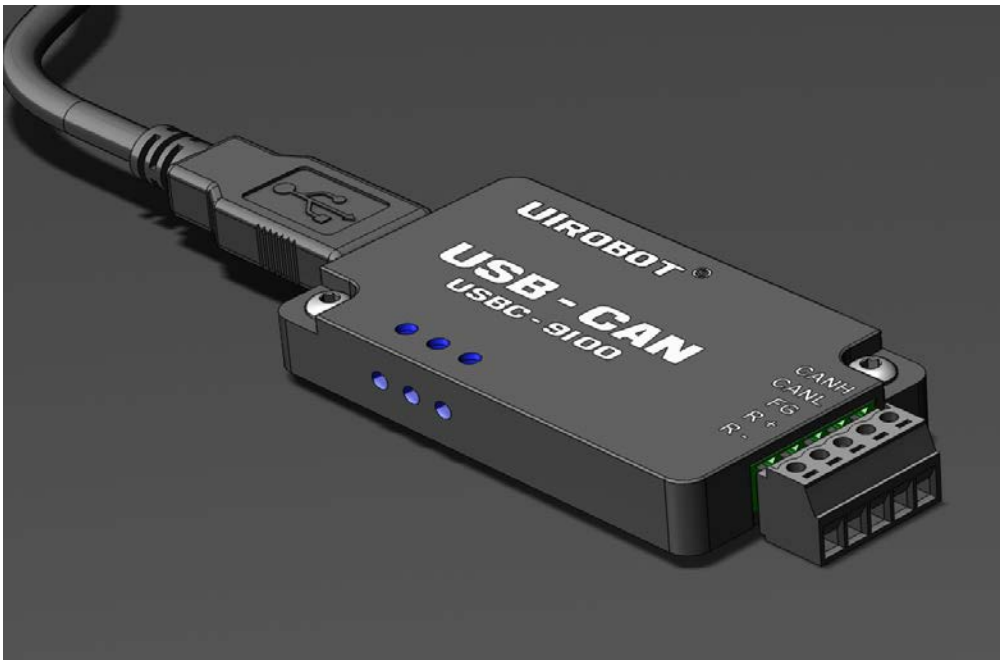
UIROBOT®
United Intelligence Robot Technology

使用手册

USBC 9100

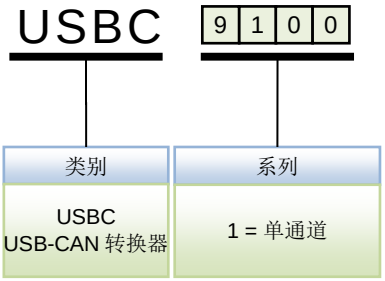
工业级智能 USB-CAN 总线转换器



[USBC9100 产品订购说明]

在订购 USBC9100 产品时请按以名下格式提供产品号，以便我们准确及时地为您提供产品：

USBC9100 产品牌号



USBC9100
USB CAN 工业级智能 CAN 转换器

性能指标

- 尺寸: 72mm * 36mm * 11mm
- 系统性能: 32 位处理器 48MIPS
- 帧流量: 业界最优性能，达到 CAN 的理论极限，实测每秒钟流量超过 6500 帧
- 传输方式: CAN 接口透明转换，兼容 CAN2.0A、CAN2.0B、CANOPEN 协议，USB 接口兼容 USB1.1 和 USB2.0 协议
- 通道数目: 1 路，可叠加使用，最多 100 台, 形成 100 通道
- 传输介质: 屏蔽或非屏蔽双绞线
- 传输速率: CAN 控制器波特率在 125Kbps～1Mbps 之间可选
- 通讯接口: 标准 CAN-bus 接口，起始端电阻自由配置
- 总线长度及节点数: 单路总线上最多可接 110 个节点，最长通讯距离 10 公里
- 供电形式: 使用 USB 总线电源，无需外部电源
- 占用资源: 即插即用，资源自动分配
- 工作温度: -40℃～+85℃
- 存储温度: -50℃～+105℃

简介

USBC9100 是兼容 USB1.1 和 USB2.0 总线，带有 1 路 CAN 接口的工业级智能型 CAN 数据接口卡采用 USBC9100 智能 CAN 转换器，PC 可以通过 USB 总线连接至 CAN 网络，构成实验室、工业控制、智能小区等 CAN 网络领域中数据处理、数据采集。

USBC9100 智能 CAN 转换器是 CAN 产品开发、CAN 数据分析的强大工具；同时，具有体积小、即插即用等特点，也是便携式系统用户的最佳选择。

USBC9100 智能 CAN 转换器上自带光电隔离模块，隔离电压达 2500V，使 USBC9100 智能 CAN 转换器避免由于地环流的损坏，增强系统在恶劣环境中使用的可靠性。

USBC9100 智能 CAN 转换器配有可在 Win9X/Me、Win2000/XP、Server 2003、Vista、Win 7 下工作的驱动程序，并提供 VB, VC、C++下的应用例程。

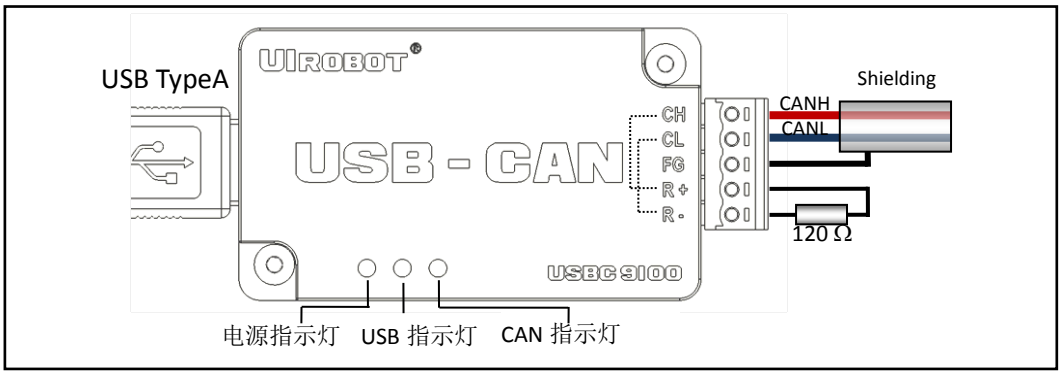
目录

1.0 设备安装

1.1 硬件接口描述

USBC9100 智能 CAN 转换器带有 1 路 CAN 通道，最多可同时叠加 100 台，可以用于连接一个 CAN-bus 网络或者 CAN-bus 接口的设备。USBC9100 智能 CAN 转换器接口布局如下：

图 0-1 USBC9100 转换器接线端子



CAN-bus 通道由 1 个 5Pin 接线端子引出。接线端子的引脚详细定义如表 0-1 所示。

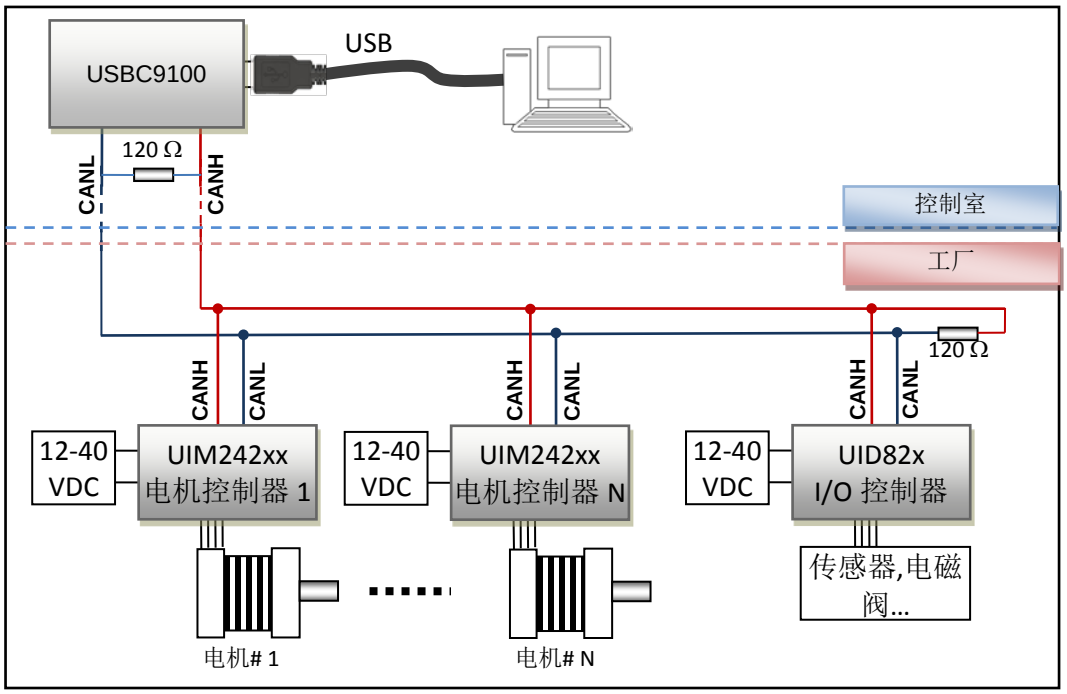
表 0-1 USBC9100 端口功能

引脚	标号	说明
1	CH	CANH 信号线
2	CL	CANL 信号线
3	FG	屏蔽线、地线（FG）
4	R+	终端电阻（内部连接到 CANH）
5	R-	终端电阻（内部连接到 CANL）

1.2 系统连接

USBC9100 智能 CAN 转换器和 CAN-bus 总线连接的时候，仅需要将 CANL 连 CANL，CANH 连 CANH 信号。CAN-bus 网络采用直线拓扑结构，总线的 2 个终端需要安装 120Ω的终端电阻；如果节点数目大于 2，中间节点不需要安装 120Ω的终端电阻。对于分支连接，其长度不应超过 3 米。CAN-bus 总线的连接见图 0-2 所示。

图 0-2 CAN-bus 网络的拓扑结构



1.2.1 终端电阻

为了增强 CAN 通讯的可靠性，CAN 总线网络的两个端点通常要加入终端匹配电阻，如图 0-2 所示。终端匹配电阻的值由传输电缆的特性阻抗所决定。例如双绞线的特性阻抗为 120Ω，则总线上的两个端点也应为 120Ω终端电阻。当 USBC9100 位于 CAN-bus 网络的一个端点上时，需要在外部端子上安装 120Ω终端电阻，即在"R-"引脚和"R＋"引脚接入终端电阻。

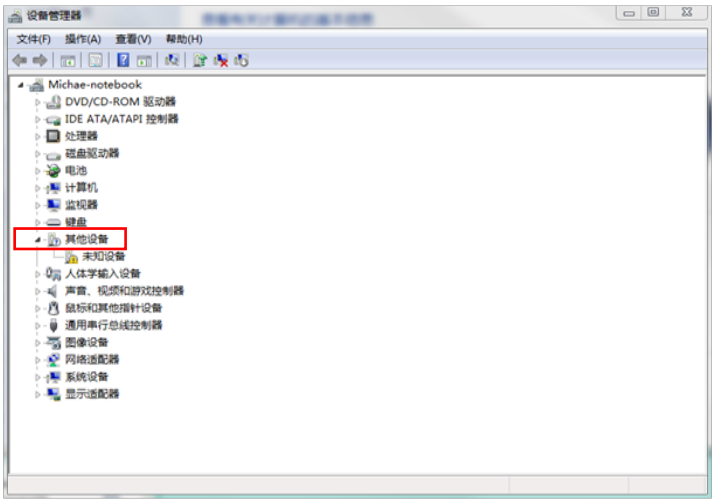
提示：当安装插件封装形式的终端电阻时，终端电阻两端的金属引脚不可短路在一起，否则会损坏设备。

1.3 驱动程序安装

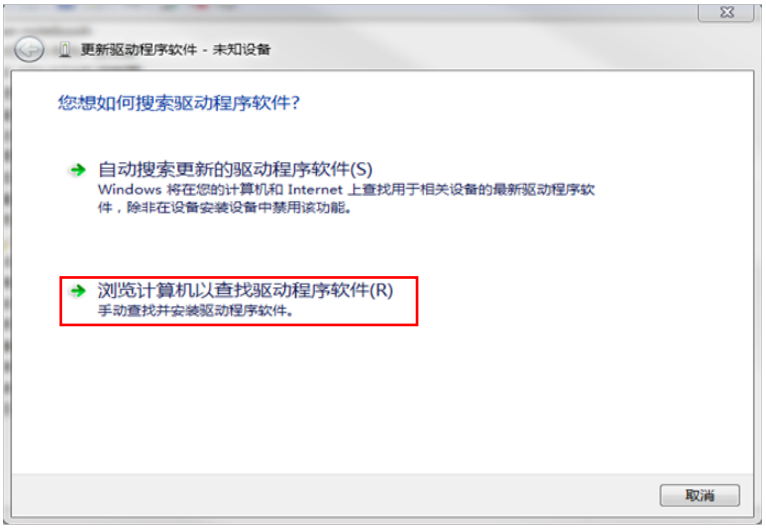
USBC 9100 智能 CAN 转换器使用 USB 直接供电并提供智能驱动安装包，安装步骤如下：

1) 将 USBC 9100 通过 USB 延长线连接至上位机；

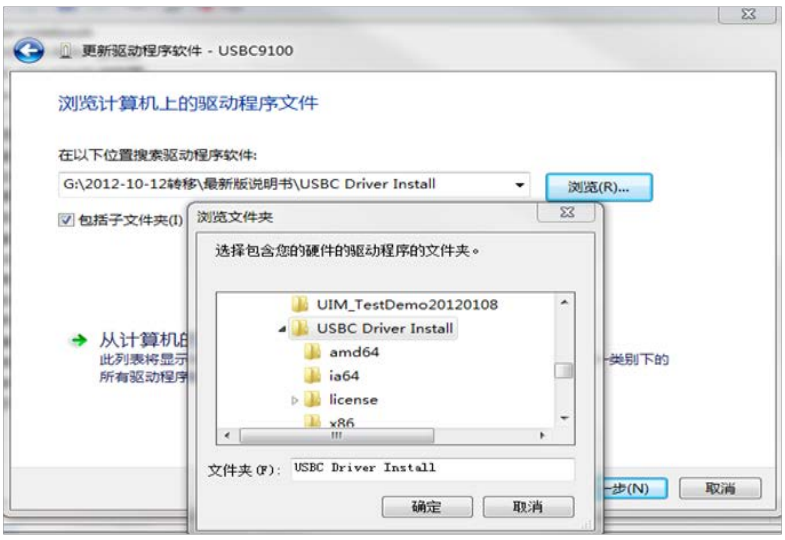
2) 打开设备管理器，弹出如下窗口：



3) 在“未知设备“处点击右键，选择“更新驱动程序软件”，弹出如下界面：



4) 选择上图所示的选项，在下图所示的浏览选项内找到驱动程序对应的文件夹，选择“USBC Driver Install”文件夹,点击“确定”，选择“下一步”：

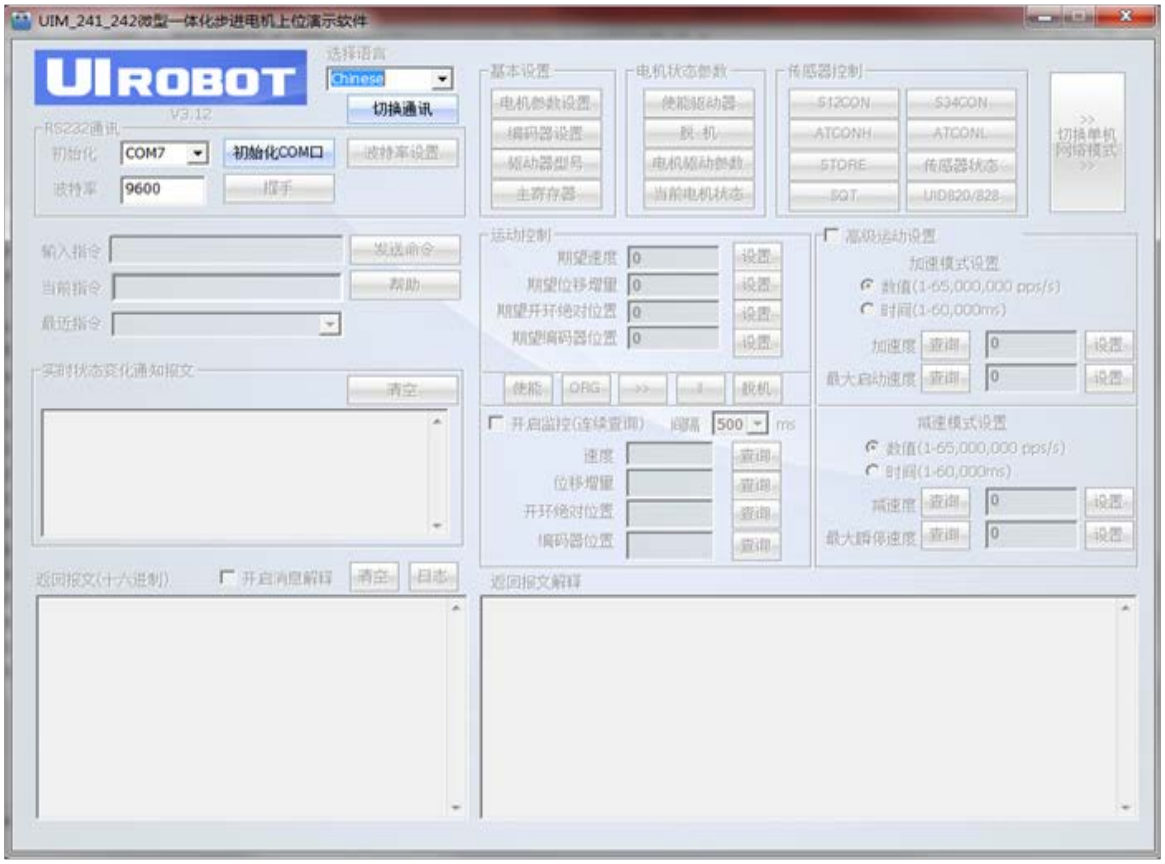


5) 程序安装成功后会弹出如下界面，至此，驱动程序安装完毕：

1.4 产品使用

2.1.1 测试软件概述

该测试软件使用极为方便，点击连接按钮后就可以发送和接受数据了，发送和接受的数据和状态在下面的信息提示框中有很清楚的显示。



2.1.2 测试软件详细介绍

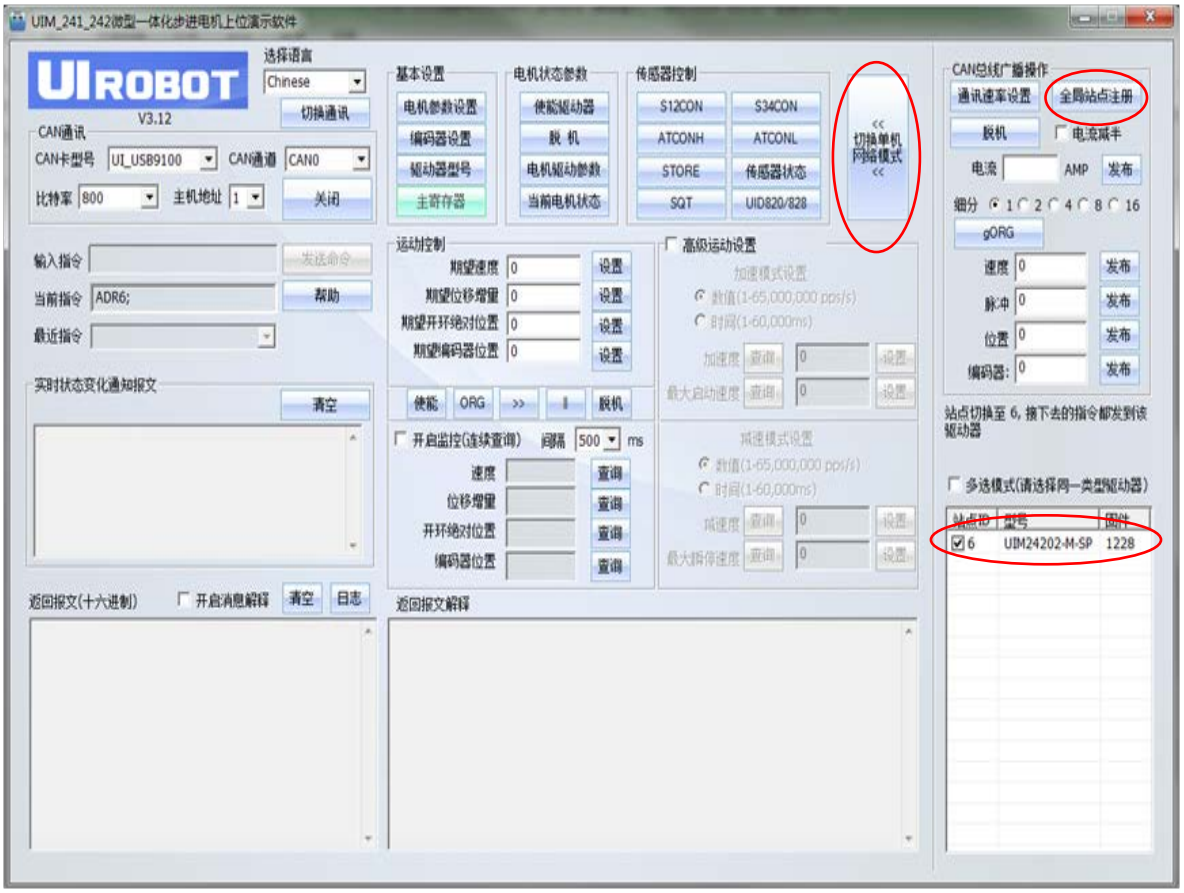
1) 搭建好 USBC 9100 及 UIM242 系列驱动控制器硬件平台，设备供电，不同设备之间需要共地；

2) 初次使用时点击“切换通讯”按钮，选择“CAN”，点击“确定”；

3) 依次选择 CAN 卡型号“UI_USBC9100”、CAN 通道“CAN0”、比特率“800”、主机地址“1”；

4) 点击“初始化 CAN”-----“切换单机网络模式”----“全局注册”，右下方的列表内即会出现当前的节点地址及型号，选中站点 ID 后即可对该节点进行控制；

5) 需要特殊说明的是，USBC9100 的 CAN 接口为完全透明转换，发送的数据帧的格式依照下位机通讯协议而定，此测试软件仅针对 UIRobot UIM242 系列下位机测试使用。



2.0 用户编程

用户如果只是利用 USBC9100 智能 CAN 转换器进行 CAN 总线通信测试，可以直接利用随机提供的测试软件，进行收发数据的测试。如果用户打算编写自己产品的软件程序。请认真阅读以下说明。UIRobot 提供基于 VC、C++的完整例程。

2.1 结构体定义

2.1.1 CAN_MSG_OBJ

功能：在 VCI_Transmit 和 VCI_Receive 函数中被用来传递 CAN 信息帧。

```
typedef struct _CAN_MSG_OBJ
{
    DWORD       ID;
    DWORD       Reserved0;
    BYTE        Reserved1;
    BYTE        SendType;
    BYTE        IDE;
    BYTE        RTR;
    BYTE        DataLen;
    BYTE        Data[8];
    BYTE        Reserved [3];
} CAN_MSG_OBJ, * P_CAN_MSG_OBJ;
```

ID	报文 ID = SID(11 位) | EID(18 位)。
Reserved0/1	保留，赋值为 0。
SendType	发送帧类型，只有在此帧为发送帧时有意义。
	0 = 正常发送
	1 = 自发自收
IDE	是否是远程帧。
	0 = 数据帧
	1 = 远程帧
RTR	是否是扩展帧。
	0 = 标准帧
	1 = 扩展帧
DataLen	表明 Data[8]数组内的字节数，长度不能超过 8。
Data	报文的数据。
	CAN 数据包原为 8 个字节，为了支持 RS232，次数据的长度增加为 128。
Reserved	系统保留。

2.1.2 CAN_CONFIG_OBJ

功能：定义了初始化 CAN 的配置。

```
typedef  struct  _ CAN_CONFIG_OBJ
{
    DWORD     AccCode;
    DWORD     AccMask;
    DWORD     Reserved;
    UCHAR     Filter;
    UCHAR     Timing0;
    UCHAR     Timing1;
    UCHAR     Mode;
} CAN_CONFIG_OBJ,*P_CAN_CONFIG_OBJ;
```

AccCode 验收码。

AccMask 屏蔽码。

Reserved 保留。

Filter 滤波方式，必须为 0。

Timing0 定时器 0。

Timing1 定时器 1。

Mode 模式，必须为 0。

备注：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

2.2 接口函数说明

2.2.1. 查找所有在线的 USBC 设备

USBCAN_API DWORD USBC_ListDevices(PDWORD pDevIndexList);

pDevIndexList 存放设备索引号列表的指针。

返回值 找到的设备数量。

示例

```
USBC_DevIndex_Count = USBC_ListDevices ((PDWORD)&USBC_DevIndex_List[0]);

if ( USBC_DevIndex_Count == 0)
{
      printf ("USBC: No USBC9100 device is found!\n");

      return;
}
else
{
      printf("USBC: Number of USBC9100 found >>> [ %d ] \n", USBC_DevIndex_Count);

      printf("\nUSBC: USBC Device Index List:\n");

      printf("============================================================\n\n");
           for (i=0;i<USBC_DevIndex_Count;i++)
           {
                     printf(" No.%d >>> Device Index = % d \n", i+1, USBC_DevIndex_List[i]);
           }
      printf("\n============================================================\n\n");

      printf("USBC: Please make sure all device indexes are unique! \n");

      printf("USBC: If any two device indexes are overlapped, please use
            USBC_SetDeviceIndex() to assign a new index number to one of the above
            devices.\n");
}
```

2.2.2. 为 USBC 分配设备索引号

USBCAN_API DWORD USBC_SetDevicesIndex(DWORD dwDevIndex);

dwDevIndex 期望的设备索引号。

返回值 为 1 表示操作成功，为-1 表示操作失败。

示例

```
printf("Press any key to start Device Index Assignment (Only one USBC should be plugged
        in) ...");

_getch();

printf("\nPlease type the Device Index Number to be assigned >>> ");

s = _getch();

printf("\n\n");

DevIndex = (BYTE) atoi(&s);

USBC_DevIndex_Count = USBC_SetDevicesIndex(DevIndex);

If ( USBC_DevIndex_Count == 0)
{
    printf("USBC: FAILED, No USBC9100 was found! \n");

    return;
}
else if ( USBC_DevIndex_Count == 1)
{
    printf("USBC: SUCCESS, USBC9100 Device Index is set to %d！ \n", DevIndex);
}
else
{
    printf("USBC: FAILED, %d USBC9100s found, please keep only one USBC9100
            plugged in. \n", USBC_DevIndex_Count);
}

return;
```

2.2.3. 打开 USBC 设备

USBCAN_API DWORD USBC_OpenDevice(DWORD DevIndex, DWORD dwReserved);

DevIndex 设备索引号。

返回值 为 1 表示操作成功，-1 表示操作失败。

示例

```
printf("\nPlease select a device index to start  >>> ");

s = _getch();

printf("\n\n");

DevIndex = (BYTE) atoi(&s);

if( USBC_OpenDevice(DevIndex,0) == -1)
{
	printf("USBC: OPEN DEVICE %d FAILED!\n",DevIndex);

	goto ending;
}
else
{
	printf("USBC: OPEN DEVICE %d SUCCESS!\n",DevIndex);
}
```

2.2.4. 关闭 USBC 设备

USBCAN_API DWORD USBC_CloseDevice(DWORD dwDevIndex);

dwDevIndex 设备索引号。

返回值 为 1 表示操作成功，-1 表示操作失败。

示例

```
BYTE    DevIndex = 1;

USBC_CloseDevice(DevIndex);
```

2.2.5. 初始化 USBC 设备的 CAN 模块

USBCAN_API DWORD USBC_InitCan (DWORD dwDevIndex, DWORD dwDevPort, P_CAN_CONFIG_OBJ pInitConfig);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

pInitConfig 配置参数结构。

返回值 为 1 表示操作成功，-1 表示操作失败。

备注 初始化后，需要用 USBC_StartCan()开启 CAN 模块。

示例

```
InitConfig.AccCode = 0x08004001;

InitConfig.AccMask = 0x1FFFFFFF;

InitConfig.Filter       = 0;

InitConfig.Mode       = 0;

InitConfig.Timing0   = 1;

InitConfig.Timing1   = 0;

if( USBC_InitCan(DevIndex, 0, &InitConfig) == 1)
{
      printf("\nUSBC: Initial No [ %d ] USBC Device CAN Module SUCCESS!\n", DevIndex);
}
else
{
      printf("\nUSBC: Initial No [ %d ] USBC Device CAN Module FAILED!\n", DevIndex);
}
```

2.2.6. 关闭 USBC 设备的 CAN 模块

USBCAN_API DWORD USBC_ResetCan(DWORD dwDevIndex, DWORD dwDevPort);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

返回值 为 1 表示操作成功，-1 表示操作失败。

示例

```
if( USBC_ResetCan(DevIndex, 0) == 1)
{
    printf("\nUSBC: Reset No [ %d ] USBC Device CAN Module SUCCESS!\n", DevIndex);
}
else
{
    printf("\nUSBC: Reset No [ %d ] USBC Device CAN Module FAILED!\n", DevIndex);
}
```

2.2.7. 开启 USBC 设备的 CAN 模块

USBCAN_API DWORD USBC_StartCan(DWORD dwDevIndex, DWORD dwDevPort);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

返回值 为 1 表示操作成功，-1 表示操作失败。

示例

```
if( USBC_StartCan(DevIndex,0) == 1)
{
    printf("\nUSBC: Start No [ %d ] USBC Device CAN Module SUCCESS!\n", DevIndex);
}
else
{
    printf("\nUSBC: Start No [ %d ] USBC Device CAN Module FAILED!\n", DevIndex);
}
```

2.2.8. 发送 CAN 报文

USBCAN_API DWORD USBC_Transmit(DWORD dwDevIndex, DWORD dwDevPort, P_CAN_MSG_OBJ pSend, ULONG ulSize);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

pSend 发送 CAN 报文参数结构（可以是数组）。

ulSize 需要发送的 CAN 报文帧数。

返回值 成功发送的帧数，为-1 表示操作失败。

示例

```
SndNum =53;

for (i =0; i<400; i++)
{
	Test_CANSnd[i].ID = 0x01280007;		//SID=0000 0001 0010 10
						//EID = 00 0000 0000 0000 0111
						//SPD=1000

	Test_CANSnd[i].IDE	= 1;

	Test_CANSnd[i].DataLen = 4;

	Test_CANSnd[i].Data[0]	= 0xE8;		//0x 00 00 03 e8 (hex) = 1000 (bcd)

	Test_CANSnd[i].Data[1]	= 0x03;

	Test_CANSnd[i].Data[2]	= 0;

	Test_CANSnd[i].Data[3]	= 0;

	Test_CANSnd[i].Data[4]	= 5;

	Test_CANSnd[i].Data[5]	= 6;

	Test_CANSnd[i].Data[6]	= 7;

	Test_CANSnd[i].Data[7]	= 8;
}

T0 = timeGetTime();

retNum = USBC_Transmit(DevIndex, 0, &Test_CANSnd[0], SndNum);

T1 = timeGetTime() - T0;

if( retNum == -1)
{
	printf("\nUSBC: Send message to No.[%d] device FAILED!\n",DevIndex,SndNum);

	printf(" >>>> Number of message sent >>> [ %d ]\n", retNum);

	printf(" >>>> Time lapse >>> [ %d ]milliseconds\n", T1);
}
else
{
	printf("\nUSBC: Send message to No.[ %d ] device SUCCESSED!\n", DevIndex, SndNum);
}
```

2.2.9. 接收 USBC 内缓存的 CAN 报文

USBCAN_API DWORD USBC_Recieve (DWORD dwDevIndex, DWORD dwDevPort, P_CAN_MSG_OBJ pReceive, ULONG ulSize, UINT uiWaitTime);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

pReceive 接收 CAN 报文参数结构（可以是数组）。

ulSize 期望接收的 CAN 报文帧数。

uiWaitTime 等待超时定时器（单位：毫秒）。

返回值 成功接收的帧数，为-1 表示操作失败。

示例

```
rcvNum = retNum;

T0 = timeGetTime();

retNum = USBC_Recieve(DevIndex, 0, &Test_CANRcv[0], rcvNum, 2000);

T2 = timeGetTime() - T0;

if( retNum == -1)
{
    printf("\nUSBC: Receive message from No.[%d] device FAILED!\n", DevIndex);
}
else
{
    printf("\nUSBC: Receive message from No.[%d] device SUCCESSED!\n", DevIndex);

    printf("\n============================================================\n");

    printf("MSG No   ID          Data      From ID  To ID    DataLen\n\n");

    for (i =0; i< retNum; i++)
    {
        from_id=((Test_CANRcv[i].ID>>8)&0x3E0)+(Test_CANRcv[i].ID>>19)&0x01F;

        to_id=((Test_CANRcv[i].ID>>3)&0x3E0)+ Test_CANRcv[i].ID>>24)&0x01F;

        printf("%d  0x%lx   %x %x %x %x %x %x %x %x     %d       %d        %d\n",
                i+1,
                Test_CANRcv[i].ID,
                Test_CANRcv[i].Data[0],
                Test_CANRcv[i].Data[1],
                Test_CANRcv[i].Data[2],
                Test_CANRcv[i].Data[3],
                Test_CANRcv[i].Data[4],
                Test_CANRcv[i].Data[5],
                Test_CANRcv[i].Data[6],
                Test_CANRcv[i].Data[7],
                from_id,
                to_id,
                Test_CANRcv[i].DataLen
                );
    }

    printf("\n============================================================\n");

    printf("\n >>>> Number of message received >>> [ %d ]\n", retNum);
```

2.2.10. 查询 USBC 内等待 HOST 接受的报文数量

USBCAN_API DWORD USBC_GetReceiveNum(DWORD dwDevIndex, DWORD dwDevPort);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

返回值 等待接收的帧数，为-1 表示操作失败。

示例

```
retNum = USBC_GetRecieveNum(DevIndex, 0);

if( retNum == -1)
{
    printf("\nUSBC: Get number of message waiting to receive of No.[ %d ] device
        FAILED!]\n", DevIndex);
}
else
{
    printf("\nUSBC: Get number of message waiting to receive of No.[ %d ] device
        SUCCESSED!]\n", DevIndex);

    printf(" >>>> The number of message waiting to receive is [ %d ]\n",retNum);
}
```

2.2.11. 清空 USBC 设备的 CAN 模块的接收缓冲区

USBCAN_API DWORD USBC_ClearBuffer(DWORD dwDevIndex, DWORD dwDevPort);

dwDevIndex 设备索引号。

dwDevPort 该设备的 CAN 端口号。

返回值 为 1 表示操作成功，为-1 表示操作失败。

示例

```
if( USBC_ClearBuffer(DevIndex, 0) == -1)
{
    printf("\nUSBC: Clear buffer of No.[ %d ] device FAILED!\n", DevIndex);
}
else
{
    printf("\nUSBC: Clear buffer of No.[ %d ] device SUCCESSED!\n ", DevIndex);
}
```

2.3 接口库函数使用方法

首先，把库函数文件都放在工作目录下。

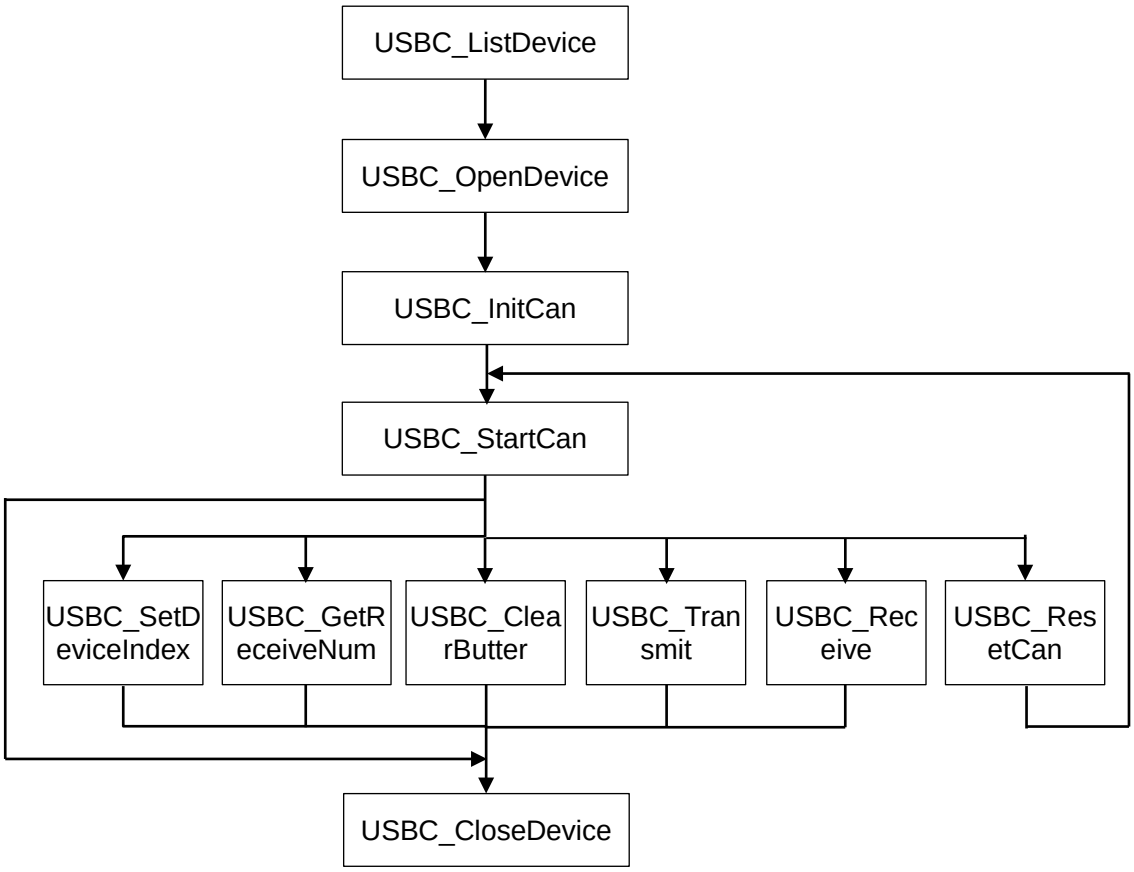
– VC 调用动态库的方法

1) 在扩展名为.CPP 的文件中包含 USBC9100.h 头文件；

如：#include”USBC9100.h”

2) 在工程的连接器设置中连接到 Winmm.lib 文件。

2.4 接口库函数使用流程



附录A CAN2.0B标准帧

CAN 标准帧信息为 11 个字节，包括两部分：信息和数据部分。前 3 个字节为信息部分。

	7	6	5	4	3	2	1	0
字节 1	FF	RTR	X	X	DLC（数据长度）			
字节 2	（报文识别码）				ID.10-ID.3			
字节 3	ID.2-ID.0			X	X	X	X	X
字节 4	数据 1							
字节 5	数据 2							
字节 6	数据 3							
字节 7	数据 4							
字节 8	数据 5							
字节 9	数据 6							
字节 10	数据 7							
字节 11	数据 8							

字节 1 为帧信息。第 7 位（FF）表示帧格式，在标准帧中，FF＝0；第 6 位（RTR）表 示帧的类型，RTR=0 表示为数据帧，RTR=1 表示为远程帧；DLC 表示在数据帧时实际的数据长度。字节 2、3 为报文识别码，11 位有效。字节 4～11 为数据帧的实际数据，远程帧时无效。

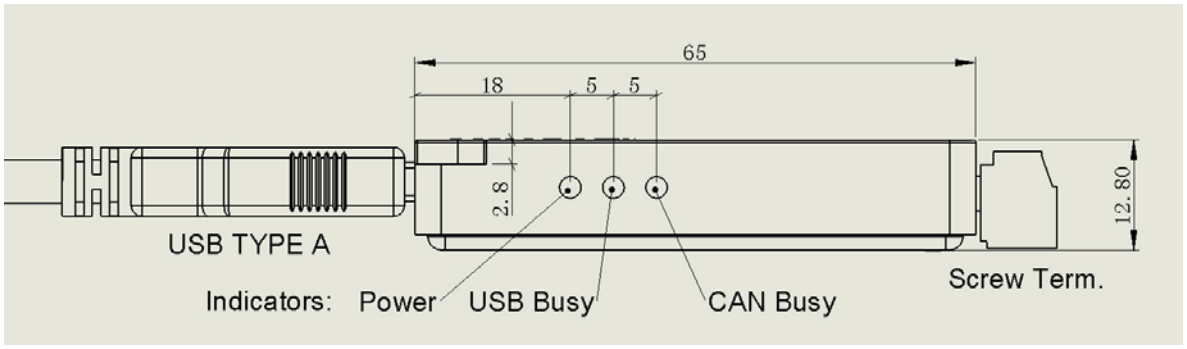
附录B CAN2.0B扩展帧

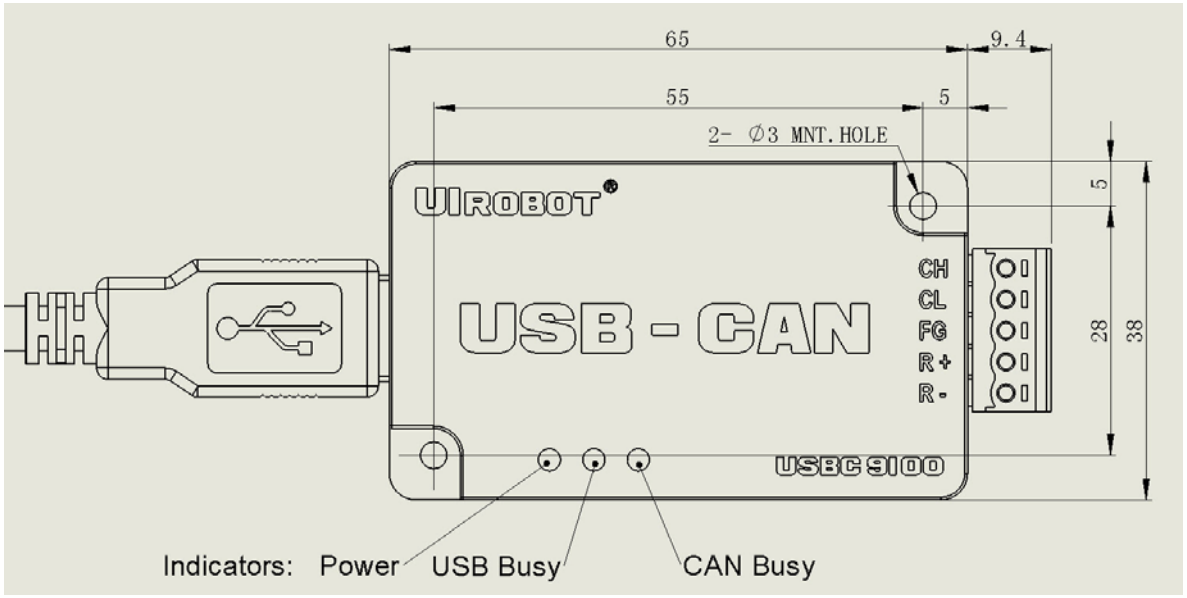
CAN 扩展帧信息为 13 个字节，包括两部分：信息和数据部分。前 5 个字节为信息部分。

	7	6	5	4	3	2	1	0
字节 1	FF	RTR	X	X	DLC（数据长度）			
字节 2	（报文识别码） ID.28-ID.21							
字节 3	ID.20-ID.13							
字节 4	ID.12-ID.5							
字节 5	ID.4-ID.0					X	X	X
字节 6	数据 1							
字节 7	数据 2							
字节 8	数据 3							
字节 9	数据 4							
字节 10	数据 5							
字节 11	数据 6							
字节 12	数据 7							
字节 13	数据 8							

字节 1 为帧信息。第 7 位（FF）表示帧格式，在扩展帧中，FF＝1；第 6 位（RTR）表 示帧的类型，RTR=0 表示为数据帧，RTR=1 表示为远程帧；DLC 表示在数据帧时实际的数据长度。字节 2~5 为报文识别码，其高 29 位有效。字节 6～13 为数据帧的实际数据，远程帧时无效。

附录C　外形尺寸图


65
18
5
5
2.8
12.80
USB TYPE A
Screw Term.
Indicators:
Power
USB Busy
CAN Busy


65
9.4
55
5
2- ∅3 MNT.HOLE
UIROBOT
USB - CAN
CH
CL
FG
R+
R-
5
28
38
USBC9100
Indicators:
Power
USB Busy
CAN Busy

附录D　转换器安装示意图