CaNMate dll aPI User ManUal

Introduction

CANMate is a high performing, yet low cost CAN to USB converter designed and manufactured by Deep Thought Systems (P) Ltd. Even though CANMate ships with powerful free software applications to do the basic operations, we perfectly understand that it is highly desirable for users to create their own custom applications. Hence we have incorporated a simple, yet powerful API implemented by the library, CANMate Dll. This document explains the relevant API functions and provides information to use the API functions in user programs.

Overview of CANMate Library

CANMate API and hardware together implements a command response mechanism to carry out the tasks related to CAN bus sniffing. CANMate Dll has a multi threaded architecture and uses virtual COM port provided by the underlying hardware to communicate with the CANMate device.

Most of the API functions are designed as blocking functions except the function for transmitting CAN messages. This makes the programming model simple and programs easy to implement. Internal time out mechanisms are implemented to avoid any API function going into a wait mode. Because of the blocking nature of the APIs only one command can be issued at a time and the next command can be issued only after the first command has been completed and the results returned by the hardware. It is to be noted that only exception for this rule is API function to transmit CAN massages since it is a non blocking function. This enables one to write applications with high message rates.

Applications using CANMate Dll are required to implement two callback functions. One is for receiving CAN Messages and other is for receiving events. Errors like CAN Bus errors are reported using the event callback mechanism.

APIs are explained in detail in the following sections.

CANMate DLL provides the required APIs to communicate with the CANMate device. The API details are given below.

Most of the CANMate APIs sends a command to the device to take some action. So it has to wait till a reply comes.

API Details

1. **HANDLE OpenCANMate(LPVOID lpDataCallBack = NULL, LPVOID lpEventCallBack = NULL)**

   This API enumerates all COM ports and finds the COM port to which CANMate hardware is connected. It then opens the CANMate. CANMate device internally initializes the CAN controller and replies with the status.

   Note : Present version of CANMate Dll supports only one CANMate. It will open only the first CANMate even if multiple CANMate devices are connected to a PC.

   Parameters :

   1. LPVOID lpDataCallBack

      This is a pointer to a call back function for the application to receive data

   2. LPVOID lpEventCallBack

      This is a pointer to a call back function for the application to receive events and error notifications.

   Return Value :

   Returns the COM port handle if SUCCESS. Returns NULL if there is any error.

2. **int CloseCANMate(HANDLE hDev)**

   This API sends the CLOSE command to CANMate. It also closes the COM port and frees all the related resources

   Parameters:

   1. HANDLE hDev

   Handle of the COM port connected to CANMate device.

   Return value :

   Returns CANMate_ERROR_SUCCESS if successful

3. **int SetCANBaudRate(HANDLE hDev, char chBaudRate)**

This command sets the CAN baud rate of the CANMate.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

2. char chBaudRate

Baud rate to be set. This should be one of the following.

```
BAUD_RATE10K
BAUD_RATE20K
BAUD_RATE50K
BAUD_RATE80K
BAUD_RATE100K
BAUD_RATE125K
BAUD_RATE150K
BAUD_RATE175K
BAUD_RATE200K
BAUD_RATE225K
BAUD_RATE250K
BAUD_RATE275K
BAUD_RATE300K
BAUD_RATE500K
BAUD_RATE625K
BAUD_RATE800K
BAUD_RATE1000K
```

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

4. int WriteCANMessage(HANDLE hDev,CANMsg* pMsg)

This function transmits a single CAN message. No ack is given for the transmitted message and hence the function does not wait.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

2. CANMsg*pMsg

Pointer to a can message structure

Return value :

Returns number of bytes written

5. Int StartReception (HANDLE hDev)

This function sets the CANMate to transfer the received messages to the PC application. CANMate firmware will not transfer the messages to PC unless this function is invoked.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

6. int StopReception (HANDLE hDev

This function instructs the CANMate firmware to stop transferring the received messages to the PC application. It is advised to call this function if the PC app is not listening for any CAN messages.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

7. Int SetNormalMode(HANDLE hDev)

This function sets the CANMate in "Normal" mode. This is the default mode in which CANMate powers up. This mode should be set for normal operation of CANMate.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

8. int SetLoopbackMode(HANDLE hDev)

This function sets the CAN controller inside CANMate in "Loopback" mode. In loopback mode transmitted messages from PC application will not be transmitted in the CAN bus, but will be returned as received messages. This mode is useful during application development as a testing and debugging aid.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

9. int GetCurrentMode(HANDLE hDev, int* pData)

This function gets the current CAN controller operating mode.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

2. Int* pData

The current mode will be returned in this pointer. It can be one of the following values.

```
ECAN_NORMAL_MODE
ECAN_LOOPBACK_MODE
```

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

10. int GetCurrentBaudRate(HANDLE hDev, int* pData)

This function gets the current CAN controller baud rate.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

2. Int* pData

The current baud rate will be returned in this pointer. It can be one of the following values.

```
BAUD_RATE10K
BAUD_RATE20K
BAUD_RATE50K
BAUD_RATE80K
BAUD_RATE100K
BAUD_RATE125K
BAUD_RATE150K
BAUD_RATE175K
BAUD_RATE200K
BAUD_RATE225K
BAUD_RATE250K
BAUD_RATE275K
BAUD_RATE300K
BAUD_RATE500K
BAUD_RATE625K
BAUD_RATE800K
BAUD_RATE1000K
```

Return value :

Returns CANMate_ERROR_SUCCESS if successful.

11.int GetFirmwareVersion(HANDLE hDev, int* pData)

This function gets the current CANMate firmware version number. Version number is a 2 byte format with MSB representing the Major version and LSB the Minor version.

Parameters

1. HANDLE hDev

Handle of the COM port connected to CANMate device.

2. Int* pData

   The current firmware version number will be returned in this pointer.

Return value :

.Returns CANMate_ERROR_SUCCESS if successful.

12. typedef int (*EVNT_CALLBACK)(CANEvent *pnEvnt)

This is the prototype of event call back.

Parameters

1. CANEvent *pnEvnt

Pointer to a CANEvent structure.

Return value :

Returns value is currently ignored by the DLL. It is advised to return 0 for future compatibility.

13. typedef int (*DATA_CALLBACK)(CANMsg *pMsg, int *nNumMsgs)

This is the prototype of data call back.

Parameters

1. CANMsg *pMsg

Pointer to a CANMsg structure. For receiving CAN Messages.

2. int *nNumMsgs

Number of messages will be returned in this pointer.

*Note : Currently, the number of messages returned is fixed as 1. Hence the application only needs to ensure that the memory pointed by parameter 1 (CANMsg *pMsg) has enough size to store a single CAN Message.*

Return value :

Returns value is currently ignored by the DLL. It is advised to return 0 for future compatibility.

CANMate Dll Structures

This section explains the structures used by the CANMate Dll.

1. CAN Message Structure

Fields :

bExtended : This field should be non zero for extended messages and 0 for standard messages

chTmStmpH : Upper byte of time stamp field.
chTmStmpL : Lower byte of time stamp field
Time stamp is ignored for transmit messages

EArbId1 : MSB containing bits 25 to 29 in the case of extended messages and igonored for standard messages

EArbId0 : 3rd Byte in the case of extended messages and igonored for standard messages

SArbId1 : 2nd Byte in the case of extended messages and MSB containing bits 9 to 11 for standard messages

SArbId0 : 1st Byte for both extended and standard messages

DLC : Message length (8 maximum)

D0 to D7 : Message bytes.

2. CAN Event Structure

Fields :

chErr : The type of error

This is a bit field and following is the definition

UART_ERROR	0x01	/ / 0x0000 0001
CAN_TXERROR	0x02	/ / 0x0000 0010
CAN_BUS_OFF	0x04	/ / 0x0000 0100
CAN_BUF_OVERFLOW	0x08	/ / 0x0000 1000
CAN_TX_PASSIVE	0x10	/ / 0x0001 0000
CAN_RX_PASSIVE	0x20	/ / 0x0010 0000

CAN_ERR_WARNING	0x40	/ / 0x0100 0000
PC_BUF_OVERFLOW	0x80	/ / 0x1000 0000

chTxErrCnt : Transmit error counter value from the hardware

chRxErrCnt : Receive error counter value.

Note on Error Handling : Specific CAN errors CAN_BUS_OFF, CAN_TX_PASSIVE and CAN_RX_PASSIVE will put CAN controller in CANMate hardware into an irrecoverable error mode and will require closing and opening the device again from the application software.

