruConnect()</i> function.
<i>unsigned long MsgID</i>	ID of periodic message to be disabled calculated by <i>PassThruStartPeriodicMsg()</i> function

Listing 13. An example usage of *PassThruStopPeriodicMsg()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idMsg; // ID of periodic message calculated by PassThruStartPeriodicMsg() function

// Disables a periodic message
J2534_ERROR_CODE passThruErr = PassThruStopPeriodicMsg(idCh1, idMsg);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- If logical channel disconnects, all periodic messages related to this channel are disabled automatically.
- All periodic messages related to selected channel can be disabled with an IOCTL command (refer to chapter 2.13).

2.12 Retrieving text description of the last error

User application can retrieve the text description for the last PassThru function that generated an error by calling *PassThruGetLastError()* function. This function should be called immediately after an error code is returned because any following PassThru function call will wipe out the error code. The error information is in saved as a NULL terminated character chain (C-String).

An example call of a *PassThruGetLastError(char * pErrorDescription)* function is shown in Listing 14. An example usage of *PassThruGetLastError()* function. Argument passed to the function is summarized in Table 13. For more detailed description of the function refer to [R1].

Table 13. Summary of *PassThruGetLastError()* function's argument

Argument	Description
<i>char * pErrorDescription</i>	Pointer to Error Description array, which will receive the error description string. Size of the array shall be at least 80 characters.

Listing 14. An example usage of *PassThruGetLastError()* function

```
#include <iostream>
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
char descArray[80]; // An array for a error's description

// Retrives text description of last PassThru error
J2534_ERROR_CODE passThruErr = PassThruGetLastError(descArray);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
else
{
    std::cout << descArray << std::endl;
}
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.13 Executing IOCTL commands

The library allows sending general purpose I/O control commands to CAN Gateway hardware by calling *PassThruIoctl()* function. Currently implemented commands are shown in **Error! Reference source not found..**

An example calls of a *PassThruIoctl(unsigned long ChannelID, J2534_IOCTL IoctlID, void * pInput, void * pOutput)* function are shown in **Error! Reference source not found..** Arguments passed to the function are summarized in **Error! Reference source not found..** For more detailed description of the function refer to [R1].

Table 14. Implemented I/O control commands

I/O command	Description
<i>CLEAR_RX_BUFFER</i>	Clears all buffered messages from the receive queue.
<i>CLEAR_TX_BUFFER</i>	Clears all buffered messages from the transmit queue.
<i>CLEAR_PERIODIC_MSGS</i>	Clears all periodic messages from the periodic message table.
<i>CLEAR_MSGS_FILTERS</i>	Clears all message filters from the message filter table.

Table 15. Summary of *PassThruIoctl()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel for I/O command calculated by <i>PassThruConnect()</i> function.
<i>J2534_IOCTL IoctlID</i>	One of I/O commands listed in Table 11.
<i>void * pInput</i>	For I/O commands listed in Table 11, shall always be <i>NULL</i> .
<i>void * pOutput</i>	For I/O commands listed in Table 11, shall always be <i>NULL</i> .

Listing 15. An example usage of *PassThruIoctl()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 calculated by PassThruConnect() function
unsigned long idCh2; // ID of the logical channel 2 calculated by PassThruConnect() function
J2534_ERROR_CODE passThruErr = STATUS_NOERROR; // Result of PassThru functions

// Clears RX buffer of logical channel 1
passThruErr = PassThruIoctl(idCh1, CLEAR_RX_BUFFER, NULL, NULL);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
// Clears TX buffer of logical channel 2
passThruErr = PassThruIoctl(idCh2, CLEAR_TX_BUFFER, NULL, NULL);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
// Disables all periodic messages on logical channel 1
passThruErr = PassThruIoctl(idCh1, CLEAR_PERIODIC_MSGS, NULL, NULL);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
// Disables all message filters on logical channel 2
passThruErr = PassThruIoctl(idCh2, CLEAR_MSGS_FILTERS, NULL, NULL);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

Additional notes:

- If logical channel disconnects, TX and RX buffers of this channel is cleared.
- If logical channel disconnects, all message filters related to this channel are disabled automatically.
- If logical channel disconnects, all periodic messages related to this channel are disabled automatically.

2.14 Getting additional information about status of the device

The library implements additional special function *PassThruGetStatus()* which is not provided by J2534 standard ([R1]). This function is helpful when there is a need to determine the cause of failure of standard PassThru function. The function reads two hardware registers of CAN Gateway:

- CAN Status Register named CAN1SR for physical channel 1,
- CAN Status Register named CAN2SR for physical channel 2.

These registers reflect status of the NXP's CAN Controller. Description of CAN1SR and CAN2SR registers' bits can be found in chapter 16.7.8 of [R2].

ATTENTION: This function is intended to be used by advanced users only – it should not be used in standard implementations.

An example calls of a *PassThruGetStatus(unsigned long ChannelID, unsigned long * status)* function are shown in Listing 16. Arguments passed to the function are summarized in Table 16.

Table 16. Summary of *PassThruGetStatus()* function's arguments

Argument	Description
<i>unsigned long ChannelID</i>	ID of the logical channel connected to physical channel of which status register is to be read.
<i>unsigned long * status</i>	Pointer to a variable to store a status register of the CAN Gateway's physical channel.

Listing 16. An example usage of *PassThruGetStatus()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library
unsigned long idCh1; // ID of the logical channel 1 connected to the physical channel 1
unsigned long idCh2; // ID of the logical channel 2 connected to the physical channel 2
unsigned long statusCh1; // Value of physical channel 1 status register
unsigned long statusCh2; // Value of physical channel 2 status register
J2534_ERROR_CODE passThruErr = STATUS_NOERROR; // Result of PassThru functions

// Reads physical channel 1 status register
passThruErr = PassThruGetStatus(idCh1, &statusCh1);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
// Reads physical channel 2 status register
passThruErr = PassThruGetStatus(idCh2, &statusCh2);
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- Status register returned by *PassThruGetStatus()* function reflects status of a physical channel connected to logical channel passed as a first argument of the function.

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

2.15 Resetting the device

The library implements additional special function *PassThruReset()* which is not provided by J2534 standard ([R1]). This function sends reset request to CAN Gateway. If CAN device receives the request it executes hardware reset immediately.

ATTENTION: This function is intended to be used by advanced users only – it should not be used in standard implementations.

An example call of a *PassThruReset(void)* function is shown in Listing 17. The function takes no argument.

Listing 17. An example usage of *PassThruReset()* function

```
#include <hdj2534.h>
using namespace J2534; // A namespace for HDJ2534 library

// Resets CAN Gateway
J2534_ERROR_CODE passThruErr = PassThruReset();
if (STATUS_NOERROR != passThruErr)
{
    // User defined error handling
}
```

Additional notes:

- Before calling *PassThruReset()* function CAN Gateway has to be opened with *PassThruOpen()* function.
- After calling *PassThruReset()* function, connection with CAN Gateway is closed automatically. Therefore in order to call any PassThru function the device has to be reopened with *PassThruOpen()* function.

3 The HDJ2534 library in multithreaded applications

Since SAE J2534 API is not multithreaded (refer to [R1]), special attention has to be paid by a software developer during accessing the API. J2534 API's functions must not be called at the same time. In order to meet this condition a standard interprocess synchronization mechanisms like mutexes or semaphores can be used to protect against simultaneous access to library's functions. A proposal of above solution has been introduced in an example discussed in chapter 4.

Please note that calling *PassThruReadMsgs()* function with active timeout can degrade performance of multithreaded applications because threads that call API functions can be blocked for interval of the timeout (as an effect of thread synchronization). Please refer to chapter 2.9 for details.

4 An example code for Linux OS

4.1 Prerequisites

In order to use an example described in this chapter following conditions must be met:

- CAN Gateway device is connected in loopback i.e. CANL line of channel 1 is connected with CANL line of channel 2 and CANH line of channel 1 is connected with CANH line of channel 2,
- Linux OS is installed on developer's machine,
- Eclipse IDE for C/C++ developers (Kepler version or newer) is installed on developer's machine,
- HDJ2534 library (v1.0.266 or newer) is installed on developer's machine,
- GCC is installed on developer's machine.

4.2 Configuration instructions for the example

The example code is provided as exported Eclipse project. In order to configure example following steps must be performed:

- 1) Run Eclipse and select proper workspace,
- 2) From main menu choose "File→Import...",
- 3) Select "Existing Project into Workspace" from "General group",
- 4) Mark "Select archive file", click "Browse..." button and select cangw_example.zip file attached to this document and click "Finish",
- 5) Build the project with "Project→Build All".

4.3 Description of the example

The example consists from three source files:

- **main.cpp**
The main file of the project. It implements general exchange mechanism for messages.
- **cangw.hpp**
This file implements CAN Gateway handling functions based on SAE J2534 API.
- **cangw.hpp**
A header file for the cangw.cpp file.

The example is a simple program which sends frames from channel 1 to channel 2 of CAN Gateway and vice versa. Messages are sent in a loop located in the main function with a *canGw::send()* function. Furthermore two periodic messages (one message for each channel) are activated. Messages are received from CAN Gateway in threads (one thread for each channel) managed by

CAN Gateway – application note

All intellectual properties belongs to Hatteland Display AS

a code located in the *cangw.cpp* file. Received messages are processed in callback functions located in the *main.cpp* file. These callbacks are raised by receive threads.

5 Related documents

- [R1] SAE J2534 API Reference, Drew Technologies, Inc. 2003
- [R2] UM10360 LPC176x/5x User manual Rev. 3, NXP Semiconductor http://www.nxp.com/documents/user_manual/UM10360.pdf