动态库接口函数使用说明

日期	版本	更改内容
2022.7.6	V1.0	初始版本

目 录

1 简介

1.1 概述

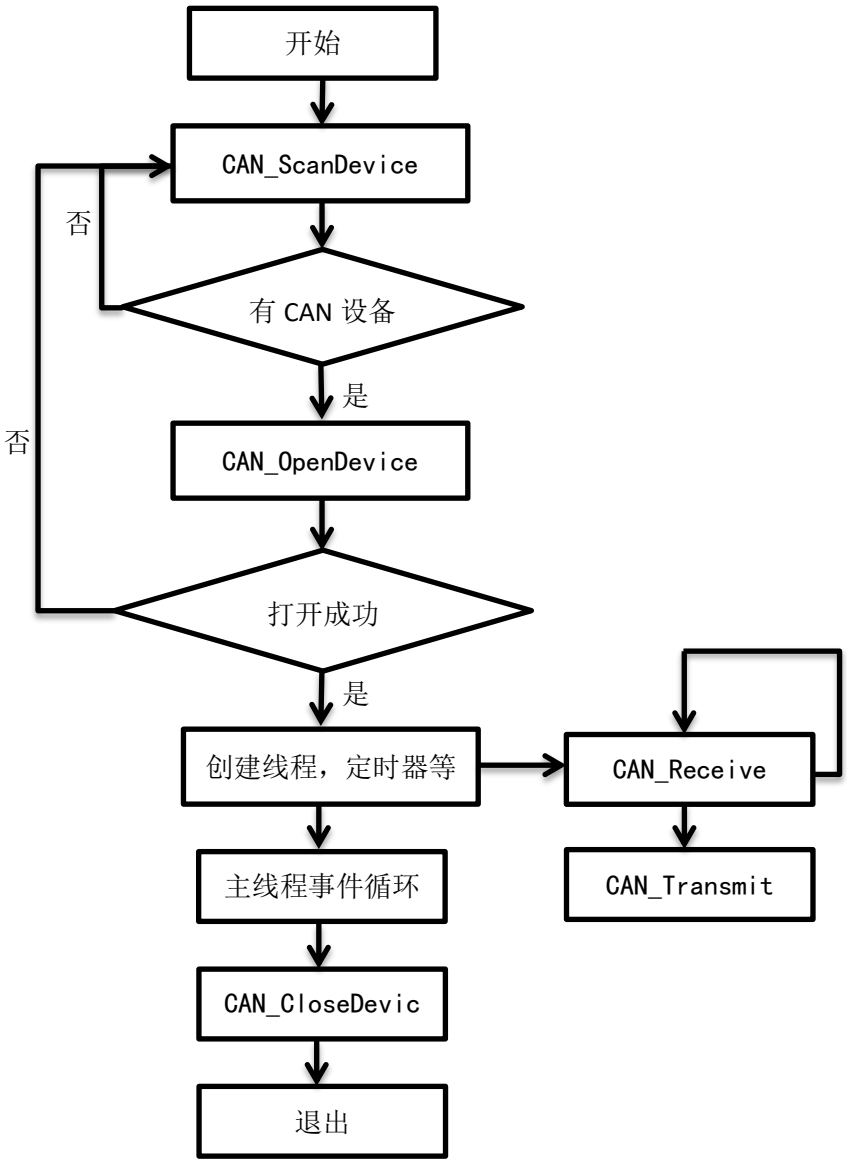
ucanbus.dll 配合 CAN 协议分析仪，可进行二次开发。

1.2 运行环境

支持 Windows 系统(WinXP-WIN11),32 位或 64 位,支持 VC,VB,C#,Qt,delphi,labview，Matlab 等开发环境。

1.3 CAN 启动及操作流程

CAN 启动和操作流程图如下：



2 结构体说明

2.1 Can_Msg

Can_Msg结构体表示帧的数据结构。在发送函数Transmit和接收函数Receive中被用来传送CAN信息帧：

```
typedef  struct  _Can_Msg {
unsigned int    ID;
unsigned int    TimeStamp;
char            FrameType;
char            DataLen;
char            Data[8];
char            ExternFlag;
char            RemoteFlag;
char            BusSatus;
char            ErrSatus;
char            TECounter;
char            RECounter;
}Can_Msg, *P_Can_Msg;
```

成员：

1. ID报文帧ID。
2. 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，单位微秒。
3. FrameType帧类型。等于0时为接收帧，等于1时为发送帧，等于2时表示传输失败。
4. DataLen数据长度DLC(<=8)，即Data的长度。
5. Data CAN报文的数据。空间受DataLen的约束。
6. ExternFlag是否是扩展帧。=0时为标准帧（11位帧ID)，=1时为扩展帧（29位帧ID)。
7. RemoteFlag是否是远程帧。=0时为数据帧，=1时为远程帧。
8. BusSatus 总线状态,定义见表1。
9. ErrSatus 错误状态，定义见表2。
10. TECounter 发送错误计数。
11. RECounter 接收错误计数。

2.2 Can_Status

Can_Status 结构体用于获取 CAN 设备状态。

```
typedef struct _Can_Status {
  char        BusSatus; 总线状态,定义见表 1。
  char        ErrSatus; 错误状态，定义见表 2。
  char        TECounter; 发送错误计数。
  char        RECounter; 接收错误计数。
}Can_Status, *P_Can_Status;
```

成员

1. BusSatus 总线状态,定义见表 1。
2. ErrSatus 错误状态，定义见表 2。
3. TECounter 发送错误计数。
4. RECounter 接收错误计数。

2.3 Can_Config

Can_Config 结构体定义了初始化 CAN 的配置。结构体将在 CAN_Init 函数中被填充，即初始化之前，要先填好这个结构体变量

```
typedef struct _Can_Config {
    unsigned int       baudrate;
    unsigned int       configs;
    unsigned int       model;
}Can_Config, *P_Can_Config;
```

成员

1. baudrate 波特率，直输，无需计算定时器。
2. configs 配置信息:0x01 接通内部电阻,0x02 离线唤醒,0x04 自动重传。
3. model 工作模式：0 正常模式,1 环回模式,2 静默模式,3 静默环回模式。

表 1 CAN 总线状态

序号	值(组合)	状态描述
1	0x01	错误警告
2	0x02	错误被动
3	0x04	CAN 总线离线
4	0x08	总线正常
5	0x10	发送邮箱总裁丢失
6	0x20	发送邮箱空
7	0x40	接收寄存器溢出
8	0x80	接收寄存器满

表 2 CAN 错误状态

序号	值(非组合)	状态描述
1	0x01	位填充错
2	0x02	格式(Form)错
3	0x03	确认(ACK)错
4	0x04	隐形位错
5	0x05	显性位错
6	0x06	CRC 错

3 函数说明

3.1 Reg_HotPlug_Func

描述

注册 CAN 设备热拔插函数，可用于检测提醒插入拔出 CAN 设备。

```
int __stdcall   Reg_HotPlug_Func(void(*pfunc)(void));
```

参数

pfunc 热拔插函数指针

返回值

0 表示成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
CcanbustestDlg dlg;
m_pMainWnd = &dlg;
pdig = &dlg;
Reg_HotPlug_Func(&func_hotplugs);
```

3.2 CAN_ScanDevice

描述

此函数用于扫描 CAN 设备。

```
int __stdcall   CAN_ScanDevice(void);
```

参数

无

返回值

返回扫描到 CAN 设备的个数，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int devs = 0;
devs = CAN_ScanDevice()。
```

3.3 CAN_OpenDevice

描述

此函数用于打开 CAN 设备。

```
int __stdcall   CAN_OpenDevice(unsigned int devNum);
```

参数

devNum 设备通道号

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret = CAN_OpenDevice(0);
if(ret != 0) {cout<<"打开 CAN 设备失败！"; return;}
```

3.4 CAN_CloseDevice

描述

此函数用于关闭 CAN 设备。

```
int __stdcall  CAN_CloseDevice(unsigned int devNum);
```

参数

devNum 设备通道号

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret = CAN_CloseDevice (0);
if(ret != 0) {cout<<"关闭 CAN 设备失败！"; return;}
```

3.5 CAN_Init

描述

此函数用于初始化 CAN 设备。

```
int __stdcall CAN_Init(unsigned int devNum , P_Can_Config pInitConfig);
```

参数

devNum 设备通道号

pInitConfig 初始化参数结构，为一个 P_Can_Config 结构体变量，初始化成功后，该结构体的成员 baudrate 被赋实际波特率值，可用作评估误码率，误差的参考。

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
Can_Config cancfg;
cancfg.model = 0;//正常工作模式
baudrate = 1000*1000;//波特率为 1m
cancfg.configs = 0x0001;  //接通内部电阻
cancfg.configs |= 0x0002; //总线离线唤醒
cancfg.configs |= 0x0004; //自动重发
ret = CAN_Init(0, &cancfg);
if(ret != 0) {cout<<"初始化 CAN 设备失败！"; return;}
cout <<"实际波特率"<< cancfg.baudrate;
```

3.6 CAN_SetFilter

描述

此函数用于设置 CAN 过滤器。

int __stdcall **CAN_SetFilter**(unsigned int devNum, char namber, char type, unsigned int ftID, unsigned int ftMask, char enable);

参数

devNum 设备通道号

namber 过滤器编号，范围 0-13

type 过滤器类型，0：2 个 16 位过滤器，1：一个 32 位过滤器。

ftID 标识符

ftMask 屏蔽位

enable 是否使能， 0：非使能，1：使能

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
ret = CAN_SetFilter(0,0,1,0,0,1);
if(ret != 0) {cout<<"操作失败！"; return;}
```

3.7 CAN_Reset

描述

此函数用于复位 CAN 过设备。

int __stdcall **CAN_Reset**(unsigned int devNum);

参数

devNum 设备通道号

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
ret = CAN_Reset (0);
if(ret != 0) {cout<<"操作失败！"; return;}
```

3.8 CAN_GetReceiveNum

描述

此函数用以获取接收缓冲区中接收到但尚未被读取的帧数量。

```
int __stdcall   CAN_GetReceiveNum(unsigned int devNum);
```

参数

devNum 设备通道号

返回值

返回 can 帧数量，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
ret = CAN_GetReceiveNum (0);
if(ret == -1) {cout<<"操作失败！"; return;}
cout <<"缓存 CAN 帧数"<< ret;
```

3.9 CAN_Transmit

描述

此函数用于发送 CAN 数据。

```
int __stdcall    CAN_Transmit(unsigned int devNum, P_Can_Msg
canmsg, unsigned int items);
```

参数

devNum 设备通道号

Canmsg 要发送的数据帧数组的首指针。

Items 要发送的数据帧个数，最大值为 100。

返回值

返回实际发送帧的数量。

示例：

```
#include "ucanbus.h"
int ret;
Can_Msg txmsg[100];
txmsg.ID  =  0x00000123;
txmsg.RemoteFlag  =  0;//数据帧
txmsg.ExternFlag  =  0;//标准帧
txmsg.DataLen  =  8;
*(int*)(&txmsg.Data[0]) = 0x11223344;
*(int*)(&txmsg.Data[4]) = 0x55667788;
ret = CAN_Transmit(0,txmsg,1);
```

3.10 CAN_Receive

描述

此函数用于接收 CAN 数据。

```
int __stdcall CAN_Receive(unsigned int devNum ,P_Can_Msg canmsg, int Len, unsigned int timeout);
```

参数

devNum 设备通道号

Canmsg 要接收的数据帧数组的首指针。

Len 要接收的数据帧个数，最大值 500。

Timeout 等待超时时间，以毫秒为单位。

返回值

返回实际发送帧的数量。

示例：

```
#include "ucanbus.h"
int ret;
ret = CAN_Receive (0,txmsg,1, 200);
cout <<"接收帧数："<< ret;
```

3.11 CAN_GetStatus

描述

此函数用于获取 CAN 设备状态，包括总线状态，错误信息。

```
int __stdcall CAN_GetStatus(unsigned int devNum,P_Can_Status status);
```

参数

devNum 设备通道号

status CAN 设备状态首指针。

返回值

为 0 表示操作成功，-1 表示操作失败。

示例：

```
#include "ucanbus.h"
int ret;
Can_Msg status;
ret = CAN_Receive (0, &status);
```

4 其他信息

暂无