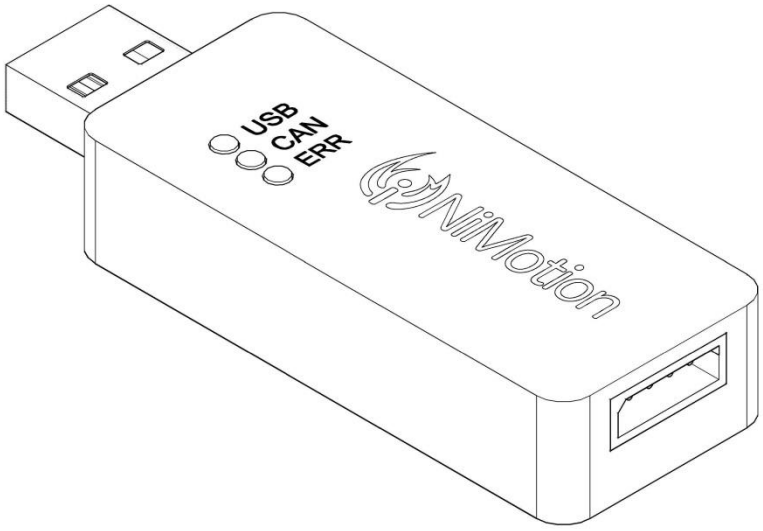
NiMotionUSBCAN 动态库使用手册

版本：B



北京立迈胜控制技术有限责任公司

Beijing Nimotion Control Technology Co., Ltd.

目　　录

1 接口函数库说明及其使用

1.1 接口卡设备类型定义

设备名称	设备类型号
USBCAN	3

1.2 错误码定义

名称	值	描述
ERR_CAN_OVERFLOW	0x00000001	CAN 控制器内部 FIFO 溢出
ERR_CAN_PASSIVE	0x00000004	CAN 控制器消极错误
ERR_CAN_BUSERR	0x00000010	CAN 控制器总线错误
ERR_BUFFEROVERFLOW	0x00000800	缓冲区溢出
ERR_CMDFAILED	0x00004000	执行命令失败错误码

1.3 接口库函数结构体

1.3.1 BOARD_INFO

描述：BOARD_INFO 结构体包含设备信息。结构体将在 NiM_ReadBoardInfo 函数中被填充。

```
typedef struct _BOARD_INFO
{
    USHORT  hw_Version;
    USHORT  fw_Version;
    USHORT  dr_Version;
    USHORT  in_Version;
    USHORT  irq_Num;
    BYTE can_Num;
    CHAR  str_Serial_Num[20];
    CHAR  str_hw_Type[40];
    USHORT  Reserved[4];
} BOARD_INFO,  *P_BOARD_INFO;
```

成员：

hw_Version

硬件版本号，用 16 进制表示。比如 0x0100 表示 V1.00。

fw_Version

APP 版本号，用 16 进制表示。

dr_Version

IAP 版本号，用 16 进制表示。

in_Version

接口库版本号，用 16 进制表示。

irq_Num

板卡所使用的中断号。

can_Num

表示有几路 CAN 通道。

str_Serial_Num

此板卡的序列号。

str_hw_Type

硬件类型，比如“USBCAN V1.00”（注意：包括字符串结束符’\0’）。

Reserved

系统保留。

1.3.2 CAN_OBJ

描述:CAN_OBJ 结构体是 CAN 帧结构体，即 1 个结构体表示一个帧的数据结构。在发送函数 NiM_Transmit 和接收函数 NiM_Receive 函数中被用来传送 CAN 信息帧。

```
typedef struct _CAN_OBJ
{
    UINT  ID;
    UINT  TimeStamp;
    BYTE TimeFlag;
    BYTE SendType;
    BYTE RemoteFlag;
    BYTE ExternFlag;
    BYTE DataLen;
    BYTE Data[8];
    BYTE Reserved[3];
} CAN_OBJ,  *P_CAN_OBJ;
```

成员:

ID

报文 ID。

TimeStamp

接收到信息帧时的时间标识，从 CAN 控制器初始化开始计时。

TimeFlag

是否使用时间标识，为 1 时 TimeStamp 有效，TimeFlag 和 TimeStamp 只在此帧为接收帧时有意义。

SendType

发送帧类型，=0 时为正常发送，=1 时为单次发送，=2 时为自发自收，=3 时为单次自发自收，只在此帧为发送帧时有意义。

RemoteFlag

是否是远程帧。

ExternFlag

是否是扩展帧。

DataLen

数据长度(<=8)，即 Data 的长度。

Data

报文的数据。

Reserved

系统保留。

1.3.3 CAN_STATUS

描述:CAN_STATUS 结构体包含 CAN 控制器状态信息。结构体将在 NiM_ReadCanStatus 函数中被填充。

```
typedef struct   _CAN_STATUS
{
    UCHAR  ErrInterrupt;
    UCHAR  regMode;
    UCHAR  regStatus;
    UCHAR  regALCapture;
    UCHAR  regECCapture;
    UCHAR  regEWLimit;
    UCHAR  regRECounter;
    UCHAR  regTECounter;
    DWORD  Reserved;
} CAN_STATUS,  *P_CAN_STATUS;
```

成员:

ErrInterrupt

中断记录，读操作会清除。

regMode

CAN 控制器模式，0 正常模式，1 睡眠模式。

regStatus

CAN 控制器状态，1 接收模式，2 发送模式，0 空闲。

regALCapture

CAN 控制器仲裁丢失寄存器，regALCapture 的后 3 位对应表示 3 个邮箱（邮箱 0,1,2）的仲裁情况，1 表示丢失。

regECCapture

CAN 控制器错误类型，1 被动模式错误，2 总线关闭，0 无错误。

regEWLimit

CAN 控制器错误警告，1 错误警告。

regRECounter

CAN 控制器接收错误寄存器。

regTECounter

CAN 控制器发送错误寄存器。

Reserved

系统保留。

1.3.4 ERR_INFO

描述：ERR_INFO 结构体用于装载 NiMotionUSBCAN 库运行时产生的错误信息。结构体将在 NIM_ReadErrInfo 函数中被填充。

```
typedef struct _ERR_INFO
{
    UINT  ErrCode;
    BYTE Passive_ErrData[3];
    BYTE ArLost_ErrData;
} ERR_INFO,  *P_ERR_INFO;
```

成员:

ErrCode

错误码。

Passive_ErrData

当产生的错误中有消极错误时表示为消极错误的错误标识数据。

ArLost_ErrData

当产生的错误中有仲裁丢失错误时表示为仲裁丢失错误的错误标识数据。

1.3.5 INIT_CONFIG

描述： INIT_CONFIG 结构体定义了初始化 CAN 的配置。结构体将在 NiM_InitCan 函数中被填充。

```
typedef struct _INIT_CONFIG
{
    DWORD   AccCode;
    DWORD   AccMask;
    DWORD   Reserved;
    UCHAR  Filter;
    UCHAR  Timing0;
    UCHAR  Timing1;
    UCHAR  Mode;
} INIT_CONFIG,  *P_INIT_CONFIG;
```

成员

AccCode

验收码。

AccMask

屏蔽码。

Reserved

保留。

Filter

滤波方式。

Timing0

定时器 0（BTR0）。

Timing1

定时器 1（BTR1）。

Mode

模式。

Timing0 和 Timing1 用来设置 CAN 波特率，几种常见的波特率设置如下：

CAN 波特率	定时器 0	定时器 1
5Kbps	0xBF	0xFF
10Kbps	0x31	0x1C
20Kbps	0x18	0x1C
40Kbps	0x87	0xFF
50Kbps	0x09	0x1C
80Kbps	0x83	0Xff
100Kbps	0x04	0x1C
125Kbps	0x03	0x1C
200Kbps	0x81	0xFA
250Kbps	0x01	0x1C
400Kbps	0x80	0xFA
500Kbps	0x00	0x1C
666Kbps	0x80	0xB6
800Kbps	0x00	0x16
1000Kbps	0x00	0x14
自适应	0xFF	0xFF

1.4 接口库函数说明

1.4.1 NiM_OpenDevice

描述： 此函数用以打开设备。注意一个设备只能打开一次。

```
DWORD call NiM_OpenDevice (DWORD DevType, DWORD DevIndex, DWORD Reserved);
```

参数:

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

Reserved

参数无意义。

返回值:

为 1 表示操作成功，0 表示操作失败。

示例:

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  =  3;  /*  usbcan  */
int   nDeviceInd =  0;  /*     */
int   nReserved  =0;
DWORD   dwRel;
dwRel    =  NiM_OpenDevice (nDeviceType,  nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"),  _T("警告"), MB_OK|MB_ICONQUESTION);
    return  FALSE;
}
```

1.4.2 NiM_CloseDevice

描述： 此函数用以关闭设备。

```
DWORD   stdcall NiM_CloseDevice (DWORD  DevType, DWORD   DevIndex);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

返回值

为 1 表示操作成功，0 表示操作失败。

示例：

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  =  3;  //
int   nDeviceInd =  0;  //
BOOL  dwRel;
dwRel =   NiM_CloseDevice (nDeviceType,   nDeviceInd);
```

1.4.3 NiM_InitCan

描述： 此函数用以初始化指定的 CAN。

```
DWORD   stdcall NiM_InitCAN (DWORD DevType,  DWORD  DevIndex, DWORD  CANIndex,
P_INIT_CONFIG  pInitConfig);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

pInitConfig

初始化参数结构

成员	功能描述
pInitConfig->AccCode	AccCode 对应 SJA1000 中的四个寄存器 ACR0，ACR1，ACR2，ACR3， 其中高字节对应 ACR0，低字节对应 ACR3；AccMask 对应 SJA1000 中的四个寄存器 AMR0，AMR1，AMR2， AMR3，其中高字节对应 AMR0，低字节对应 AMR3。
pInitConfig->AccMask	
pInitConfig->Reserved	保留
pInitConfig->Filter	滤波方式，1 表示单滤波，0 表示不双滤波
pInitConfig->Timing0	定时器 0
pInitConfig->Timing1	定时器 1
pInitConfig->Mode	模式，0 表示正常模式，1 表示只听模式

返回值 为 1 表示操作成功，0 表示操作失败。

示例：

```
#include "NiMotionUSBCAN.h"
int    nDeviceType  =  3;  //
int    nDeviceInd =  0;  //
int    nCANInd     =  0;
int    nReserved  =  0;  //
DWORD   dwRel;
dwRel = NiM_InitCAN (nDeviceType,  nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"),  _T("警告"), MB_OK|MB_ICONQUESTION);
}
```

1.4.4 NiM_ReadBoardInfo

描述：此函数用以获取设备信息。

```
DWORD call NiM_ReadBoardInfo (DWORD DevType, DWORD DevIndex, P_BOARD_INFO pInfo);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

pInfo

用来存储设备信息的 BOARD_INFO 结构指针。 返回值

为 1 表示操作成功，0 表示操作失败。

示例：

```
#include "NiMotionUSBCAN.h"
int nDeviceType = 3;
int nDeviceInd = 0;
int nCANInd = 0;
INIT_CONFIG vic;
BOARD_INFO vbi;
DWORD dwRel;
dwRel = NiM_ReadBoardInfo (nDeviceType, nDeviceInd, nCANInd, &vbi);
```

1.4.5 NiM_ReadErrInfo

描述： 此函数用以获取最后一次错误信息。

```
DWORD call NiM_ReadErrInfo (DWORD DevType,DWORD DevIndex,DWORD CANIndex,P_ERR_INFO pErrInfo);
```

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

pErrInfo

用来存储错误信息的 ERR_INFO 结构指针。

名称	值	描述
ERR_CAN_OVERFLOW	0x00000001	CAN 控制器内部 FIFO 溢出
ERR_CAN_PASSIVE	0x00000004	CAN 控制器消极错误
ERR_CAN_BUSERR	0x00000010	CAN 控制器总线错误

ERR_BUFFEROVERFLOW	0x00000800	缓冲区溢出
ERR_CMDFAILED	0x00004000	执行命令失败错误码

返回值 为 1 表示操作成功，0 表示操作失败。

备注

当(PErrInfo->ErrCode&0x0004)==0x0004 时，存在 CAN 控制器消极错误。

PErrInfo->Passive_ErrData[0]错误代码捕捉位功能表示

错误代码类型功能说明

错误代码	含义
000	没有错误
001	位填充错
010	格式错
011	确认错
100	隐性位错
101	显性位错
110	CRC 错
111	由软件写入

PErrInfo->Passive_ErrData[1] 表示接收错误计数器

PErrInfo->Passive_ErrData[2] 表示发送错误计数器

示例:

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  = 3;
int   nDeviceInd =  0;
int   nCANInd    =  0;
VCI_ERR_INFO  vei;
DWORD   dwRel;
bRel =  NiM_ReadErrInfo (nDeviceType, nDeviceInd, nCANInd,  &vei);
```

1.4.6 NiM_ReadCANStatus

描述: 此函数用以获取 CAN 状态。

```
DWORD call  NiM_ReadCANStatus(DWORD DevType, DWORD DevIndex, DWORD CANIndex,
P_CAN_STATUS pCANStatus);
```

参数:

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

pCANStatus

用来存储 CAN 状态的 CAN_STATUS 结构指针。

返回值:

为 1 表示操作成功，0 表示操作失败。

示例

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  =  3;  //
int   nDeviceInd =  0;  //
int   nCANInd    =  0;
VCI_INIT_CONFIG  vic;
VCI_CAN_STATUS  vcs;
DWORD   dwRel;
dwRel =  NiM_ReadCANStatus(nDeviceType,  nDeviceInd, nCANInd,  &vcs);
```

1.4.7 NiM_GetReceiveNum

描述:此函数用以获取指定接收缓冲区中接收到但尚未被读取的帧数。主要用途是配合 NIM_Receive 使用，即缓冲区有数据，再接收。用户无需一直调用 NIM_Receive，可以节约 PC 系统资源，提高程序效率。

```
ULONG call  NiM_GetReceiveNum(DWORD   DevType, DWORD    DevIndex,  DWORD CANIndex);
```

参数:

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

返回值:

返回尚未被读取的帧数。 示例

```
#include "NiMotionUSBCAN.h"
int nDeviceType  =3;
int   nDeviceInd =  0;
int   nCANInd    =  0;
DWORD   dwRel;
bRel =  NiM_GetReceiveNum(nDeviceType,  nDeviceInd, nCANInd);
```

1.4.8 NiM_ClearBuffer

描述:此函数用以清空指定缓冲区。主要用于需要清除接收缓冲区数据的情况。

```
DWORD call  NiM_ClearBuffer(DWORD  DevType,   DWORD   DevIndex,   DWORD CANIndex);
```

参数:

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

返回值:

为 1 表示操作成功，0 表示操作失败。 示例

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  =  3;
int   nDeviceInd =  0;
int   nCANInd    =  0;
DWORD   dwRel;
dwRel =  NiM_ClearBuffer(nDeviceType,  nDeviceInd, nCANInd);
```

1.4.9 NiM_StartCAN

描述： 此函数用以启动 CAN。

```
DWORD call NiM_StartCAN (DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

返回值：

为 1 表示操作成功，0 表示操作失败。

示例：

```
#include "NiMotionUSBCAN.h"
int nDeviceType  =3;
int   nDeviceInd =  0;
int   nCANInd    =  0;
int   nReserved  =  0;
VCI_INIT_CONFIG  vic;
DWORD   dwRel;
dwRel    =  NiM_OpenDevice(nDeviceType,  nDeviceInd, nReserved);
if (dwRel != STATUS_OK)
{
    MessageBox(_T("打开设备失败!"),  _T("警告"), MB_OK|MB_ICONQUESTION);
    return  FALSE;
}
dwRel  =  NiM_InitCAN(nDeviceType, nDeviceInd, nCANInd,  &vic);
if (dwRel ==  STATUS_ERR)
```

```
{
    NiM_CloseDevice (nDeviceType,  nDeviceInd);
    MessageBox(_T("初始化设备失败!"),  _T("警告"), MB_OK|MB_ICONQUESTION);
    return  FALSE;
}
dwRel  = NiM_StartCAN(nDeviceType,  nDeviceInd, nCANInd);
if (dwRel ==  STATUS_ERR)
{
    NiM_CloseDevice(nDeviceType,  nDeviceInd);
    MessageBox(_T("启动设备失败!"),  _T("警告"), MB_OK|MB_ICONQUESTION);
    return  FALSE;
}
```

1.4.10 NiM_Transmit

描述：此函数从指定的设备 CAN 通道的发送数据。

```
ULONG call NiM_Transmit(DWORD DevType, DWORD DevIndex, DWORD CANIndex, P_CAN_OBJ pSend,
ULONG  Len);
```

参数

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

pSend

要发送的数据帧数组的首指针。

Len

要发送的数据帧数组的长度。

返回值：

返回实际发送的帧数。

示例：

```
#include "NiMotionUSBCAN.h"
#include <string.h>
int   nDeviceType  =  3;
int   nDeviceInd =  0;
int   nCANInd    =  0;
DWORD   dwRel;
VCI_CAN_OBJ  vco;
ZeroMemory(&vco,sizeof(CAN_OBJ));
vco.ID  =  0x00000000;
vco.SendType  =  0;
```

```
vco.RemoteFlag = 0;
vco.ExternFlag = 0;
vco.DataLen = 8;
dwRel = NiM_Transmit(nDeviceType, nDeviceInd, nCANInd, &vco, i);
```

1.4.11 NiM_Receive

描述： 此函数从指定的设备 CAN 通道的接收缓冲区中读取数据。建议在调用之前，先调用 NIM_GetReceiveNum。函数获知缓冲区中有多少帧，然后对应地去接收。

```
ULONG stdcall NiM_Receive(DWORD DevType, DWORD DevIndex, DWORD CANIndex,
PVCI_CAN_OBJ pReceive, ULONG Len, INT WaitTime=－1);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

pReceive

用来接收的数据帧数组的首指针。

Len

用来接收的数据帧数组的长度。

WaitTime

等待超时时间，以毫秒为单位。

返回值

返回实际读取到的帧数。如果返回值为 0xFFFFFFFF，则表示读取数据失败，有错误发生，请调用 NiM_ReadErrInfo 函数来获取错误码。

示例：

```
#include "NiMotionUSBCAN.h"
#include <string.h>
int nDeviceType = 3;
int nDeviceInd = 0;
int nCANInd = 0;
DWORD dwRel;
VCI_CAN_OBJ vco[100];
lRet = NiM_Receive(nDeviceType, nDeviceInd, nCANInd, vco, 100, 10);
```

1.4.12 NiM_ResetCAN

描述： 此函数用以复位 CAN。主要用与 NiM_StartCAN 配合使用，无需再初始化，即可恢复 CAN 卡的正常状态。比如当 CAN 卡进入总线关闭状态时，可以调用这个函数。

```
DWORD call NiM_ResetCAN (DWORD DevType, DWORD DevIndex, DWORD CANIndex);
```

参数：

DevType

设备类型号。

DevIndex

设备索引号，比如当只有一个设备时，索引号为 0，有两个时可以为 0 或 1。

CANIndex

第几路 CAN。

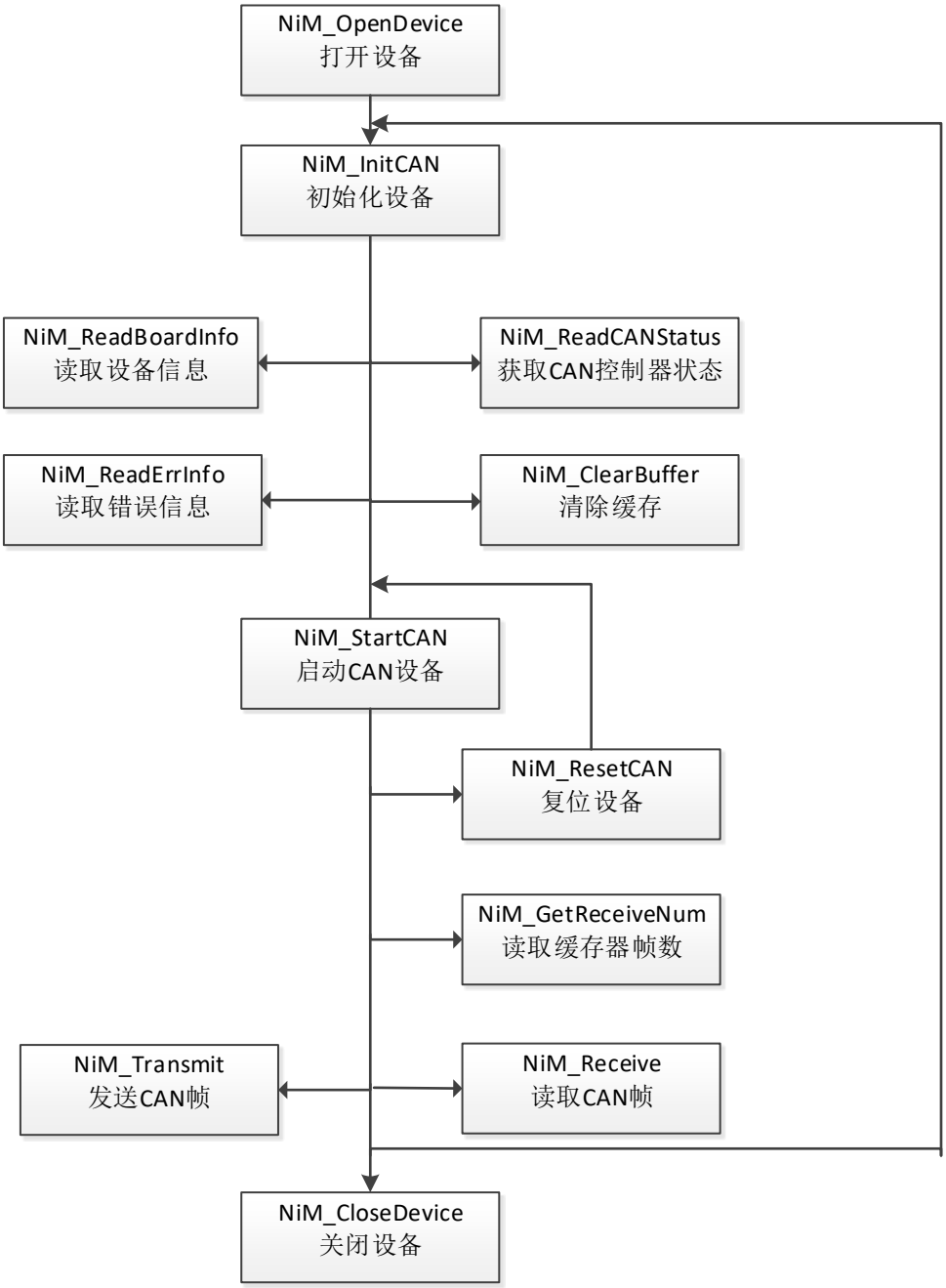
返回值

为 1 表示操作成功，0 表示操作失败。

示例：

```
#include "NiMotionUSBCAN.h"
int   nDeviceType  =  6;
int   nDeviceInd =  0;
int   nCANInd   =  0;
DWORD   dwRel;
bRel =  NiM_ResetCAN (nDeviceType,  nDeviceInd, nCANInd);
```

2 函数调用方法


NiM_OpenDevice
打开设备
NiM_InitCAN
初始化设备
NiM_ReadBoardInfo
读取设备信息
NiM_ReadCANStatus
获取CAN控制器状态
NiM_ReadErrInfo
读取错误信息
NiM_ClearBuffer
清除缓存
NiM_StartCAN
启动CAN设备
NiM_ResetCAN
复位设备
NiM_GetReceiveNum
读取缓存器帧数
NiM_Transmit
发送CAN帧
NiM_Receive
读取CAN帧
NiM_CloseDevice
关闭设备