SLCAN API

Руководство пользователя

Версия 1.2

28.05.2017

Введение

Данное руководство предназначено для программистов, создающих программное обеспечение, которое использует USB-CAN конвертеры производства «НПП «Славна», «НПП «Славна-Спектр», «Лаборатория СК».

Руководство содержит полное описание состава и функций библиотеки SLCAN, а также описание порядка ее использования. Руководство содержит фрагменты исходного кода примеров использования библиотеки. Руководство также содержит полный исходный код интерфейсных модулей.

Библиотека SLCAN

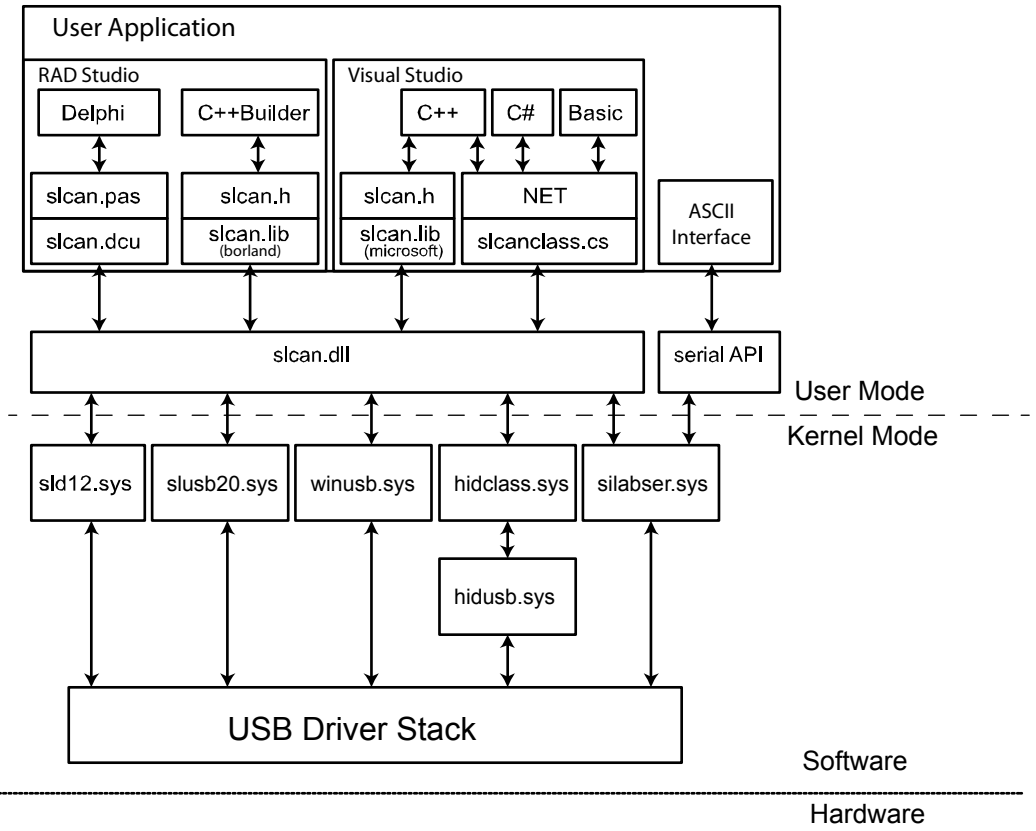
Назначение

Основное назначение библиотеки SLCAN – это поддержка разработки программного обеспечения, использующего USB-CAN конвертеры производства «НПП «Славна», «НПП «Славна-Спектр» и «Лаборатория СК».

Состав

Библиотека состоит собственно из файла динамической библиотеки windows slcan.dll. Также в состав библиотеки входят файлы, обеспечивающие ее использование в различных средах программирования. Для C и C++ это заголовочный файл slcan.h и файлы библиотек динамической компоновки slcan.lib для Microsoft C и для Borland C. Для pascal (Delphi) модуль slcan.pas. Для использования в NET служит файл slcanclass.cs (C#), реализующий статический класс-«обертку» для использования функций библиотеки. Устройства, работающие через виртуальный драйвер COM порта, поддерживают также ASCII интерфейс.

Место библиотеки slcan при разработке программного обеспечения демонстрирует рисунок.



Использование

Для разработки приложений библиотека предоставляет полный набор функций, которые позволяют использовать все возможности USB-CAN конвертеров по приему и передачи CAN сообщений. Библиотека также обеспечивает пользователя функциями для управления работой конвертера, настройки CAN контроллера, мониторинга работы CAN шины, диагностики и идентификации ошибок CAN сети.

В библиотеке также реализован механизм автоматического отслеживания подключения и отключения конвертеров от компьютера. Механизм формирует список подключенных устройств и генерирует вызовы callback функций при изменении этого списка, а также при изменении состояния каждого отдельного устройства, представляющим собой элемент списка.

Ниже приведен весь список функций, сгруппированный по функциональности.

Функции для работы собственно с библиотекой

SlCan_Load	Загружает библиотеку. Устанавливает callback функции. Инициализирует внутренние переменные.
SlCan_Free	Выгружает библиотеку.
SlCan_Update	Обновляет список устройств.
SlCan_GetDeviceCount	Получает количество устройств в системе.
SlCan_GetDevice	Получает ссылку на устройство.

Базовые функции для работы с USB-CAN конвертером

SlCan_DeviceGetProperty	Получает свойства устройства.
SlCan_DeviceGetCapabilities	Получает возможности устройства.
SlCan_DeviceOpen	Открывает устройство.
SlCan_DeviceClose	Закрывает устройство.
SlCan_DeviceSetBitrate	Устанавливает скорость передачи.
SlCan_DeviceGetBitrate	Получает скорость передачи.
SlCan_DeviceSetMode	Устанавливает режим работы устройства.
SlCan_DeviceGetMode	Получает режим работы устройства.
SlCan DeviceWriteMessages	Передает CAN фреймы в сеть.
SlCan_DeviceReadMessages	Читает принятые CAN фреймы.

Расширенные функции работы с USB-CAN конвертером

SlCan_DeviceGetState	Получает состояние устройства.
SlCan_DeviceSetTXTimeOut	Устанавливает таймаут передачи.
SlCan_DeviceGetTXTimeOut	Получает таймаут передачи.
SlCan_DeviceEnaRec	Разрешает или запрещает прием фреймов или событий
SlCan_DeviceSetLatency	Устанавливает значение задержки передачи неполного USB пакета.
SlCan_DeviceGetLatency	Получает значение задержки передачи неполного USB пакета
SlCan_DevicePurge	Очищает входной и выходной буферы, а так же может прервать текущую передачу и прием данных
SlCan_DeviceSetEventLevel	Устанавливает уровень генерации событий устройством.
SlCan_DeviceGetEventLevel	Получает уровень генерации событий устройством.
SlCan_DeviceSetExMode	Устанавливает расширенный режим работы устройства
SlCan_DeviceGetExMode	Получает значение расширенного режима работы устройства
SlCan_DeviceWriteMessgesEx	Функция расширенной записи CAN фреймов в устройство для передачи.
SlCan_DeviceReadEvents	Читает события, генерируемые устройством.

SlCan_DeviceSetStartTimeStamp	Устанавливает начальное значение счетчика временных меток.
SlCan_DeviceGetStartTimeStamp	Получает начальное значение счетчика временных меток.
SlCan_DeviceGetTimeStamp	Получает текущее значение счетчика временных меток.
SlCan_DeviceSetTimeStampPeriod	Устанавливает значение периода генерации временных меток.
SlCan_DeviceGetTimeStampPeriod	Получает значение периода генерации временных меток.
SlCan_DeviceSetContext	Устанавливает контекст устройства.
SlCan_DeviceGetContext	Получает указатель на контекст устройства
SlCan_DeviceGetContextSize	Получает размер контекста устройства.
SlCan_DeviceSetAlias	Устанавливает псевдоним устройства.
SlCan_DeviceGetAlias	Получает псевдоним устройства

Порядок использования.

Программа, использующая библиотеку, должна быть запущена с правами администратора.

Порядок использования функций библиотеки следующий:

1. Инициализировать библиотеку с помощью функции SlCan_Load.
2. Получить ссылку на нужное устройство вызовом функции SlCan_GetDevice.
3. Открыть устройство функцией SlCan_DeviceOpen.
4. Настроить скорость передачи с помощью функции SlCan_DeviceSetBitRate.
5. Установить режим работы функцией SlCan_DeviceSetMode.
6. Теперь можно принимать и передавать сообщения с помощью функций SlCan_DeviceReadMessages и SlCan_DeviceWriteMessages.
7. Закрыть устройство функцией SlCan_DeviceClose.
8. Деинициализироать библиотеку функцией SlCan_Free.

Простейший пример использования показывает следующая программа на Visual C (проект **hello**):

```
#include "stdafx.h"
#include <slcan.h>

int _tmain(int argc, _TCHAR* argv[])
{
    BOOL b;
    ULONG devicecount;
    HSLCAN hDevice;
    SLCAN_BITRATE br;
    SLCAN_MESSAGE OutMsg;
    SLCAN_MESSAGE InMsg[100];
    int i;
    BYTE WriteStatus;
    ULONG ReadCount;

    // Загрузим библиотеку. Необходимо это сделать один раз
    // до вызова любой другой функции библиотеки.
    b=SlCan_Load(NULL,NULL);
    if (!b) return 0;
    // Выясним, сколько устройств доступно
    devicecount = SlCan_GetDeviceCount();
    if (!devicecount) return 0;
```

```
    // Получим ссылку на первое устройство
    hDevice = SlCan_GetDevice(0);
    if (hDevice==INVALID_HANDLE_VALUE) return 0;

    // Откроем устройство.
    b = SlCan_DeviceOpen(hDevice);
    if (!b) return 0;

    // При открытии устройства устанавливается максимально возможная
    // скорость передачи. Но мы установим 125kb/s
    br.BRP = SLCAN_BR_CIA_125K;
    b = SlCan_DeviceSetBitRate(hDevice,&br);
    if (!b) return 0;

    // Установим режим самоприёма
    b = SlCan_DeviceSetMode(hDevice,SLCAN_MODE_LOOPBACK);
    if (!b) return 0;

    // Передадим один расширенный фрейм с данными,
    // предварительно проинициализировав его
    OutMsg.ID = 0xABCDEF;
    OutMsg.Info = SLCAN_MES_INFO_EXT;
    OutMsg.DataCount = 8;
    for (i = 0;i<OutMsg.DataCount;i++)  OutMsg.Data[i]=(BYTE)i;
    b=SlCan_DeviceWriteMessages(hDevice,&OutMsg,1,&WriteStatus);
    if (!b) return 0;

    // Проверим статус передачи
    if (WriteStatus!=SLCAN_TX_STATUS_OK) return 0;

    // Прочитаем принятые сообщения с ожиданием 1000 ms
    b=SlCan_DeviceReadMessages(hDevice,1000,InMsg,100,&ReadCount);
    if (!b) return 0;
    if (ReadCount)
        printf("read %d messages",ReadCount);

    // Выгрузим  библиотеку.
    b=SlCan_Free(FALSE);
    return 0;
}
```

Выбор устройства

Библиотека при инициализации функцией SlCan_Load формирует список подключенных к компьютеру и доступных для работы устройств. Этот список обновляется автоматически при использовании callback функций или вручную с помощью функции SlCan_Update. Количество устройств возвращает функция SlCan_GetDeviceCount.

Необходимое для работы устройство может быть выбрано с помощью строковых свойств, например уникального серийного номера, псевдонима и др. Значение свойства может быть получено с помощью функции SlCan_DeviceGetProperty.

Ниже приведен пример реализации метода класса SlCan выбора устройства по его серийному номеру

```
public static IntPtr GetDeviceBySerial(string serial)
{
    for (uint i = 0; i < GetDeviceCount(); i++)
    {
        IntPtr hD = GetDevice(i);
        if (DeviceGetProperty(hD, PROPERTY_INDEX_SERIAL) == serial)
           return hD;
    }
    return (IntPtr)INVALID_HANDLE_VALUE;
}
```

Для удобства выбора устройств в пользовательском программном обеспечении библиотека вводит понятие псевдонима устройства. Псевдоним устройства это строка, которая однозначно связана с конкретным экземпляром устройства в системе. Значение псевдонима хранится в реестре. Библиотека предоставляет функции SlCan_DeviceSetAlias и SlCan_DeviceGetAlias для установки значения псевдонима и его получения соответственно.

Callback функции

При инициализации библиотеки функцией SlCan_Load в качестве параметров могут быть указаны адреса функций типа SlCan_DeviceCallBack и SlCan_DeviceListCallBack. Данные функции используются ядром библиотеки при обработке сообщения WM_DEVICE, которое генерируется системой при подключении и отключении plug and play устройств, которыми являются USB – CAN конвертеры.

Ядро библиотеки при обработке сообщений данного типа автоматически управляет windows ссылками на драйверы устройств, что позволяет избегать в многопоточной среде операций ввода-вывода с «мертвыми» ссылками, которые часто приводят к «зависанию» программ.

При изменении состояния устройства ядро вызывает данные функции, указывая в параметрах ссылку на устройство и состояния статуса его подключения. Пользователь может использовать данные функции для собственных нужд.

Проект **callback** показывает пример такого использования.

Прием и передача CAN фреймов

Типичное применения USB-CAN конвертеров предполагает его использование для приема и передачи CAN фреймов. Для этого библиотека предоставляет функции чтения принятых устройством CAN сообщений SlCan_DeviceReadMessages и записи сообщений в устройство для передачи SlCan_DeviceWriteMessages. Также библиотека поддерживает передачу TTCAN сообщений с помощью функции SlCan_DeviceWriteMessagesEx, если это позволяет устройство.

При этом следует отметить, что процедуры приема и передачи дважды буферизованы, т.е. реально функции записи и чтения работают с буферами драйвера, а драйвер и устройство самостоятельно обмениваются данными между буферами драйвера и устройства, причем это производится независимым образом.

Для настройки скорости передачи CAN контроллера устройства используется функция SlCan_DeviceSetBitRate, при этом могут быть установлены как предопределенные значения, так и пользовательские, если это поддерживает устройство.

Настройка скорости может быть проведена только в конфигурационном режиме, который устанавливается функцией SlCan_DeviceSetMode с параметром SLCAN_MODE_CONFIG. Данный режим устанавливается автоматически при открытии устройства.

Для передачи устройство должно находиться в режиме SLCAN_MODE_NORMAL или SLCAN_MODE_LOOPBACK. Настройка таймаута передачи производится функцией SlCan_DeviceSetTXTimeOut.

Устройство начинает принимать фреймы сразу же после установки отличного от SLCAN_MODE_CONFIG режима работы. Принятые фреймы сохраняются в буфере устройства и автоматически передаются из него в буфер драйвера. Временной лаг такой передачи может быть настроен с помощью SlCan_DeviceSetLatency.

При использовании функции чтения сообщений SlCan_DeviceReadMessages рекомендуется использовать уровень генерации событий SLCAN_EVT_LEVEL_RX_MSG, который устанавливается по умолчанию при открытии устройства.

Анализатор CAN шины

Создание анализаторов CAN сети предъявляет определенные требования к возможностям CAN конвертеров. К таким возможностям можно отнести генерацию событий в ответ на изменения состояния CAN контроллера, прием и передачу CAN фреймов, изменения счетчиков ошибок, диагностики самих ошибок и их типов. Также необходимо обеспечить измерение времени появления событий и передачу данных об этих событиях в компьютер. Наши конвертеры обладают такими возможностями, при этом объем этих возможностей у разных моделей может быть разный. Библиотека предоставляет набор функций для использования таких возможностей при создании пользовательского программного обеспечения.

Набор возможностей конвертера по генерации событий определяет поле *bMaxEventLevel* структуры SlCan_DeviceCapabilities. Сама структура может быть получена с помощью функции

SlCan_DeviceGetCapabilities. Текущий уровень генерации событий устанавливается функцией SlCan_DeviceSetEventLevel.

События, генерируемые конвертером, могут быть прочитаны функцией SlCan_DeviceReadEvents. Функция читает события типа SlCan_Event. События содержат информацию о принятых и переданных фреймах, изменения состояния CAN контроллера, изменениях счетчиков ошибок, ошибках сети при приеме и передаче, типе ошибке, месте CAN фрейма, где произошла ошибка, потери арбитража и бите, в котором это произошло.

Все события снабжены временной меткой. Разрядность временной метки 64 бита, при этом младшие 24 бита содержит в себе структура события любого типа. При изменении старших 40 бит генерируется собственный тип события. Генератор временных меток работает во время рабочего режима работы конвертера, инкрементируя счетчик временных меток. Счётчик устанавливается в начальное значение в момент перехода в рабочий режим. По умолчанию начальное значение счетчика равно 0, но может быть изменено с помощью функции SlCan_DeviceSetStartTimeStamp.

Период генератора временных меток по умолчанию равен 1 мкс, но может быть установлен в другое значение с помощью функции SlCan_DeviceSetTimeStampPeriod. Единицей измерения периода может быть микросекунда или длительность бита текущей скорости передачи.

Проект **simplemonitor** демонстрирует пример реализации простой программы анализатора CAN сети.

Примеры

Комплект поставки библиотеки slcan содержит примеры программ, использующих библиотеку. Программы созданы в средах Embarcadero RAD Studio – Delphi и C++ Builder и Microsoft Visual Studio – Visual C++, Visual C# и Visual Basic.

Проект	Описание	Delphi	C++ Builder	Visual C++	Visual C#	Visual Basic
hello	Консольное приложение, демонстрирующее порядок использования библиотеки.	Да	Да	Да	Да	Да
callback	Оконное приложение, демонстрирующее использование callback функций	Да	Нет	Нет	Да	Нет
slcanterm	Консольное приложение – терминал для работы с конвертерами по приему и передаче сообщений	Да	Да	Да	Да	Нет
simpleio	Оконное приложение – прием и передача CAN сообщение.	Нет	Нет	Нет	Да	Да
simplemonitor	Оконное приложение – анализатор CAN сети.	Да	Нет	Нет	Да	Да

Функции

SlCan_Load

Функция загружает и инициализирует библиотеку slcan, устанавливая callback функции

C

```
SLCANAPI BOOL STDCALL SlCan_Load(
   SLCAN_DEVICE_CALLBACK DeviceProc,
   SLCAN_DEVICELIST_CALLBACK DeviceListProc
);
```

Pascal

```
function    SlCan_Load(
   DeviceProc:TSlCan_DeviceCallBack;
   DeviceListProc:TSlCan_DeviceListCallBack
   ):longbool;stdcall;
```

C#

```
public static extern bool Load(DeviceCallBack DeviceProc, DeviceListCallBack
DeviceListProc);
```

VB

```
Public Function Shared Load(ByVal DeviceProc As DeviceCallBack, ByVal DeviceListProc
As DeviceListCallBack) As Boolean
```

Параметры

DeviceProc

Указатель на callback функцию изменения состояния устройства типа **SlCan_DeviceCallBack**. Может принимать значение NULL.

DeviceListProc

Указатель на callback функцию, которая вызывается при изменении списка устройств типа **SlCan_DeviceListCallBack**. Может принимать значение NULL.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Если хотя бы один из параметров не равен NULL, то библиотека создает невидимое окно в основном потоке приложения с обработчиком сообщения WM_DEVICE, в результате действия которого вызываются callback функции DeviceProc и DeviceListProc. При выполнении функции список устройств обновляется автоматически, что эквивалентно вызову функции SlCan_Update.

Примечание

Сообщение WM_DEVICE генерируется системой при подключении и отключении устройств slcan.

SlCan_Free

Функция выгружает библиотеку slcan.

C
```
SLCANAPI BOOL STDCALL SlCan_Free(
    BOOL    bDoCallBack
);
```

Pascal
```
function    SlCan_Free(
   bDoCallBack:longbool
   ):longbool;stdcall;
```

C#
```
public static extern bool Free(bool bDoCallBack);
```

VB
```
Public Function Shared Free(ByVal bDoCallBack As Boolean) As Boolean
```

Параметры

bDoCallBack
Если TRUE, то callback функции, установленные функцией SlCan_Load, вызываются, если FALSE, то нет.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Функция выгружает библиотеку, последовательно выполняя следующие действия:

- закрывает открытые устройства;

- удаляет объекты устройств из списка и уничтожает их;

- удаляет список устройств;

SlCan_Update

Функция обновляет список устройств slcan.

C
```
SLCANAPI BOOL STDCALL SlCan_Update();
```

Pascal
```
function    SlCan_Update:longbool;stdcall;
```

C#
```
public static extern bool Update();
```

VB
```
Public Shared Function Update() As Boolean
```

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Функция обновляет список устройств slcan, последовательно перебирая подключенные устройства, поддерживающие интерфейс slcan, и выполняет следующие действия:

- если оно не находится в списке, то;

- создает объект устройства и вставляет его в список устройств;

- если устройство уже находится в списке, то не делает ничего;

SlCan_GetDeviceCount

Функция возвращает количество устройств slcan.

C
```
SLCANAPI DWORD STDCALL SlCan_GetDeviceCount();
```

Pascal
```
function    SlCan_GetDeviceCount:longword;stdcall;
```

C#
```
public static extern uint GetDeviceCount();
```

VB
```
Public Shared Function GetDeviceCount() As UInt32
```

Результат

Функция возвращает количество устройств slcan, подключенных к системе..

SlCan_GetDevice

Функция возвращает ссылку на устройство slcan.

C
```
SLCANAPI HSLCAN STDCALL SlCan_GetDevice(
    DWORD       dwIndex
);
```

Pascal
```
function    SlCan_GetDevice(
   dwIndex:longword
   ):TSlCanDevice;stdcall;
```

C#
```
public static extern IntPtr GetDevice(uint dwIndex);
```

VB
```
Public Shared Function GetDevice(ByVal dwIndex As Integer) As IntPtr
```

Параметры

dwIndex

Номер устройства в списке устройств, может принимать значения от 0 до SlCan_GetDeviceCount -1.

Результат

Если результат равен 0xFFFFFFFF, то произошла ошибка, если нет, то полученное число однозначно идентифицирует объект устройства slcan и его можно использовать в качестве ссылки на устройство. Ссылка на устройство используется в качестве первого параметра функций SlCan_DeviceXXX. Ссылка становится неопределенной после вызова функции SlCan_Free или callback функции SlCan_Device_CallBack с параметром dwOperation равным SLCAN_DEVOP_DESTROY.

SlCan_DeviceGetProperty

Функция получает информацию об устройстве slcan.

C

```
SLCANAPI DWORD STDCALL SlCan_DeviceGetProperty(
   HSLCAN       hDevice,
   DWORD        dwIndex,
   PCHAR        pBuf,
   DWORD        dwSize
);
```

Pascal

```
function    SlCan_DeviceGetProperty(
   hDevice:TSlCanDevice;
   dwIndex:longword;
   pBuf:PChar;
   dwSize:longword
   ):longword;stdcall;
```

C#

```
public static extern uint DeviceGetProperty(IntPtr hDevice, uint dwIndex,
StringBuilder pBuf, int dwSize);
```

VB

```
Public Shared Function DeviceGetProperty(ByVal hDevice As IntPtr, ByVal dwIndex As
UInteger, ByVal pInfo As StringBuilder, ByVal dwSize As Integer) As UInteger
```

Параметры

hDevice

[in] Ссылка на устройство.

dwIndex

[in] Числовой индекс строки с информацией об устройстве. Может принимать значения констант SLCAN_INFO_INDEX_XXX.

Константа	Значение
SLCAN_INFO_INDEX_LINKNAME	0x00000000
SLCAN_INFO_INDEX_INSTANCEID	0x00000001
SLCAN_INFO_INDEX_DEVICEDESC	0x00000002
SLCAN_INFO_INDEX_FRIENDLYNAME	0x00000003
SLCAN_INFO_INDEX_PHOBJECTNAME	0x00000004
SLCAN_INFO_INDEX_MFG	0x00000005
SLCAN_INFO_INDEX_LOCATIONINFO	0x00000006
SLCAN_INFO_INDEX_ENUMERATOR	0x00000007
SLCAN_INFO_INDEX_CLASS	0x00000008
SLCAN_INFO_INDEX_CLASSGUID	0x00000009
SLCAN_INFO_INDEX_SERVICE	0x0000000A
SLCAN_INFO_INDEX_DRIVER	0x0000000B
SLCAN_INFO_INDEX_PORTNAME	0x0000000C
SLCAN_INFO_INDEX_PRODUCT	0x0000000D
SLCAN_INFO_INDEX_MANUFACTURER	0x0000000E
SLCAN_INFO_INDEX_CONFIGURATION	0x0000000F
SLCAN_INFO_INDEX_INTERFACE	0x00000010
SLCAN_INFO_INDEX_SERIAL	0x00000011
SLCAN_INFO_INDEX_ALIAS	0x00000012
SLCAN_INFO_INDEX_CHANNELLINK	0x00000013
SLCAN_INFO_INDEX_SERIALID	0x00000014

pBuf

[out] Указатель на буфер для приема строковой информации. Может быть NULL.

dwSize

[in]Размер буфера в байтах для приема строковой информации. Может быть 0.

Результат

Функция возвращает размер строки с информацией об устройстве.

Замечания

Информация об устройстве хранится в виде списка строк. Для доступа к строкам используется номер строки в списке – dwIndex.

Если указатель на буфер не равен NULL и его размер больше или равен длине строки с информацией, то функция копирует в буфер эту строку и возвращает ее размер в байтах. Если же это условие не выполняется, то функция просто возвращает размер строки.

SlCan_DeviceSetContext

Функция создает контекст устройства slcan.

C

```
SLCANAPI PVOID STDCALL SlCan_DeviceSetContext(
   HSLCAN       hDevice,
   PVOID        pBuf,
   DWORD        dwBufSize
);
```

Pascal

```
function    SlCan_DeviceSetContext(
   hDevice:TSlCanDevice;
   pBuf:pointer;
   dwBufSize:longword
   ):pointer;stdcall;
```

C#

```
public static extern IntPtr DeviceSetContext(IntPtr hDevice,
IntPtr pBuf, uint dwBufSize);
```

VB

```
Public Shared Function DeviceSetContext(ByVal hDevice As IntPtr, ByVal pBuf As IntPtr,
ByVal dwBufSize As Integer) As IntPtr
```

Параметры

hDevice

[in] Ссылка на устройство.

pBuf

[in] Указатель на буфер, содержание которого копируется в контекст устройства.

dwBufSize

[in] Размер буфера в байтах.

Результат

Если функция успешна, то результат равен указателю на вновь выделенную область памяти, в которой содержится контекст устройства, если параметр pBuf не равен NULL и параметр dwBufSize не равен 0. Если параметр pBuf равен NULL и/или параметр dwBufSize равен 0, то результат равен NULL.

Если произошла ошибка, то результат равен NULL.

Замечания

Контекст устройства – это область памяти, выделяемая библиотекой устройству для хранения пользовательских данных и переменных, связанных с устройством.

Порядок действия функции следующий:

1. Если память для контекста уже была выделена, то она освобождается, т.е. предыдущий контекст уничтожается.
2. Если pBuf не равен NULL и dwBufSize >0, то выделяется блок памяти размером dwBufSize байт. Если нет, то функция возвращается с результатом NULL.
3. Если блок памяти выделен, то в него копируются dwBufSize байт данных из буфера pBuf.
4. Функция возвращается с результатом равным указателю на выделенный блок памяти.

SlCan_DeviceGetContext

Функция возвращает ссылку на контекст устройства slcan.

```
SLCANAPI PVOID STDCALL SlCan_DeviceGetContext(
   HSLCAN      hDevice
);
```

Pascal

```
function    SlCan_DeviceGetContext(
   hDevice:TSlCanDevice
   ):pointer;stdcall;
```

C#

```
public static extern IntPtr DeviceGetContext(IntPtr hDevice);
```

VB

```
Public Shared Function DeviceGetContext(ByVal hDevice As IntPtr) As IntPtr
```

Параметры

hDevice

[in] Ссылка на устройство.

Результат

Если результат не равен NULL, то он является указателем на область памяти, в которой содержится контекст устройства.

SlCan_DeviceGetContextSize

Функция возвращает размер контекста устройства slcan.

C

```
SLCANAPI DWORD STDCALL SlCan_DeviceGetContextSize(
   HSLCAN      hDevice
);
```

Pascal

```
function    SlCan_DeviceGetContextSize(
   hDevice:TSlCanDevice;
   ):longword;stdcall;
```

C#

```
public static extern uint DeviceGetContextSize(IntPtr hDevice);
```

VB

```
Public Function DeviceGetContextSize(ByVal hDevice As IntPtr) As Integer
```

Параметры

hDevice

[in] Ссылка на устройство.

Результат

Результатом функции является размер контекста устройства в байтах.

SlCan_DeviceSetAlias

Функция устанавливает псевдоним устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetAlias(
   HSLCAN      hDevice,
   PCHAR       pBuf
);
```

Pascal

```
function    SlCan_DeviceSetAlias (
   hDevice:TSlCanDevice;
   pBuf:PChar
   ):longbool;stdcall;
```

C#

Параметры

hDevice

[in] Ссылка на устройство.

pBuf

[in] Указатель на буфер, в котором содержится строка с псевдонимом устройства. Строка должна заканчиваться 0.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Псевдоним устройства – строка, которая связана с конкретным экземпляром устройства с данным серийным номером и сохраняется в реестре. Псевдоним может быть использован для идентификации устройства в системе.

SlCan_DeviceGetAlias

Функция получает строковый псевдоним устройства slcan.

C

```
SLCANAPI DWORD STDCALL SlCan_DeviceGetAlias(
   HSLCAN       hDevice,
   PCHAR        pBuf,
   DWORD        dwSize
);
```

Pascal

```
function    SlCan_DeviceGetAlias(
   hDevice:TSlCanDevice;
   pBuf:PChar;
   dwSize:longword
   ):longword;stdcall;
```

C#

Параметры

hDevice

[in] Ссылка на устройство.

pBuf

[out] Указатель на буфер, который функция будет заполнять символами псевдонима устройства. Может быть NULL.

dwSize

[in] Размер буфера в байтах. Может быть 0.

Результат

Функция возвращает размер строки псевдонима в байтах.

Замечания

Псевдоним устройства – строка, которая связана с конкретным экземпляром устройства, т.е. устройством с определенным серийным номером, и сохраняется в реестре. Псевдоним может быть использован для идентификации устройства в системе.

Если pBuf не NULL и dwSize больше или равна размеру строки псевдонима, то функция копирует символы псевдонима в буфер.

SlCan_DeviceGetCapabilities

Функция получает возможности устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetCapabilities(
  HSLCAN       hDevice,
  PSLCAN_CAPABILITIES   pCap
);
```

Pascal

```
function    SlCan_DeviceGetCapabilities(
   hDevice: TSlCanDevice;
   pCap:    pSlCanCapabilities
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetCapabilities(IntPtr hDevice,
out Capability Cap);
```

VB

```
Public Shared Function DeviceGetCapabilities(ByVal hDevice As IntPtr, ByRef Cap As
Capability) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pCap

[out] Указатель на структуру типа SlCan_DeviceCapabilities.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

SlCan_DeviceOpen

Функция открывает устройство slcan для операций ввода вывода.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceOpen(
  HSLCAN        hDevice
);
```

Pascal

```
function    SlCan_DeviceOpen(
   hDevice:TSlCanDevice
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceOpen(IntPtr hDevice);
```

VB

```
Public Function DeviceOpen(ByVal hDevice As IntPtr) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Если устройство уже открыто, то функция не делает ничего и возвращает TRUE. Если устройство не было открыто, то функция выполняет следующее:

1. Вызывает функцию WinAPI CreateFile. В случае успеха вызывает callback функцию DeviceProc с параметром dwOperation = SLCAN_DEVOP_CREATEHANDLE. Если вызов CreateFile неудачен, то функция возвращается с результатом FALSE.
2. Производит инициализацию устройства, устанавливая параметры работы в начальное состояние. Инициализирует буферы для ввода-вывода. Устанавливает режим SLCAN_MODE_CONFIG. Устанавливает уровень генерации событий SLCAN_EVT_LEVEL_RX_MSG и максимально возможное для данного устройства значение скорости передачи. Если устройство поддерживает генерацию временных меток, то устанавливает период генератора в 1мкс и начальное значение счетчика в 0. В случае успеха вызывает callback функцию DeviceProc с параметром dwOperation = SLCAN_DEVOP_OPEN. Если произошла ошибка, то вызывает функцию WinAPI CloseHandle, затем вызывает callback функцию DeviceProc с параметром dwOperation = SLCAN_DEVOP_DESTROYHANDLE и возвращается с результатом FALSE.
3. Возвращается с результатом TRUE.

SlCan_DeviceClose

Функция закрывает устройство slcan для операций ввода вывода.

C
```
SLCANAPI BOOL STDCALL SlCan_DeviceClose(
  HSLCAN   hDevice
);
```

Pascal
```
function    SlCan_DeviceClose(
   hDevice:TSlCanDevice
   ):longbool;stdcall;
```

C#
```
public static extern bool DeviceClose(IntPtr hDevice);
```

VB
```
Public Function DeviceClose(ByVal hDevice As IntPtr) As Boolean
```

Параметры

hDevice
[in] Ссылка на устройство.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Если устройство не открыто, то функция не делает ничего и возвращает TRUE. Если устройство было открыто, то функция выполняет следующее:

1. Производит деинициализацию устройства. Останавливает все операции ввода – вывода. Переводит устройство в режим SLCAN_MODE_CONFIG. Если установлена callback функция DeviceProc, то вызывает её с параметром dwOperation = SLCAN_DEVOP_CLOSE.
2. Вызывает функцию WinAPI CloseHandle, затем вызывает callback функцию DeviceProc с параметром dwOperation = SLCAN_DEVOP_ DESTROYHANDLE.
3. Возвращается с результатом TRUE.

SlCan_DeviceSetMode

Функция устанавливает режим работы устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetMode(
  HSLCAN       hDevice,
  DWORD        dwMode
);
```

Pascal

```
function    SlCan_DeviceSetMode(
   hDevice:TSlCanDevice;
   dwMode:longword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetMode(IntPtr hDevice,uint dwMode);
```

VB

```
Public Shared Function DeviceSetMode(ByVal hDevice As IntPtr, ByVal dwMode As Integer)
As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

dwMode

[in] Режим работы устройства. Может принимать следующие значения.

Константа	Значение	Режим
SLCAN_MODE_CONFIG	0x00	Режим конфигурации устройства.
SLCAN_MODE_NORMAL	0x01	Нормальный режим работы. Устройство может посылать и принимать CAN фреймы.
SLCAN_MODE_LISTEN_ONLY	0x02	Режим прослушивания. Устройство может только принимать фреймы и ничем не обнаруживает себя в сети.
SLCAN_MODE_LOOP_BACK	0x03	Режим самопрослушивания. Устройство принимает посланные самим собой фреймы.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Режимы работы, отличные от SLCAN_MODE_CONFIG, могут быть установлены только, если устройство находится в режиме SLCAN_MODE_CONFIG.

Если устройство поддерживает генерацию временных меток, то при переходе с режима SLCAN_MODE_CONFIG устанавливает период генератора и начальное значение счетчика, ранее установленных с помощью функций SlCan_DeviceSetStartTimeStamp и SlCan_SetTimeStampPeriod.

Режимы работы, поддерживаемые устройством, определяются полем bModes структуры SlCan_DeviceCapabilities.

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceGetMode

Функция получает текущий режим работы устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetMode(
  HSLCAN       hDevice,
  PDWORD       pdwMode
);
```

Pascal

```
function    SlCan_DeviceGetMode(
   hDevice:TSlCanDevice;
   pdwMode:Plongword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetMode(IntPtr hDevice, out uint dwMode);
```

VB

```
Public Shared Function DeviceGetMode(ByVal hDevice As IntPtr, ByRef dwMode As Integer)
As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pdwMode

[out] Указатель на переменную, в которую функция копирует текущий режим работы устройства. Может принимать следующие значения:

Константа	Значение	Режим
SLCAN_MODE_CONFIG	0x00	Режим конфигурации устройства.
SLCAN_MODE_NORMAL	0x01	Нормальный режим работы. Устройство может посылать и принимать CAN фреймы.
SLCAN_MODE_LISTEN_ONLY	0x02	Режим прослушивания. Устройство может только принимать фреймы и ничем не обнаруживает себя в сети.
SLCAN_MODE_LOOP_BACK	0x03	Режим самопрослушивания. Устройство принимает посланные самим собой фреймы.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечания

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceGetState

Функция получает текущее состояние устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetState(
  HSLCAN       hDevice,
  PSLCAN_STATE pState
);
```

Pascal

```
function    SlCan_DeviceGetState(
   hDevice:TSlCanDevice;
   pState:PSlCanState
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetState(IntPtr hDevice, ref SlCan.State State);
```

VB

```
Public Shared Function DeviceGetState(ByVal hDevice As IntPtr, ByRef dwState As
UInteger) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pState

[out] Указатель на переменную, в которую функция копирует текущее состояние устройства.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Для более подробного пояснения см. SlCan_State

SlCan_DeviceGetBitRate

Функция получает текущие настройки скорости приема передачи CAN устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetBitRate(
  HSLCAN            hDevice,
  PSLCAN_BITRATE   pBitRate
);
```

Pascal

```
function    SlCan_DeviceGetBitRate(
   hDevice:TSlCanDevice;
   pBitRate:PSlCanBitRate
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetBitRate(IntPtr hDevice, out BitRate bitrate);
```

VB

```
Public Shared Function DeviceGetBitRate(ByVal hDevice As IntPtr, ByRef bitrate As
BitRate) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pBitRate

[out] Указатель на переменную, в которую функция копирует текущее состояние скорости передачи.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Для более подробного пояснения см. SlCan_BitRate.

SlCan_DeviceSetBitRate

Функция устанавливает настройки скорости приема передачи CAN устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetBitRate(
  HSLCAN           hDevice,
  PSLCAN_BITRATE   pBitRate
);
```

Pascal

```
function    SlCan_DeviceSetBitRate(
   hDevice:TSlCanDevice;
   pBitRate:PSlCanBitRate
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetBitRate(IntPtr hDevice, ref BitRate bitrate);
```

VB

```
Public Shared Function DeviceSetBitRate(ByVal hDevice As IntPtr, ByRef bitrate As
BitRate) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pBitRate

[in] Указатель на переменную, в которой находится устанавливаемые параметры состояние скорости передачи.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто и находится в режиме SLCAN_MODE_CONFIG.

Для более подробного пояснения см. SlCan_BitRate.

SlCan_DeviceSetTXTimeOut

Функция устанавливает значение таймаута передачи CAN фреймов устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetTXTimeOut(
  HSLCAN      hDevice,
  DWORD       dwMilliseconds
);
```

Pascal

```
function    SlCan_DeviceSetTXTimeOut(
   hDevice:TSlCanDevice;
   dwMilliseconds:longword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetTXTimeOut(IntPtr hDevice, uint dwMilliseconds);
```

VB

```
Public Shared Function DeviceSetTXTimeOut(ByVal hDevice As IntPtr, ByVal
dwMilliseconds As UInteger) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

dwMilliseconds

[in] Значения таймаута передачи в миллисекундах.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Для более подробного пояснения см. SlCan_DeviceWriteMessages.

SlCan_DeviceGetTXTimeOut

Функция получает значение таймаута передачи CAN фреймов устройства slcan.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetTXTimeOut(
  HSLCAN           hDevice,
  PDWORD           pdwMillisecond
);
```

Pascal

```
function    SlCan_DeviceGetTXTimeOut(
   hDevice:TSlCanDevice;
   pdwMillisecond:Plongword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetTXTimeOut(IntPtr hDevice, out uint dwMilliseconds);
```

VB

```
Public Shared Function DeviceGetTXTimeOut(ByVal hDevice As IntPtr, ByRef
dwMilliseconds As UInteger) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pdwMilliseconds

[out] Указатель на переменную, в которой функция передает значение таймаута передачи в миллисекундах.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceEnaRec

Функция разрешения и запрета передачи устройством в PC принятых CAN фреймов.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceEnaRec(
  HSLCAN           hDevice,
  BYTE             bValue
);
```

Pascal

```
function    SlCan_DeviceEnaRec(
   hDevice:TSlCanDevice;
   bValue:byte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceEnaRec(IntPtr hDevice, byte bValue);
```

VB

```
Public Shared Function DeviceEnaRec(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

bValue

[in] Если 0 то передача принятых фреймов запрещена, если <> 0, то передача разрешена.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Основным назначением функции является ограничение трафика данных без закрытия устройства.

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceWriteMessages

Функция записи CAN фреймов в устройство для передачи.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceWriteMessages(
  HSLCAN           hDevice,
  PSLCAN_MESSAGE   pMsg,
  DWORD            dwCount,
  PBYTE            pbStatus
);
```

Pascal

```
function    SlCan_DeviceWriteMessages(
   hDevice:TSlCanDevice;
   pMsg:pSlCanMessage;
   Count:longword;
   pStatus:pByte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceWriteMessages(IntPtr hDevice,[In,Out] SlCan.Message[]
pCanMessage,uint Count,out byte Status);
```

VB

```
Public Shared Function DeviceWriteMessages(ByVal hDevice As IntPtr, <[In](), Out()>
ByVal pMsg() As Message, ByVal dwCount As UInteger, ByRef pbStatus As Byte) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pMsg

[in] Указатель на буфер, содержащий CAN сообщения.

dwCount

[in] Количество сообщений в буфере.

pStatus

[out] Указатель на переменную, в которой функция устанавливает значение статуса передачи. Если 0, то передача осуществлена успешно, если нет, то произошла ошибка.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Своим действием функция направляет запрос на передачу в устройство. Если запрос не передан в устройство из-за ошибки, то функция возвращает FALSE.

Если запрос передан в устройство, то оно обрабатывает запрос на передачу, после обработки передает статус передачи и функция возвращает TRUE. Возможны следующие значения статуса передачи.

Константа	Значение	Значение
SLCAN_TX_STATUS_OK	0x00	Все сообщения переданы.
SLCAN_TX_STATUS_TIMEOUT	0x01	Сообщения не переданы. Передача прервана из-за таймаута.
SLCAN_TX_STATUS_BUSOFF	0x02	Сообщения не переданы. Передача прервана из-за ошибок на шине CAN и перехода CAN контроллера устройства в режим BUS OFF..
SLCAN_TX_STATUS_ABORT	0x03	Сообщения не переданы. Передача прервана внешней командой.
SLCAN_TX_STATUS_NOT_ENA	0x04	Сообщения не переданы. Передача запрещена.
SLCAN_TX_STATUS_ERROR_ONE_SHOT	0x05	Сообщение передавалось в режиме ONE

		SHOT и произошла ошибка.
SLCAN_TX_STATUS_INVALID_MODE	0x06	Сообщения не переданы. Режим устройства не поддерживает передачу.
SLCAN_TX_STATUS_UNKNOWN	0x0F	Сообщения не переданы. Ошибка неизвестна.

Для успешного выполнения функции устройство должно быть открыто и находится в режиме, допускающем передачу фреймов.

SlCan_DeviceReadMessages

Функция чтения принятых CAN фреймов из устройства.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceReadMessages(
  HSLCAN            hDevice,
  DWORD             dwTimeOut,
  PSLCAN_MESSAGE    pMsg,
  DWORD             dwCount,
  PDWORD            pdwCount
);
```

Pascal

```
function    SlCan_DeviceReadMessages(
   hDevice:TSlCanDevice;
   dwTimeOut:longword;
   pMsg:pSlCanMessage;
   dwCount:longword;
   pdwCount:pLongword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceReadMessages(IntPtr hDevice, uint dwTimeOut, [In, Out]
CanMessage[] pMsg, uint dwCount, out uint pdwCount);
```

VB

```
Public Shared Function DeviceReadMessages(ByVal hDevice As IntPtr, ByVal dwTimeOut As
Integer, <[In](), Out()> ByVal pMsg() As Message, ByVal dwCount As UInteger, ByRef
pdwCount As UInteger) As Boolean
```

Параметры

hDevice
[in] Ссылка на устройство.

dwTimeOut
[in] Время таймаута в миллисекундах.

pMsg
[in] Указатель на буфер для приема сообщений. Должен существовать до вызова функции.

dwCount
[in] Размер буфера в сообщениях.

pdwCount
[out] Указатель на переменную, в которой функция устанавливает количество принятых сообщений.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Действие функции зависит от значения параметра dwTimeOut

dwTimeOut = 0. В этом случае функция считывает из внутреннего буфера драйвера максимум dwCount уже принятых сообщений, и немедленно возвращается с результатом TRUE,установив в pdwCount количество прочитанных сообщений.

dwTimeOut = 0xFFFFFFFF. В этом случае функция ожидает бесконечно долго появления в буфере драйвера dwCount сообщений, считывает из него сообщения и немедленно возвращается с результатом TRUE, установив в pdwCount значение dwCount. Если в буфере драйвера уже есть dwCount сообщений, то функция не ожидает, просто считывает эти сообщения. В случае появления во время ожидания ошибки функция немедленно возвращается с результатом FALSE.

0 < dwTimeOut < 0xFFFFFFFF. В этом случае функция ожидает появления в буфере драйвера dwCount сообщений или прохождения времени dwTimeOut миллисекунд, считывает из него, если есть, сообщения и немедленно возвращается с результатом TRUE, установив в pdwCount значение dwCount. Если в буфере

драйвера уже есть dwCount сообщений, то функция не ожидает, а просто считывает эти сообщения. В случае появления во время ожидания ошибки функция немедленно возвращается с результатом FALSE.

Для успешного выполнения функции устройство должно быть открыто и находится в режиме, допускающем приём фреймов.

Примечание

Следует с осторожностью использовать значения dwCount, равные 0xFFFFFFFF, а также большим значениям времени, в однопоточных приложениях.

Следует также избегать использования данной функции, если установлен уровень генерации событий больший, чем SLCAN_EVT_LEVEL_RX_MSG, в этом случае лучше использовать функцию SlCan_DeviceReadEvents и выделять самостоятельно из потока событий принятые CAN сообщения.

SlCan_DeviceSetLatency

Устанавливает значение задержки передачи неполного USB пакета

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetLatency(
  HSLCAN            hDevice,
  BYTE              bValue
);
```

Pascal

```
function SlCan_DeviceSetLatency(
   hDevice: TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetLatency(IntPtr hDevice, byte bValue);
```

VB

```
Public Shared Function DeviceSetLatency(ByVal hDevice As IntPtr, ByVal bValue As Byte)
As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

bValue

[in] Значение задержки в миллисекундах.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Задержка передачи неполного USB пакета определяет время ожидания приема очередного фрейма или генерации события перед тем, как устройство отправит этот пакет в PC.

При открытии устройства данное значение устанавливается равным 10 миллисекундам.

Для успешного выполнения функции устройство должно быть открыто.

SICan_DeviceGetLatency

Поучает значение задержки передачи неполного USB пакета

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetLatency(
  HSLCAN            hDevice,
  PBYTE             pbValue
);
```

Pascal

```
function SlCan_DeviceGetLatency(
   hDevice: TSlCanDevice;
   pbValue: pbyte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetLatency(IntPtr hDevice, out byte pbValue);
```

VB

```
Public Shared Function DeviceGetLatency(ByVal hDevice As IntPtr, ByRef pbValue As
Byte) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pbValue

[out] Буфер, в который функция помещает значение задержки в миллисекундах.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DevicePurge

Очищает входной и выходной буферы, а так же может прервать текущую передачу и прием данных.

C

```
SLCANAPI BOOL STDCALL SlCan_DevicePurge(
  HSLCAN  hDevice,
  BYTE    bValue
);
```

Pascal

```
function SlCan_DevicePurge(
   hDevice: TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;
```

C#

```
public static extern bool DevicePurge(IntPtr hDevice, byte bValue);
```

VB

```
Public Shared Function DevicePurge(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

bValue

[in] Параметр из комбинации следующих значений.

Константа	Значение	Назначение
SLCAN_PURGE_TX_ABORT	0x01	Прерывает текущий запрос на передачу.
SLCAN_PURGE_RX_ABORT	0x02	Прерывает текущую операцию чтения.
SLCAN_PURGE_TX_CLEAR	0x04	Очищает буфер передачи.
SLCAN_PURGE_RX_CLEAR	0x08	Очищает буфер приема.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceSetEventLevel

Устанавливает уровень генерации событий устройством.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetEventLevel(
  HSLCAN   hDevice,
  BYTE     bValue
);
```

Pascal

```
function SlCan_DeviceSetEventLevel(
   hDevice: TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetEventLevel(IntPtr hDevice, byte bValue);
```

VB

```
Public Shared Function DeviceSetEventLevel(ByVal hDevice As IntPtr, ByVal bValue As
Byte) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

bValue

[in] Уровень генерации событий, может принимать следующие значения.

Константа	Значение	Генерируемые события
SLCAN_EVT_LEVEL_RX_MSG	0x00	SLCAN_EVT_TYPE_RX
SLCAN_EVT_LEVEL_TIME_STAMP	0x01	SLCAN_EVT_TYPE_RX SLCAN_EVT_STAMP_INC
SLCAN_EVT_LEVEL_TX_MSG	0x02	SLCAN_EVT_TYPE_RX SLCAN_EVT_STAMP_INC SLCAN_EVT_TYPE_START_TX SLCAN_EVT_TYPE_END_TX SLCAN_EVT_TYPE_ABORT_TX
SLCAN_EVT_LEVEL_BUS_STATE	0x03	SLCAN_EVT_TYPE_RX. SLCAN_EVT_STAMP_INC SLCAN_EVT_TYPE_START_TX SLCAN_EVT_TYPE_END_TX SLCAN_EVT_TYPE_ABORT_TX SLCAN_EVT_TYPE_BUS_STATE
SLCAN_EVT_LEVEL_COUNTS	0x04	SLCAN_EVT_TYPE_RX SLCAN_EVT_STAMP_INC SLCAN_EVT_TYPE_START_TX SLCAN_EVT_TYPE_END_TX SLCAN_EVT_TYPE_ABORT_TX SLCAN_EVT_TYPE_BUS_STATE SLCAN_EVT_TYPE_ERROR_COUNTS
SLCAN_EVT_LEVEL_ERRORS	0x05	SLCAN_EVT_TYPE_RX SLCAN_EVT_STAMP_INC SLCAN_EVT_TYPE_START_TX SLCAN_EVT_TYPE_END_TX SLCAN_EVT_TYPE_ABORT_TX SLCAN_EVT_TYPE_BUS_STATE SLCAN_EVT_TYPE_ERROR_COUNTS SLCAN_EVT_TYPE_BUS_ERROR SLCAN_EVT_TYPE_ARBITRATION

Устанавливаемый уровень не должен превышать максимального значения, поддерживаемого устройством, иначе функция вернёт FALSE.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Максимальный уровень генерации событий, поддерживаемый данным устройством, определяется полем bMaxEventLevel структуры SICan_DeviceCapabilities.

SlCan_DeviceGetEventLevel

Получает уровень генерации событий устройством.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetEventLevel(
    HSLCAN  hDevice,
    PBYTE   pbValue
);
```

Pascal

```
function SlCan_DeviceGetEventLevel(
   hDevice: TSlCanDevice;
   pbValue: pbyte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetEventLevel(IntPtr hDevice, out byte pbValue);
```

VB

```
Public Shared Function DeviceGetEventLevel(ByVal hDevice As IntPtr, ByRef pbValue As
Byte) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pbValue

[out] Буфер, в который функция помещает текущий уровень.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceSetStartTimeStamp

Устанавливает начальное значение счетчика временных меток.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetStartTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);
```

Pascal

```
function SlCan_DeviceSetStartTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetStartTimeStamp(IntPtr hDevice, ref UInt64 Value);
```

VB

```
Public Shared Function DeviceSetStartTimeStamp(ByVal hDevice As IntPtr, ByRef Value As
UInt64) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pValue

[in] Указатель на буфер, содержащий начальное значение счетчика временных меток.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Примечание

Счетчик временных меток работает только во время режимов работы устройства, отличных от SLCAN_MODE_CONFIG. Во время работы значение счетчика инкрементируется на единицу с определенным периодом. Счетчик загружается начальным значением во время смены режима работы функцией SlCan_DeviceSetMode.

SICan_DeviceGetStartTimeStamp

Получает начальное значение счетчика временных меток.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetStartTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);
```

Pascal

```
function SlCan_DeviceGetStartTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetStartTimeStamp(IntPtr hDevice, out UInt64 pValue);
```

VB

```
Public Shared Function DeviceGetStartTimeStamp(ByVal hDevice As IntPtr, ByRef pValue
As UInt64) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pValue

[out] Указатель на буфер, в который функция помещает начальное значение счетчика временных меток.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceGetTimeStamp

Получает текущее значение счетчика временных меток.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);
```

Pascal

```
function SlCan_DeviceGetTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetTimeStamp(IntPtr hDevice, out UInt64 pValue);
```

VB

```
Public Shared Function DeviceGetTimeStamp(ByVal hDevice As IntPtr, ByRef pValue As
UInt64) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pValue

[out] Указатель на буфер, в который функция помещает текущее значение счетчика временных меток.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Примечание

Счетчик временных меток работает только во время режимов работы устройства, отличных от SLCAN_MODE_CONFIG. Во время работы значение счетчика инкрементируется на единицу с определенным периодом. Во время режима SLCAN_MODE_CONFIG счетчик не инкрементируется Счетчик загружается начальным значением установленным функцией SlCan_DeviceSetStartTimeStamp во время смены режима работы функцией SlCan_DeviceSetMode.

SlCan_DeviceSetTimeStampPeriod

Устанавливает период счетчика временных меток.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetTimeStampPeriod(
  HSLCAN  hDevice,
  LONG    lValue
);
```

Pascal

```
function SlCan_DeviceSetTimeStampPeriod(
   hDevice: TSlCanDevice;
   lValue: longint
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetTimeStampPeriod(IntPtr hDevice, int lValue);
```

VB

```
Public Shared Function DeviceSetTimeStampPeriod(ByVal hDevice As IntPtr, ByVal lValue
As Integer) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

lValue

[in] Период счетчика временных меток. Если значение положительно, то период измеряется в микросекундах, если отрицательно, то период измеряется в длительности бита текущей скорости передачи.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Функция записывает значение периода во внутреннюю переменную устройства. Период счетчика устанавливается во время перехода с режима SLCAN_MODE_CONFIG на любой другой режим работы в результате действия функции SlCan_DeviceSetMode.

SlCan_DeviceGetTimeStampPeriod

Получает период счетчика временных меток.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetTimeStampPeriod(
  HSLCAN  hDevice,
  PLONG   plValue
);
```

Pascal

```
function SlCan_DeviceGetTimeStampPeriod(
   hDevice: TSlCanDevice;
   plValue: Plongint
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetTimeStampPeriod(IntPtr hDevice, out int lValue);
```

VB

```
Public Shared Function DeviceGetTimeStampPeriod(ByVal hDevice As IntPtr, ByRef lValue
As Integer) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

plValue

[out] Буфер, в который функция помещает значение периода счетчика временных меток. Если значение положительно, то период измеряется в микросекундах, если значением отрицательно, то период измеряется в длительности бита текущей скорости передачи.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

SlCan_DeviceSetExMode

Устанавливает расширенный режим работы устройства.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceSetExMode(
  HSLCAN      hDevice,
  BYTE        bValue
);
```

Pascal

```
function SlCan_DeviceSetExMode(
   hDevice: TSlCanDevice;
   bMode:   byte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceSetEXMode(IntPtr hDevice, byte bValue);
```

VB

```
Public Shared Function DeviceSetEXMode(ByVal hDevice As IntPtr, ByVal bValue As Byte)
As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

bValue

[in] Значение расширенного режима.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Примечание

Функция пока не определена.

SICan_DeviceGetExMode

Получает значение расширенного режима работы устройства.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceGetExMode(
  HSLCAN            hDevice,
  PBYTE             pbValue
);
```

Pascal

```
function SlCan_DeviceGetExMode(
   hDevice:TSlCanDevice;
   pbMode:  pbyte
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceGetExMode(IntPtr hDevice, out byte pbValue);
```

VB

```
Public Shared Function DeviceGetExMode(ByVal hDevice As IntPtr, ByRef pbValue As Byte)
As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pbValue

[out] Значение расширенного режима.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Для успешного выполнения функции устройство должно быть открыто.

Примечание

Функция пока не определена.

SlCan_DeviceWriteMessagesEx

Функция расширенной записи CAN фреймов в устройство для передачи.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceWriteMessagesEx(
  HSLCAN           hDevice,
  PSLCAN_TXMESSAGE pMsg,
  DWORD            dwCount,
  PBYTE            pbStatus
  PDWORD           pdwCount
);
```

Pascal

```
function    SlCan_DeviceWriteMessagesEx(
   hDevice:TSlCanDevice;
   pMsg:pSlCanTxMessage;
   Count:longword;
   pStatus:pByte;
   pdwCount:plongword;
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceWriteMessagesEx(IntPtr hDevice,[In,Out]
SlCan.TxMessage[] pMsg,uint Count,out byte Status, out uint pdwCount);
```

VB

```
Public Shared Function DeviceWriteMessagesEx(ByVal hDevice As IntPtr, <[In](), Out()>
ByVal pMsg() As TxMessage, ByVal dwCount As UInteger, ByRef pbStatus As Byte, ByRef
pdwCount As UInteger) As Boolean
```

Параметры

hDevice

[in] Ссылка на устройство.

pMsg

[in] Указатель на буфер, содержащий CAN сообщения для передачи.

dwCount

[in] Количество сообщений в буфере.

pStatus

[out] Указатель на переменную, в которой функция устанавливает значение статуса передачи. Если 0, то передача осуществлена успешно, если нет, то произошла ошибка.

pdwCount

[out] Указатель на переменную, в которой функция устанавливает количество переданных сообщений.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Своим действием функция направляет запрос на передачу в устройство. Если запрос не передан в устройство из-за ошибки, то функция возвращает FALSE.

Если запрос передан в устройство, то оно обрабатывает запрос на передачу, после обработки передает статус передачи, количество переданных сообщений и функция возвращает TRUE. Возможны следующие значения статуса передачи.

Константа	Значение	Значение
SLCAN_TX_STATUS_OK	0x00	Все сообщения переданы.
SLCAN_TX_STATUS_TIMEOUT	0x01	Сообщения не переданы. Передача прервана из-за таймаута.
SLCAN_TX_STATUS_BUSOFF	0x02	Сообщения не переданы. Передача прервана

		из-за ошибок на шине CAN и перехода CAN контроллера устройства в режим BUS OFF..
SLCAN_TX_STATUS_ABORT	0x03	Сообщения не переданы. Передача прервана внешней командой.
SLCAN_TX_STATUS_NOT_ENA	0x04	Сообщения не переданы. Передача запрещена.
SLCAN_TX_STATUS_ERROR_ONE_SHOT	0x05	Сообщение передавалось в режиме ONE SHOT и произошла ошибка.
SLCAN_TX_STATUS_INVALID_MODE	0x06	Сообщения не переданы. Режим устройства не поддерживает передачу.
SLCAN_TX_STATUS_UNKNOWN	0x0F	Сообщения не переданы. Ошибка неизвестна.

Для успешного выполнения функции устройство должно быть открыто и находится в режиме, допускающем передачу фреймов.

SlCan_DeviceReadEvents

Функция чтения событий, генерируемых устройством.

C

```
SLCANAPI BOOL STDCALL SlCan_DeviceReadEvents(
  HSLCAN            hDevice,
  DWORD             dwTimeOut,
  PSLCAN_EVENT      pEvent,
  DWORD             dwCount,
  PDWORD            pdwCount
);
```

Pascal

```
function SlCan_DeviceReadEvents(
   hDevice:  TSlCanDevice;
   dwTimeOut:longword;
   pEvent:   PSlCanEvent;
   Count:    longword;
   pCount:   Plongword
   ):longbool;stdcall;
```

C#

```
public static extern bool DeviceReadEvents(IntPtr hDevice, uint dwTimeOut, [In, Out]
SlCan.Event[] pEvent, uint dwCount, out uint pdwCount);
```

VB

```
Public Shared Function DeviceReadEvents(ByVal hDevice As IntPtr, ByVal dwTimeOut As
UInteger, <[In](), Out()> ByVal pEvent() As CEvent, ByVal dwCount As UInteger, ByRef
pdwCount As UInteger) As Boolean
```

Параметры

hDevice
[in] Ссылка на устройство.

dwTimeOut
[in] Время таймаута в миллисекундах.

pEvent
[in] Указатель на буфер для приема событий. Должен существовать до вызова функции.

dwCount
[in] Размер буфера в событиях.

pdwCount
[out] Указатель на переменную, в которой функция устанавливает количество принятых событий.

Результат

В случае успешного выполнения функция возвращает TRUE, иначе функция возвращает FALSE.

Замечание

Действие функции зависит от значения параметра dwTimeOut

dwTimeOut = 0. В этом случае функция считывает из внутреннего буфера драйвера максимум dwCount уже принятых событий, и немедленно возвращается с результатом TRUE,установив в pdwCount количество прочитанных событий.

dwTimeOut = 0xFFFFFFFF. В этом случае функция ожидает бесконечно долго появления в буфере драйвера dwCount событий, считывает из него события и немедленно возвращается с результатом TRUE, установив в pdwCount значение dwCount. Если в буфере драйвера уже есть dwCount событий, то функция не ожидает, просто считывает эти события. В случае появления во время ожидания ошибки функция немедленно возвращается с результатом FALSE.

0 < dwTimeOut < 0xFFFFFFFF. В этом случае функция ожидает появления в буфере драйвера dwCount событий или прохождения времени dwTimeOut миллисекунд, считывает из него, если есть, события и немедленно возвращается с результатом TRUE, установив в pdwCount значение dwCount. Если в буфере

драйвера уже есть dwCount событий, то функция не ожидает, а просто считывает эти события. В случае появления во время ожидания ошибки функция немедленно возвращается с результатом FALSE.

Для успешного выполнения функции устройство должно быть открыто.

Примечание

Следует с осторожностью использовать значения dwCount, равные 0xFFFFFFFF, а также большим значениям времени, в однопоточных приложениях.

Если установлен уровень генерации событий SLCAN_EVT_LEVEL_RX_MSG, в этом случае лучше использовать функцию SlCan_DeviceReadMessages, так как не нужно выделять самостоятельно из потока событий принятые CAN сообщения.

Типы

HSLCAN, TSlCanDevice

Тип ссылки на устройство slcan.

C
```
typedef     PVOID   HSLCAN;
```

Pascal
```
type
{$IFDEF CompilerVersion}
{$IF CompilerVersion >= 16 }
TSlCanDevice = NativeUInt;
{$ELSE}
TSlCanDevice = longword;
{$ENDIF}
{$ELSE}
TSlCanDevice = longword;
{$ENDIF}
```

C#
```
IntPtr
```

VB
```
IntPtr
```

Замечание

Ссылка представляет собой целое число базового типа, т.е. для 32-разрядных приложений - это 32-битное целое, для 64-разрядных приложений – это 64-битное целое.

Ссылка является однозначным идентификатором устройства slcan в библиотеке.

Ссылка не является ни указателем, ни handle драйвера устройства.

SICan_Device_CallBack

Тип callback функции изменения состояния устройства slcan.

C

```
typedef    void (STDCALL* SLCAN_DEVICE_CALLBACK)(
    HSLCAN     hDevice,
    DWORD      dwIndex,
    DWORD      dwOperation,
    PVOID      pContext,
    DWORD      dwContextSize
);
```

Pascal

```
type
   TSlCan_DeviceCallBack = procedure(
      dwDevice:TSlCanDevice;
      dwIndex:longword;
      dwOperation:longword;
      pContext:pointer;
      dwContextSize:longword
      );stdcall;
```

C#

```
public delegate void DeviceCallBack(IntPtr hDevice, uint dwIndex, DeviceOp
dwOperation, IntPtr pContext, uint dwContextSize);
```

VB

```
Public Delegate Sub DeviceCallBack(ByVal hDevice As IntPtr, ByVal dwIndex As UInteger,
ByVal dwOperation As UInteger, ByVal pContext As IntPtr, ByVal dwContextSize As
UInteger)
```

Параметры

hDevice
Ссылка на устройство.

dwIndex
Номер устройства в списке устройств.

dwOperation
Тип изменения.

pContext
Указатель на контекст устройства.

dwContextSize
Размер контекста устройства в байтах.

Замечание

Для того чтобы использовать callback функцию устройства, необходимо при инициализации библиотеки указать адрес функции данного типа в качестве первого параметра при вызове функции SICan_Load. Тогда она будет вызываться ядром библиотеки при изменении состояния объекта устройства, указывая тип изменения в параметре dwOperation, который может принимать следующие значения:

SLCAN_DEVOP_CREATE
Значение параметра указывает на создание объекта устройства и включения его в список устройств. Это происходит в следующих случаях:

1. Библиотека загружена. Устройство подключается к компьютеру. Вызов функции осуществляется при обработке сообщения WM_DEVICE.
2. Устройство уже подключено к компьютеру. Библиотека загружается. Вызов функции осуществляется во время исполнения функции SICan_Load.

SLCAN_DEVOP_CREATEHANDLE
Значение параметра указывает на успешный вызов функции CreateFile во время работы функции SICan_DeviceOpen.

SLCAN_DEVOP_OPEN
Значение параметра указывает на успешную инициализацию устройства после вызова функции с параметром SLCAN_DEVOP_CREATEHANDLE . Функция вызывается во время работы функции SlCan_DeviceOpen.

SLCAN_DEVOP_CLOSE
Значение параметра указывает на успешную деинициализацию устройства во время вызова функции SlCan_DeviceClose или вызова SlCan_Free с параметром bDoCallBack=TRUE..

SLCAN_DEVOP_DESTROYHANDLE
Значение параметра указывает на успешный вызов системной функции CloseHandle в следующих случаях.

1. Во время закрытия устройства после вызова функции с параметром SLCAN_DEVOP_CLOSE.
2. После отключения открытого устройства от компьютера. Вызов функции осуществляется при обработке сообщения WM_DEVICE.

SLCAN_DEVOP_DESTROY
Значение параметра указывает на уничтожение объекта устройства и исключения его из списка устройств. Это происходит в следующих случаях:

1. Устройство подключено к компьютеру. Библиотека выгружается. Вызов функции осуществляется во время исполнения функции SlCan_Free с параметром bDoCallBack=TRUE.
2. Библиотека загружена. Устройство отключается от компьютера. Вызов функции осуществляется при обработке сообщения WM_DEVICE.

SlCan_DeviceList_CallBack

Тип callback функции изменения состояния списка устройств slcan.

C

```
typedef  VOID (STDCALL* SLCAN_DEVICELIST_CALLBACK)(
    HSLCAN     hDevice,
    DWORD      dwIndex,
    PVOID      pContext,
    DWORD      dwContextSize
);
```

Pascal

```
type
   TSlCan_DeviceListCallBack = procedure(
      dwDevice:TSlCanDevice;
      dwIndex:longword;
      pContext:pointer;
      dwContextSize:longword
      );stdcall;
```

C#

```
public delegate void DeviceListCallBack(IntPtr hDevice, uint dwIndex, IntPtr pContext,
uint dwContextSize);
```

VB

```
Public Delegate Sub DeviceListCallBack(ByVal hDevice As IntPtr, ByVal dwIndex As
UInteger, ByVal pContext As IntPtr, ByVal dwContextSize As UInteger)
```

Параметры

hDevice

Ссылка на устройство. Может принимать значение 0xFFFFFFFF.

dwIndex

Номер устройства в списке устройств. Может принимать значение 0xFFFFFFFF.

pContext

Указатель на контекст устройства.

dwContextSize

Размер контекста устройства в байтах.

Замечание

Для того чтобы использовать callback функцию устройства, необходимо при инициализации библиотеки указать адрес функции данного типа в качестве второго параметра при вызове функции SlCan_Load. Тогда она будет вызываться ядром библиотеки при изменении состояния списка объектов устройств.

Если параметр hDevice равен 0xFFFFFFFF, то это означает, что список устройств изменился.

Если параметр hDevice не равен 0xFFFFFFFF, то это означает, что вновь подключенное устройство уже добавлено в список и с ним можно работать, например, открыть и передавать и принимать фреймы.

SICan_DeviceCapabilities

Структура значений возможностей устройства.

C

```
typedef    struct _SLCAN_CAPABILITIES{
    BYTE    bAvaibleModes;
    BYTE    bAvaibleTXMode;
    BYTE    bMaxEventLevel;
    BYTE    bController;
    BYTE    bPhysical;
    BYTE    bPhysicalLoad;
    BYTE    bBitrates;
    BYTE    bAdvancedModes;
    DWORD   dwBaseClk;
    DWORD   dwTimeStampClk;
    WORD    wMaxBRP;
}SLCAN_CAPABILITIES,*PSLCAN_CAPABILITIES;
```

Pascal

```
type
PSlCanCapabilities = ^TSlCanCapabilities;
TSlCanCapabilities = packed record
    bAvaibleModes: byte;
    bAvaibleTXMode:byte;
    bMaxEventLevel:byte;
    bController:byte;
    bPhysical:byte;
    bPhysicalLoad:byte;
    bBitrates:byte;
    bAdvancedModes:byte;
    dwBaseClk:longword;
    dwTimeStampClk:longword;
    wMaxBrp:word;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Capability
{
    public byte bAvaibleModes;
    public byte bAvaibleTXMode;
    public byte MaxEventLevel;
    public byte bController;
    public byte bPhysical;
    public byte bPhysicalLoad;
    public byte bBitrates;
    public byte bAdvancedModes;
    public uint dwBaseClk;
    public uint dwTimeStampClk;
    public ushort wMaxBrp;
}
```

VB

```
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure Capability
    Public bModes As Byte
    Public bTXModes As Byte
    Public bMaxEventLevel As Byte
    Public bController As Byte
    Public bPhysical As Byte
    Public bPhysicalLoad As Byte
    Public bBitrates As Byte
    Public bAdvancedModes As Byte
    Public dwCanBaseClk As UInteger
    Public dwTimeStampClk As UInteger
```

```
    Public wMaxBRP As UShort
End Structure
```

Поля

bModes

Поддерживаемые режимы работы устройства. Может быть комбинацией следующих значений:

Константа	Значение	Режим
SLCAN_CAP_MODE_NORMAL	0x01	Нормальный режим работы. Устройство может посылать и принимать CAN фреймы.
SLCAN_CAP_MODE_LISTEN_ONLY	0x02	Режим прослушивания. Устройство может только принимать фреймы и ничем не обнаруживает себя в сети.
SLCAN_CAP_MODE_LOOP_BACK	0x04	Режим самопрослушивания. Устройство принимает посланные самим собой фреймы.
SLCAN_CAP_MODE_SLEEP	0x08	Устройство может переходить в спящий режим.

bTXModes

Поддерживаемые расширенные режимы передачи. Может быть комбинацией следующих значений:

Константа	Значение	Режим
SLCAN_CAP_TXMODE_ONE_SHOT	0x01	Режим однократной передачи. Т.е. в случае возникновения ошибки, CAN контроллер не производит повтор передачи.
SLCAN_CAP_TXMODE_TIME_STAMP	0x02	Устройство может сформировать задержку в передаче текущего или последующего фрейма.

bMaxEventLevel

Максимальный уровень событий, поддерживаемых устройством.

bController

Тип CAN контроллера устройства. Может быть одним из следующих значений

Константа	Значение
SLCAN_CAP_CONTR_MCP2515	0x01
SLCAN_CAP_CONTR_SJA1000	0x02
SLCAN_CAP_CONTR_LPC17	0x81
SLCAN_CAP_CONTR_STM32	0x82
SLCAN_CAP_CONTR_STM8	0x83
SLCAN_CAP_CONTR_PIC	0x84
SLCAN_CAP_CONTR_PIC_ECAN	0x85

bPhysical

Тип интерфейса CAN устройства. Может быть комбинацией следующих значений:

Константа	Значение	Тип интерфейса
SLCAN_CAP_ PHYS_HS	0x01	Высокоскоростной до 1Мб/сек согласно ISO11898-2
SLCAN_CAP_ PHYS_LS	0x02	Низкоскоростной до 125Кб/сек отказоустойчивый согласно ISO11898-3
SLCAN_CAP_ PHYS_SW	0x04	Однопроводной до 83333б/сек согласно SAE J2411
SLCAN_CAP_ PHYS_J1708	0x08	RS485 приемопередатчик с включением согласно SAE J1708
SLCAN_CAP_ PHYS_LIN	0x10	LIN приемопередатчик (SAE J2602).
SLCAN_CAP_ PHYS_KLINE	0x20	Приемопередатчик ISO1941

SLCAN_CAP_ PHYS_ISO11992	0x40	Интерфейс ISO11992

bPhysicalLoad

Зарезервировано. Всегда 0.

bBitrates

Тип настройки скорости передачи, который поддерживается устройством. Может быть комбинацией следующих значений:

Константа	Значение	Тип настройки
SLCAN_CAP_BITRATE_INDEX	0x01	Устройство поддерживает предопределенные настройки скорости
SLCAN_CAP_BITRATE_CUSTOM	0x02	Устройство позволяет настраивать скорость передачи
SLCAN_CAP_BITRATE_AUTOMATIC	0x04	Устройство может автоматически настроить параметры скорости передачи.

Более подробно см. SlCan_BitRate

bAdvancedModes

Зарезервировано. Всегда 0.

dwBaseClk

Частота тактового генератора устройства в Гц.

dwTimeStampClk

Частота генератора меток времени устройства в Гц.

wMaxBrp

Максимальное возможное значение делителя частоты тактового генератора устройства. Используется для формирования кванта времени.

SICan_BitRate

Тип структуры параметров битовой скорости передачи CAN устройства slcan

C

```
typedef struct   _SLCAN_BITRATE {
    WORD    BRP;
    BYTE    TSEG1;
    BYTE    TSEG2;
    BYTE    SJW;
    BYTE    SAM;
}SLCAN_BITRATE,*PSLCAN_BITRATE;
```

Pascal

```
type
PSlCanBitRate = ^TSlCanBitRate;
TSlCanBitRate = packed record
    BRP:word;
    TSEG1:byte;
    TSEG2:byte;
    SJW:byte;
    SAM:byte;
end;
```

C#

```
public struct BitRate
{
    public ushort BRP;
    public byte TSEG1;
    public byte TSEG2;
    public byte SJW;
    public byte SAM;
}
```

VB

```
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure BitRate
    Public BRP As UShort
    Public TSEG1 As Byte
    Public TSEG2 As Byte
    Public SJW As Byte
    Public SAM As Byte
End Structure
```

Поля

BRP

Если старший бит поля равен 0 , то поле определяет значение делителя частоты генератора устройства для формирования кванта времени, который используется для получения номинальной длительности бита.

Если старший бит поля равен 1, то оставшиеся младшие биты определяют номер предустановленных значений параметров формирователя скорости передачи. Может принимать следующие значения:

Константа	Значение	Скорость
SLCAN_BR_CIA_1000K	0x8000	1000 кбит/сек
SLCAN_BR_CIA_800K	0x8001	800 кбит/сек
SLCAN_BR_CIA_500K	0x8002	500 кбит/сек
SLCAN_BR_CIA_250K	0x8003	250 кбит/сек
SLCAN_BR_CIA_125K	0x8004	125 кбит/сек
SLCAN_BR_CIA_50K	0x8005	50 кбит/сек
SLCAN_BR_CIA_20K	0x8006	20 кбит/сек
SLCAN_BR_CIA_10K	0x8007	150 кбит/сек

SLCAN_BR_400K	0x8008	400 кбит/сек
SLCAN_BR_200K	0x8009	200 кбит/сек
SLCAN_BR_100K	0x800A	100 кбит/сек
SLCAN_BR_83333	0x800B	83333 бит/сек
SLCAN_BR_33333	0x800C	33333 бит/сек
SLCAN_BR_25K	0x800D	25 кбит/сек
SLCAN_BR_5K	0x800E	5 кбит/сек

TSEG1

Длительность сегмента TSEG1 в квантах времени. Допустимые значения в пределах 2..16. Значение не должно быть меньше bSJW.

TSEG2

Длительность сегмента TSEG2 в квантах времени. Допустимые значения в пределах 1..8. Значение не должно быть меньше bSJW

SJW

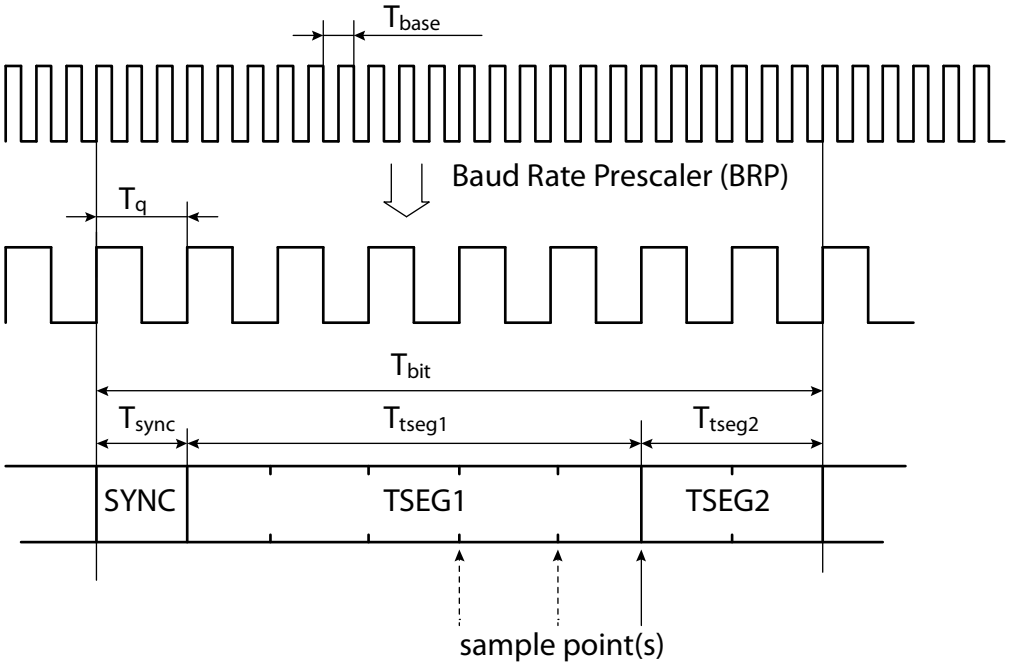
Длительность ширины перехода пересинхронизации. Допустимые значения 1..4.

SAM

Количество точек определения значения бита. Если 0, то значение бита определяется в одной точке. Если >0, то

Замечание

Значение параметров настройки скорости передачи поясняет рисунок.



Скорость передачи бит CAN фрейма V в бит/сек определяется как:

$$V = \frac{1}{T_{bit}} \qquad (1)$$

где – T_{bit} номинальная длительность бита в секундах.

Номинальная длительность бита T_{bit} кратна кванту времени T_q. Квант времени – это период тактового генератора с частотой в BRP раз меньшей, чем частота базового генератора устройства F_{base}, т.е. длительность бита связана с базовой частотой соотношением

$$T_{bit} = Q \times T_q = Q \times \frac{BRP}{F_{base}} \qquad (2)$$

где Q – количество квантов времени в бите, а BRP – значение делителя базовой частоты.

Количество квантов времени Q в бите является суммой трех сегментов SYNC, TSEG1, TSEG2. Предпочтительные значения Q должны лежать в пределах 8..25. Соотношение сегментов TSEG1 и TSEG2 определяет относительное положение точки измерения значения бита. Значение бита определяется в конце сегмента TSEG1 один раз или по мажоритарной системе из трех определений с периодом Tq.

Сегмент SYNC всегда равен 1. Таким образом, номинальная длительность бита определяется по формуле:

$$T_{bit} = \frac{(1 + TSEG1 + TSEG2) \times BRP}{F_{base}} \quad (3)$$

а скорость передачи по формуле

$$V = \frac{F_{base}}{(1 + TSEG1 + TSEG2) \times BRP} \quad (4)$$

При этом на значения TSEG1 и TSEG2 накладываются следующие ограничения:

$$\begin{aligned} 2 \le TSEG1 \le 16 \\ 1 \le TSEG2 \le 8 \end{aligned} \quad (5)$$

Для компенсации фазового сдвига между тактовыми генераторами различных узлов CAN сети бит может укоротиться или удлиниться на время пересинхронизации SJW. Количество квантов времени определяет длительность времени пересинхронизации, при этом SJW может принимать значения:

$$\begin{aligned} 1 \le SJW \le 4 \\ SJW \le TSEG2 \end{aligned} \quad (6)$$

Соответствие значений параметров настройки скорости передачи и полей структур SlCan_BitRate и SlCan_DeviceCapabilities показано в таблице:

Параметр	Поле	Описание
F_{base}	SlCan_DeviceCapabilities.dwBaseClk	Базовая частота, т.е. частота тактового генератора устройства в Гц.
	SlCan_DeviceCapabilities.wMaxBrp	Максимально возможное значение делителя базовой частоты.
TSEG1	SlCan_BitRate.TSEG1	Длительность сегмента TSEG1 в квантах времени.
TSEG2	SlCan_BitRate.TSEG2	Длительность сегмента TSEG2 в квантах времени.
BRP	SlCan_BitRate.BRP (<0x8000)	Делитель тактовой частоты.
SJW	SlCan_BitRate.SJW	Длительность времени пересинхронизации в квантах времени.
	SlCan_BitRate.SAM	Количество определений значения бита.

Таким образом, используя значения полей формулы (3) и (4) можно записать так:

$$T_{bit} = \frac{(1 + SlCan\_Bitrate.TSEG1 + SlCan\_Bitrate.TSEG2) \times SlCan\_Bitrate.BRP}{SlCan\_DeviceCapabilities.dwBaseClk} \quad (3a)$$

$$V = \frac{SlCan\_DeviceCapabilities.dwBaseClk}{(1 + SlCan\_Bitrate.TSEG1 + SlCan\_Bitrate.TSEG2) \times SlCan\_Bitrate.wBRP} \quad (4a)$$

При этом:

$$\begin{aligned} 2 \le SlCan\_BitRate.TSEG1 \le 16 \\ 1 \le SlCan\_BitRate.TSEG2 \le 8 \\ SlCan\_BitRate.SJW \le SlCan\_BitRate.TSEG2 \\ SlCan\_BitRate.BRP \le SlCan\_DeviceCapabilities.wMaxBrp \end{aligned} \quad (5a)$$

SICan_Message

Тип структуры CAN сообщения

C

```
typedef struct _SLCAN_MESSAGE{
    BYTE    Info;
    DWORD   ID;
    BYTE    DataCount;
    BYTE    Data[8];
}SLCAN_MESSAGE,*PSLCAN_MESSAGE;
```

Pascal

```
type
PSlCanMessage = ^TSlCanMessage;
TSlCanMessage = packed record
    Info:byte;
    ID:longword;
    DataCount:byte;
    Data:array [0..7] of byte;
  end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Message
{
    public byte Info;
    public UInt32 ID;
    public byte DataCount;
    public byte Data0;
    public byte Data1;
    public byte Data2;
    public byte Data3;
    public byte Data4;
    public byte Data5;
    public byte Data6;
    public byte Data7;
}
```

VB

```
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure Message
    Public Info As Byte
    Public ID As UInt32
    Public DataCount As Byte
    Public Data0 As Byte
    Public Data1 As Byte
    Public Data2 As Byte
    Public Data3 As Byte
    Public Data4 As Byte
    Public Data5 As Byte
    Public Data6 As Byte
    Public Data7 As Byte
End Structure
```

Поля

Info

Битовое поле, определяющее тип CAN фрейма. Может быть комбинацией следующих значений:

Константа	Значение	Тип настройки
SLCAN_MES_INFO_EXT	0x01	Бит определяет формат CAN фрейма. Если 1, то фрейм расширенного формата с 29-битовым идентификатором. Если 0, то

		фрейм стандартного формата с 11-битовым идентификатором.
SLCAN_MES_INFO_RTR	0x02	Бит определяет, является ли фрейм запросом передачи данных Remout Transmit Request. Если 1, то является, если 0, то нет.
SLCAN_MES_INFO_ONESHOT	0x04	Бит определяет передачу фрейма в One Shot режиме. Бит является значащим, если устройство может поддерживать данный режим передачи и используется функция SlCan_DeviceWriteMesageEx.

ID

Идентификатор CAN фрейма.

DataCount

Значение поля DLC CAN фрейма.

Data

Значения поля данных CAN фрейма.

SICan_TxMessage

Тип структуры CAN сообщения для расширенной функции передачи

C

```
typedef struct _SLCAN_TXMESSAGE{
    LONG            Delay;
    SLCAN_MESSAGE   Msg;
}SLCAN_TXMESSAGE,*PSLCAN_TXMESSAGE;
```

Pascal

```
type
PSlCanTxMessage = ^TSlCanTxMessage;
TSlCanTxMessage = packed record
   Delay:longint;
   Msg:TSlCanMessage;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct TxMessage
{
    Public Delay;
    public byte Info;
    public UInt32 ID;
    public byte DataCount;
    public byte Data0;
    public byte Data1;
    public byte Data2;
    public byte Data3;
    public byte Data4;
    public byte Data5;
    public byte Data6;
    public byte Data7;
}
```

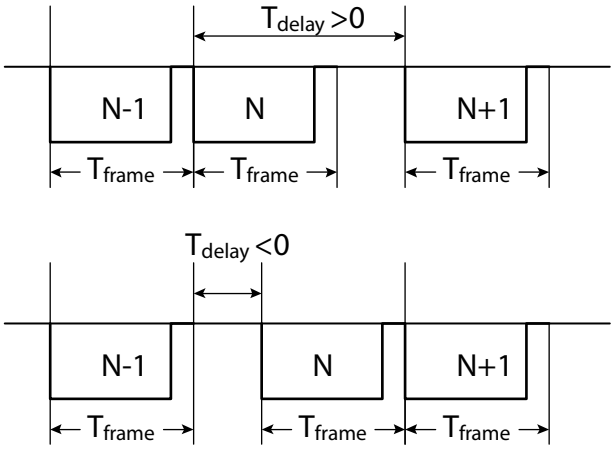
VB

```
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure TxMessage
    Public Delay As Int32
    Public Info As Byte
    Public ID As UInt32
    Public DataCount As Byte
    Public Data0 As Byte
    Public Data1 As Byte
    Public Data2 As Byte
    Public Data3 As Byte
    Public Data4 As Byte
    Public Data5 As Byte
    Public Data6 As Byte
    Public Data7 As Byte
End Structure
```

Поля

Delay

Значение временной задержки в микросекундах от начала передачи данного фрейма. Если значение положительно, то задерживается следующий фрейм, если значение отрицательно, то текущий. Назначение задержки поясняет рисунок



Msg

CAN сообщение

SICan_Event

Тип структуры события.

C

```
typedef struct _SLCAN_EVENT{
    BYTE    EventType;
    DWORD   TimeStampLo;
    union {
        SLCAN_MESSAGE Msg;
        DWORD TimeStamp[2];
        DWORD64 TimeStamp64;
        struct {
            BYTE BusMode;
            BYTE Dummy1;
            BYTE ErrCountRx;
            BYTE ErrCountTx;
            BYTE ErrType;
            BYTE ErrDir;
            BYTE ErrFrame;
            BYTE LostArbitration;
        };
    };
}SLCAN_EVENT,*PSLCAN_EVENT;
```

Pascal

```
type
PSlCanEvent = ^TSlCanEvent;
TSlCanEvent = packed record
   EventType:byte;
   TimeStampLo:longword;
   case integer of
   0:(Msg:TSlCanMessage);
   1:(TimeStamp: array [0 .. 1] of longword);
   2:(TimeStamp64: Uint64);
   3:(BusMode:byte;
      Dummy1:byte;
      ErrCountRX:byte;
      ErrCountTX:byte;
      ErrType:byte;
      ErrDir:byte;
      ErrFrame:byte;
      LostArbitration:byte);
end;
```

C#

Смотри объявление данной структуры SICan.Event в slcanclass.cs

VB

Смотри объявление данной структуры SICan.CEvent в slcanclass.vb

Поля

EventType

Тип события, может принимать следующие значения:

Константа	Значение	Событие
SLCAN_EVT_TYPE_RX	0x00	Принят CAN фрейм. Принятый фрейм находится в поле Msg.
SLCAN_EVT_TYPE_START_TX	0x01	Начата передача CAN фрейма. Передающийся фрейм находится в поле Msg.
SLCAN_EVT_TYPE_END_TX	0x02	Передача CAN фрейма успешно завершена. Переданный фрейм

находится в поле Msg.

SLCAN_EVT_TYPE_ABORT_TX	0x03	Передача CAN фрейма отменена. Отменённый фрейм находится в поле Msg.
SLCAN_EVT_TYPE_BUS_STATE	0x04	Изменилось состояние CAN контроллера. Значение текущего состояния находится в поле BusMode
SLCAN_EVT_TYPE_ERROR_COUNTS	0x05	Изменилось значение одного из счетчика ошибок. Значения счетчиков ошибок приема и передачи находятся в полях ErrCountRX и ErrCountTx.
SLCAN_EVT_TYPE_BUS_ERROR	0x06	Произошла ошибка шины CAN. Тип ошибки находится в поле ErrType. Произошла ошибка при приеме или передаче определяет поле ErrDir. В каком месте фрейма произошла ошибка, определяет поле ErrFrame.
SLCAN_EVT_TYPE_ARBITRATION	0x07	Произошла ошибка арбитража. Номер бита находится в поле LostArbitration
SLCAN_EVT_TYPE_STAMP_INC	0x0F	Изменились старшие 40 бит счетчика временных меток. Значение временной метки находится в полях TimeStamp64 и TimeStamp/

TimeStampLo:
Младшие 24-бита временной метки события.

Msg
CAN фрейм. Поле определено только, если тип события равен одному из значений: SLCAN_EVT_TYPE_RX, SLCAN_EVT_TYPE_START_TX, SLCAN_EVT_TYPE_END_TX, SLCAN_EVT_TYPE_ABORT_TX.

TimeStamp64
Значение временной метки. Определено, если тип события равен SLCAN_EVT_TYPE_STAMP_INC.

TimeStamp
Массив, определяющий значение временной метки. В первом элементе массива содержатся младшие 32 бита, во втором старшие 32 бита. Поле определено, если тип события равен SLCAN_EVT_TYPE_STAMP_INC.

BusMode
Состояние CAN контроллера устройства. Может принимать следующие значения:

Константа	Значение	Состояние
SLCAN_BUS_STATE_ERROR_ACTIVE	0x00	CAN контроллер находится в состоянии ERROR ACTIVE. Значения счетчиков ошибок меньше 96.
SLCAN_BUS_STATE_ERROR_ACTIVE_WARN	0x01	CAN контроллер находится в состоянии ERROR ACTIVE. Значение одного из счетчиков ошибок больше или равно 96.
SLCAN_BUS_STATE_ERROR_PASSIVE	0x02	CAN контроллер находится в состоянии ERROR PASSIVE. Значение одного из счетчиков ошибок больше 127.
SLCAN_BUS_STATE_BUSOFF	0x03	CAN контроллер находится в состоянии BUS OFF. Контроллер переходит в это состояние при превышении счетчика ошибок передачи значения 256 или при переводе в режим работы SLCAN_MODE_CONFIG

Поле определено только, если тип события равен одному из значений:
SLCAN_EVT_TYPE_BUS_STATE, SLCAN_EVT_TYPE_ERROR_COUNTS,
SLCAN_EVT_TYPE_BUS_ERROR, SLCAN_EVT_TYPE_ARBITRATION

Dummy1

Назначение поля зарезервировано.

ErrCountRX

Значение счетчика ошибок приема. Поле определено только, если тип события равен одному из значений: SLCAN_EVT_TYPE_ERROR_COUNTS, SLCAN_EVT_TYPE_BUS_ERROR, SLCAN_EVT_TYPE_ARBITRATION.

ErrCountTX

Значение счетчика ошибок передачи. Поле определено только, если тип события равен одному из значений: SLCAN_EVT_TYPE_ERROR_COUNTS, SLCAN_EVT_TYPE_BUS_ERROR, SLCAN_EVT_TYPE_ARBITRATION.

ErrType

Значение типа ошибки CAN шины. Может принимать следующие значения:

Константа	Значение	Тип ошибки
SLCAN_EVT_ERR_TYPE_BIT	0x00	Битовая ошибка. Возникает, когда при передачи вместо рецессивного бита диагностируется доминантный.
SLCAN_EVT_ERR_TYPE_FORM	0x01	Ошибка формы. Возникает, когда в фиксированных полях фрейма диагностируется недопустимый бит.
SLCAN_EVT_ERR_TYPE_STUFF	0x02	Ошибка битовой подстановки (битстаффинга). Возникает, когда во фрейме диагностируется шесть последовательных бит одного значения.
SLCAN_EVT_ERR_TYPE_OTHER	0x03	Другие ошибки – ошибка CRC или ACK. Ошибка CRC возникает, когда вычисленное приемником значение CRC отличается от принятого. Ошибка ACK возникает, когда в ACK слоте не диагностируется доминантный бит.

Поле определено только, если тип события равен значению SLCAN_EVT_TYPE_BUS_ERROR.

ErrDir

Значение источника ошибки CAN шины. Может принимать следующие значения:

Константа	Значение	Тип ошибки
SLCAN_EVT_ERR_DIR_TX	0x00	Ошибка возникла при передаче.
SLCAN_EVT_ERR_DIR_RX	0x01	Ошибка возникла при приёме.

Поле определено только, если тип события равен значению SLCAN_EVT_TYPE_BUS_ERROR.

ErrFrame

Значение места ошибки CAN шины во фрейме. Может принимать следующие значения:

SLCAN_EVT_ERR_FRAME_SOF	0x03
SLCAN_EVT_ERR_FRAME_ID28_ID21	0x02
SLCAN_EVT_ERR_FRAME_ID20_ID18	0x06
SLCAN_EVT_ERR_FRAME_SRTR	0x04
SLCAN_EVT_ERR_FRAME_IDE	0x05

SLCAN_EVT_ERR_FRAME_ID17_ID13	0x07
SLCAN_EVT_ERR_FRAME_ID12_ID5	0x0F
SLCAN_EVT_ERR_FRAME_ID4_ID0	0x0E
SLCAN_EVT_ERR_FRAME_RTR	0x0C
SLCAN_EVT_ERR_FRAME_RSRV0	0x0D
SLCAN_EVT_ERR_FRAME_RSRV1	0x09
SLCAN_EVT_ERR_FRAME_DLC	0x0B
SLCAN_EVT_ERR_FRAME_DATA	0x0A
SLCAN_EVT_ERR_FRAME_CRC_SEQ	0x08
SLCAN_EVT_ERR_FRAME_CRC_DEL	0x18
SLCAN_EVT_ERR_FRAME_ACK_SLOT	0x19
SLCAN_EVT_ERR_FRAME_ACK_DEL	0x1B
SLCAN_EVT_ERR_FRAME_EOF	0x1A
SLCAN_EVT_ERR_FRAME_INTER	0x12
SLCAN_EVT_ERR_FRAME_AER_FLAG	0x11
SLCAN_EVT_ERR_FRAME_PER_FLAG	0x16
SLCAN_EVT_ERR_FRAME_TDB	0x13
SLCAN_EVT_ERR_FRAME_ERR_DEL	0x17
SLCAN_EVT_ERR_FRAME_OVER_FLAG	0x1C

Поле определено только, если тип события равен значению SLCAN_EVT_TYPE_BUS_ERROR.

LostArbitration

Номер бита поля идентификатора фрейма, после которого дальнейшая передача фрейма приостановлена из-за диагностирования в данном бите доминантного уровня, когда был передан рецессивный. Поле определено только, если тип события равен значению SLCAN_EVT_TYPE_ARBITRATION.

SICan_State

Тип структуры события.

C

```
typedef struct _SLCAN_STATE{
    BYTE BusMode;
    BYTE Dummy1;
    BYTE ErrCountRX;
    BYTE ErrCountTX;
}SLCAN_STATE,*PSLCAN_STATE;
```

Pascal

```
type
PSlCanState = ^TSlCanState;
TSlCanState = packed record
    BusMode:byte;
    Dummy1:byte;
    ErrCountRX:byte;
    ErrCountTX:byte;
end;
```

C#

```
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct State
{
    public byte BusMode;
    public byte Dummy1;
    public byte ErrCountRX;
    public byte ErrCountTX;

}
```

VB

```
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
    Public Structure State
    Public BusMode As Byte
    Public Dummy1 As Byte
    Public ErrCountRX As Byte
    Public ErrCountTX As Byte
End Structure
```

Поля

BusMode

Состояние CAN контроллера устройства. Может принимать следующие значения:

Константа	Значение	Состояние
SLCAN_BUS_STATE_ERROR_ACTIVE	0x00	CAN контроллер находится в состоянии ERROR ACTIVE. Значения счетчиков ошибок меньше 96.
SLCAN_BUS_STATE_ERROR_ACTIVE_WARN	0x01	CAN контроллер находится в состоянии ERROR ACTIVE. Значение одного из счетчиков ошибок больше или равно 96.
SLCAN_BUS_STATE_ERROR_PASSIVE	0x02	CAN контроллер находится в состоянии ERROR PASSIVE. Значение одного из счетчиков ошибок больше 127.
SLCAN_BUS_STATE_BUSOFF	0x03	CAN контроллер находится в состоянии BUS OFF. Контроллер переходит в это состояние при превышении счетчика ошибок

передачи значения 256 или при переводе в режим работы SLCAN_MODE_CONFIG

ErrCountRX
Значение счетчика ошибок приема.

ErrCountTX
Значение счетчика ошибок передачи.

Приложения

Программные модули

slcan.h

```
#ifndef     __SLCAN_H__
#define     __SLCAN_H__

#include <windows.h>

#define     STDCALL  __stdcall
#define     SLCANAPI __declspec(dllimport)

#define     SLCAN_PROPERTY_INDEX_LINKNAME        0
#define     SLCAN_PROPERTY_INDEX_INSTANCEID      1

#define     SLCAN_PROPERTY_INDEX_DEVICEDESC      2
#define     SLCAN_PROPERTY_INDEX_FRIENDLYNAME    3
#define     SLCAN_PROPERTY_INDEX_PHOBJECTNAME    4
#define     SLCAN_PROPERTY_INDEX_MFG             5
#define     SLCAN_PROPERTY_INDEX_LOCATIONINFO    6
#define     SLCAN_PROPERTY_INDEX_ENUMERATOR      7
#define     SLCAN_PROPERTY_INDEX_CLASS           8
#define     SLCAN_PROPERTY_INDEX_CLASSGUID       9
#define     SLCAN_PROPERTY_INDEX_SERVICE         10
#define     SLCAN_PROPERTY_INDEX_DRIVER          11
#define     SLCAN_PROPERTY_INDEX_PORTNAME        12
#define     SLCAN_PROPERTY_INDEX_PRODUCT         13L
#define     SLCAN_PROPERTY_INDEX_MANUFACTURER    14L
#define     SLCAN_PROPERTY_INDEX_CONFIGURATION   15L
#define     SLCAN_PROPERTY_INDEX_INTERFACE       16L
#define     SLCAN_PROPERTY_INDEX_SERIAL          17L
#define     SLCAN_PROPERTY_INDEX_ALIAS           18L
#define     SLCAN_PROPERTY_INDEX_CHANNELLINK     19L
#define     SLCAN_PROPERTY_INDEX_SERIALID        20L

#define     SLCAN_MODE_CONFIG                   0x00
#define     SLCAN_MODE_NORMAL                   0x01
#define     SLCAN_MODE_LISTENONLY               0x02
#define     SLCAN_MODE_LOOPBACK                 0x03
#define     SLCAN_MODE_SLEEP                    0x04

#define     SLCAN_BR_CIA_1000K                    0x8000
#define     SLCAN_BR_CIA_800K                     0x8001
#define     SLCAN_BR_CIA_500K                     0x8002
#define     SLCAN_BR_CIA_250K                     0x8003
#define     SLCAN_BR_CIA_125K                     0x8004
#define     SLCAN_BR_CIA_50K                      0x8005
#define     SLCAN_BR_CIA_20K                      0x8006
#define     SLCAN_BR_CIA_10K                      0x8007
#define     SLCAN_BR_400K                         0x8008
#define     SLCAN_BR_200K                         0x8009
#define     SLCAN_BR_100K                         0x800A
#define     SLCAN_BR_83333                        0x800B
#define     SLCAN_BR_33333                        0x800C
#define     SLCAN_BR_25K                          0x800D
#define     SLCAN_BR_5K                           0x800E
#define     SLCAN_BR_30K                          0x800F
#define     SLCAN_BR_300K                         0x8010
#define     SLCAN_BR_LASTINDEX                  SLCAN_BR_300K

#define     SLCAN_CAP_MODE_NORMAL                0x01
```

```
#define     SLCAN_CAP_MODE_LISTEN_ONLY          0x02
#define     SLCAN_CAP_MODE_LOOP_BACK            0x04
#define     SLCAN_CAP_MODE_SLEEP                0x08

#define     SLCAN_CAP_TXMODE_ONE_SHOT           0x01
#define     SLCAN_CAP_TXMODE_TIMESTAMP          0x02

#define     SLCAN_CAP_CONTR_EXTERNAL            0x00
#define     SLCAN_CAP_CONTR_MCP2515             0x01
#define     SLCAN_CAP_CONTR_SJA1000             0x02

#define     SLCAN_CAP_CONTR_INTERNAL            0x80
#define     SLCAN_CAP_CONTR_LPC                 0x81
#define     SLCAN_CAP_CONTR_STM32               0x82
#define     SLCAN_CAP_CONTR_STM8                0x83
#define     SLCAN_CAP_CONTR_PIC                 0x84
#define     SLCAN_CAP_CONTR_PIC_ECAN            0x85

#define     SLCAN_CAP_PHYS_HS                   0x01
#define     SLCAN_CAP_PHYS_LS                   0x02
#define     SLCAN_CAP_PHYS_SW                   0x04
#define     SLCAN_CAP_PHYS_J1708                0x08
#define     SLCAN_CAP_PHYS_LIN                  0x10
#define     SLCAN_CAP_PHYS_KLINE                0x20

#define     SLCAN_CAP_PHYS_LOAD                 0x01

#define     SLCAN_CAP_BITRATE_INDEX             0x01
#define     SLCAN_CAP_BITRATE_CUSTOM            0x02
#define     SLCAN_CAP_BITRATE_AUTOMATIC         0x04

#define     SLCAN_EVT_LEVEL_RX_MSG              0
#define     SLCAN_EVT_LEVEL_TIME_STAMP          1
#define     SLCAN_EVT_LEVEL_TX_MSG              2
#define     SLCAN_EVT_LEVEL_BUS_STATE           3
#define     SLCAN_EVT_LEVEL_COUNTS              4
#define     SLCAN_EVT_LEVEL_ERRORS              5

#define     SLCAN_EVT_TYPE_RX                   0x0
#define     SLCAN_EVT_TYPE_START_TX             0x1
#define     SLCAN_EVT_TYPE_END_TX               0x2
#define     SLCAN_EVT_TYPE_ABORT_TX             0x3
#define     SLCAN_EVT_TYPE_BUS_STATE            0x4
#define     SLCAN_EVT_TYPE_ERROR_COUNTS         0x5
#define     SLCAN_EVT_TYPE_BUS_ERROR            0x6
#define     SLCAN_EVT_TYPE_ARBITRATION    0x7
#define     SLCAN_EVT_STAMP_INC                 0xF

#define     SLCAN_BUS_STATE_ERROR_ACTIVE        0x00
#define     SLCAN_BUS_STATE_ERROR_ACTIVE_WARN   0x01
#define     SLCAN_BUS_STATE_ERROR_PASSIVE       0x02
#define     SLCAN_BUS_STATE_BUSOFF              0x03

#define     SLCAN_MES_INFO_EXT              0x01
#define     SLCAN_MES_INFO_RTR              0x02
#define     SLCAN_MES_INFO_ONESHOT          0x04

#define     SLCAN_DEVOP_CREATE              0x00000000
#define     SLCAN_DEVOP_CREATEHANDLE        0x00000001
#define     SLCAN_DEVOP_OPEN                0x00000002
#define     SLCAN_DEVOP_CLOSE               0x00000003
#define     SLCAN_DEVOP_DESTROYHANDLE       0x00000004
#define     SLCAN_DEVOP_DESTROY             0x00000005
```

```
#define     SLCAN_INVALID_HANDLE_ERROR          0xE0001001
#define     SLCAN_DEVICE_INVALID_HANDLE_ERROR   0xE0001120
#define     SLCAN_HANDLE_INIT_ERROR             0xE0001017
#define     SLCAN_DEVICE_NOTOPEN_ERROR          0xE0001121

#define     SLCAN_EVT_ERR_TYPE_BIT               0x00
#define     SLCAN_EVT_ERR_TYPE_FORM              0x01
#define     SLCAN_EVT_ERR_TYPE_STUFF             0x02
#define     SLCAN_EVT_ERR_TYPE_OTHER             0x03

#define     SLCAN_EVT_ERR_DIR_TX                 0x00
#define     SLCAN_EVT_ERR_DIR_RX                 0x01

#define     SLCAN_EVT_ERR_FRAME_SOF              0x03
#define     SLCAN_EVT_ERR_FRAME_ID28_ID21        0x02
#define     SLCAN_EVT_ERR_FRAME_ID20_ID18        0x06
#define     SLCAN_EVT_ERR_FRAME_SRTR             0x04
#define     SLCAN_EVT_ERR_FRAME_IDE              0x05
#define     SLCAN_EVT_ERR_FRAME_ID17_ID13        0x07
#define     SLCAN_EVT_ERR_FRAME_ID12_ID5         0x0F
#define     SLCAN_EVT_ERR_FRAME_ID4_ID0          0x0E
#define     SLCAN_EVT_ERR_FRAME_RTR              0x0C
#define     SLCAN_EVT_ERR_FRAME_RSRV0            0x0D
#define     SLCAN_EVT_ERR_FRAME_RSRV1            0x09
#define     SLCAN_EVT_ERR_FRAME_DLC              0x0B
#define     SLCAN_EVT_ERR_FRAME_DATA             0x0A
#define     SLCAN_EVT_ERR_FRAME_CRC_SEQ          0x08
#define     SLCAN_EVT_ERR_FRAME_CRC_DEL          0x18
#define     SLCAN_EVT_ERR_FRAME_ACK_SLOT         0x19
#define     SLCAN_EVT_ERR_FRAME_ACK_DEL          0x1B
#define     SLCAN_EVT_ERR_FRAME_EOF              0x1A
#define     SLCAN_EVT_ERR_FRAME_INTER            0x12
#define     SLCAN_EVT_ERR_FRAME_AER_FLAG         0x11
#define     SLCAN_EVT_ERR_FRAME_PER_FLAG         0x16
#define     SLCAN_EVT_ERR_FRAME_TDB              0x13
#define     SLCAN_EVT_ERR_FRAME_ERR_DEL          0x17
#define     SLCAN_EVT_ERR_FRAME_OVER_FLAG        0x1C

#define     SLCAN_TX_STATUS_OK                   0x00
#define     SLCAN_TX_STATUS_TIMEOUT              0x01
#define     SLCAN_TX_STATUS_BUSOFF               0x02
#define     SLCAN_TX_STATUS_ABORT                0x03
#define     SLCAN_TX_STATUS_NOT_ENA              0x04
#define     SLCAN_TX_STATUS_ERROR_ONE_SHOT       0x05
#define     SLCAN_TX_STATUS_INVALID_MODE         0x06
#define     SLCAN_TX_STATUS_UNKNOWN              0x0F

#define     SLCAN_PURGE_TX_ABORT                 0x01
#define     SLCAN_PURGE_RX_ABORT                 0x02
#define     SLCAN_PURGE_TX_CLEAR                 0x04
#define     SLCAN_PURGE_RX_CLEAR                 0x08

#pragma pack(push,1)

#ifdef __cplusplus
extern "C"{
#endif

typedef     PVOID   HSLCAN;

typedef    struct _SLCAN_CAPABILITIES{

    BYTE    bModes;
    BYTE    bTXModes;
```

```
    BYTE    bMaxEventLevel;
    BYTE    bController;
    BYTE    bPhysical;
    BYTE    bPhysicalNAGr;
    BYTE    bBitrates;
    BYTE    bAdvancedModes;
    DWORD   dwCanBaseClk;
    DWORD   dwTimeStampClk;
    WORD    wMaxBRP;
}SLCAN_CAPABILITIES,*PSLCAN_CAPABILITIES;

typedef struct   _SLCAN_BITRATE {
    WORD    BRP;
    BYTE    TSEG1;
    BYTE    TSEG2;
    BYTE    SJW;
    BYTE    SAM;
}SLCAN_BITRATE,*PSLCAN_BITRATE;

typedef    void (STDCALL* SLCAN_DEVICE_CALLBACK)(
    HSLCAN      hDevice,
    DWORD       dwIndex,
    DWORD       dwOperation,
    PVOID       pContext,
    DWORD       dwContextSize
);

typedef  VOID (STDCALL* SLCAN_DEVICELIST_CALLBACK)(
    HSLCAN      hDevice,
    DWORD       dwIndex,
    PVOID       pContext,
    DWORD       dwContextSize
);

typedef struct _SLCAN_MESSAGE{
    BYTE    Info;
    DWORD   ID;
    BYTE    DataCount;
    BYTE    Data[8];
}SLCAN_MESSAGE,*PSLCAN_MESSAGE;

typedef struct _SLCAN_TXMESSAGE{
    LONG            dwDelay;
    SLCAN_MESSAGE   Msg;
}SLCAN_TXMESSAGE,*PSLCAN_TXMESSAGE;

typedef struct _SLCAN_EVENT{
    BYTE    EventType;
    DWORD   TimeStampLo;
    union {
        SLCAN_MESSAGE Msg;
        DWORD TimeStamp[2];
        struct {
            BYTE BusMode;
            BYTE Dummy1;
            BYTE ErrCountRx;
            BYTE ErrCountTx;
            BYTE ErrType;
            BYTE ErrDir;
            BYTE ErrFrame;
            BYTE LostArbitration;
        };
    };
}SLCAN_EVENT,*PSLCAN_EVENT;
```

```
typedef struct _SLCAN_STATE{
    BYTE BusMode;
    BYTE Dummy1;
    BYTE ErrCountRX;
    BYTE ErrCountTX;
}SLCAN_STATE,*PSLCAN_STATE;

typedef union _SLCAN_TIMESTAMP{
    UINT64 Value;
    DWORD  dwValue[2];
    USHORT wValue[4];
    BYTE   bValue[8];
}SLCAN_TIMESTAMP,*PSLCAN_TIMESTAMP;



SLCANAPI BOOL STDCALL SlCan_Load(
   SLCAN_DEVICE_CALLBACK DeviceProc,
   SLCAN_DEVICELIST_CALLBACK DeviceListProc
);

SLCANAPI BOOL STDCALL SlCan_Free(
       BOOL    bDoCallBack
);

SLCANAPI BOOL STDCALL SlCan_Update();

SLCANAPI HSLCAN STDCALL SlCan_GetDevice(
       DWORD       dwIndex
);

SLCANAPI DWORD STDCALL SlCan_GetDeviceCount();


SLCANAPI HANDLE STDCALL SlCan_DeviceGetHandle(
       DWORD       dwIndex
);

SLCANAPI DWORD STDCALL SlCan_DeviceGetProperty(
   HSLCAN       hDevice,
   DWORD        dwIndex,
   PCHAR        pBuf,
   DWORD        dwSize
);

SLCANAPI DWORD STDCALL SlCan_DeviceGetPropertyW(
   HSLCAN       hDevice,
   DWORD        dwIndex,
   PWCHAR       pBuf,
   DWORD        dwSize
);

SLCANAPI HKEY STDCALL SlCan_DeviceGetRegKey(
   HSLCAN       hDevice,
   DWORD        dwIndex
);

SLCANAPI PVOID STDCALL SlCan_DeviceSetContext(
   HSLCAN       hDevice,
   PVOID        pBuf,
   DWORD        dwBufSize
);
```

```
SLCANAPI PVOID STDCALL SlCan_DeviceGetContext(
   HSLCAN       hDevice
);

SLCANAPI DWORD STDCALL SlCan_DeviceGetContextSize(
   HSLCAN       hDevice
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetAlias(
   HSLCAN       hDevice,
   PCHAR        pBuf
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetAliasW(
   HSLCAN       hDevice,
   PWCHAR        pBuf
);

SLCANAPI DWORD STDCALL SlCan_DeviceGetAlias(
   HSLCAN       hDevice,
   PCHAR        pBuf,
   DWORD        dwSize
);

SLCANAPI DWORD STDCALL SlCan_DeviceGetAliasW(
   HSLCAN       hDevice,
   PWCHAR        pBuf,
   DWORD        dwSize
);
SLCANAPI BOOL STDCALL SlCan_DeviceGetCapabilities(
  HSLCAN        hDevice,
  PSLCAN_CAPABILITIES   pCapabilities
);

SLCANAPI BOOL STDCALL SlCan_DeviceOpen(
  HSLCAN        hDevice
);

SLCANAPI BOOL STDCALL SlCan_DeviceClose(
  HSLCAN   hDevice
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetMode(
  HSLCAN        hDevice,
  DWORD         dwMode
);
SLCANAPI BOOL STDCALL SlCan_DeviceGetMode(
  HSLCAN        hDevice,
  PDWORD        pdwMode
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetState(
  HSLCAN        hDevice,
  PSLCAN_STATE  pState
);


SLCANAPI BOOL STDCALL SlCan_DeviceSetTXTimeOut(
  HSLCAN            hDevice,
  DWORD             dwMillisecond
);
SLCANAPI BOOL STDCALL SlCan_DeviceGetTXTimeOut(
  HSLCAN            hDevice,
  PDWORD            pdwMillisecond
```

```
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetBitRate(
  HSLCAN            hDevice,
  PSLCAN_BITRATE    pBitRate
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetBitRate(
  HSLCAN            hDevice,
  PSLCAN_BITRATE    pBitRate
);


SLCANAPI BOOL STDCALL SlCan_DeviceEnaRec(
  HSLCAN            hDevice,
  BYTE              bValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetLatency(
  HSLCAN            hDevice,
  BYTE              bValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetLatency(
  HSLCAN            hDevice,
  PBYTE             pbValue
);

SLCANAPI BOOL STDCALL SlCan_DevicePurge(
  HSLCAN            hDevice,
  BYTE              bValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetEventLevel(
  HSLCAN            hDevice,
  BYTE              bValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetEventLevel(
  HSLCAN            hDevice,
  PBYTE             pbValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetStartTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetStartTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetTimeStamp(
  HSLCAN            hDevice,
  PSLCAN_TIMESTAMP  pValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetTimeStampPeriod(
  HSLCAN            hDevice,
  LONG              lValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetTimeStampPeriod(
```

```
  HSLCAN            hDevice,
  PLONG             plValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceSetExMode(
  HSLCAN            hDevice,
  BYTE              bValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceGetExMode(
  HSLCAN            hDevice,
  PBYTE             pbValue
);

SLCANAPI BOOL STDCALL SlCan_DeviceWriteMessages(
  HSLCAN            hDevice,
  PSLCAN_MESSAGE    pMsg,
  DWORD             dwCount,
  PBYTE             pbStatus
);

SLCANAPI BOOL STDCALL SlCan_DeviceWriteMessagesEx(
  HSLCAN            hDevice,
  PSLCAN_TXMESSAGE  pMsg,
  DWORD             dwCount,
  PBYTE             pbStatus,
  PDWORD            pdwCount
);

SLCANAPI BOOL STDCALL SlCan_DeviceReadMessages(
  HSLCAN            hDevice,
  DWORD             dwTimeOut,
  PSLCAN_MESSAGE    pMsg,
  DWORD             dwCount,
  PDWORD            pdwCount
);

SLCANAPI BOOL STDCALL SlCan_DeviceReadEvents(
  HSLCAN            hDevice,
  DWORD             dwTimeOut,
  PSLCAN_EVENT      pEvent,
  DWORD             dwCount,
  PDWORD            pdwCount
);

#ifdef      __cplusplus
}
#endif

#pragma     pack(pop)

#endif      //__SLCAN_H
```

slcan.pas

```
unit slcan;

interface

uses Windows;

const
   SLCAN_PROPERTY_INDEX_LINKNAME       = 0;
   SLCAN_PROPERTY_INDEX_INSTANCEID     = 1;
```

```
SLCAN_PROPERTY_INDEX_DEVICEDESC     = 2;
SLCAN_PROPERTY_INDEX_FRIENDLYNAME   = 3;
SLCAN_PROPERTY_INDEX_PHOBJECTNAME   = 4;
SLCAN_PROPERTY_INDEX_MFG            = 5;
SLCAN_PROPERTY_INDEX_LOCATIONINFO   = 6;
SLCAN_PROPERTY_INDEX_ENUMERATOR     = 7;
SLCAN_PROPERTY_INDEX_CLASS          = 8;
SLCAN_PROPERTY_INDEX_CLASSGUID      = 9;
SLCAN_PROPERTY_INDEX_SERVICE        = 10;
SLCAN_PROPERTY_INDEX_DRIVER         = 11;
SLCAN_PROPERTY_INDEX_PORTNAME       = 12;
SLCAN_PROPERTY_INDEX_PRODUCT        = 13;
SLCAN_PROPERTY_INDEX_MANUFACTURER   = 14;
SLCAN_PROPERTY_INDEX_CONFIGURATION  = 15;
SLCAN_PROPERTY_INDEX_INTERFACE      = 16;
SLCAN_PROPERTY_INDEX_SERIAL         = 17;
SLCAN_PROPERTY_INDEX_ALIAS          = 18;
SLCAN_PROPERTY_INDEX_CHANNELLINK    = 19;
SLCAN_PROPERTY_INDEX_SERIALID       = 20;

SLCAN_MODE_CONFIG       = $00;
SLCAN_MODE_NORMAL       = $01;
SLCAN_MODE_LISTENONLY   = $02;
SLCAN_MODE_LOOPBACK     = $03;
SLCAN_MODE_SLEEP        = $04;

SLCAN_BR_CIA_1000K      = $8000;
SLCAN_BR_CIA_800K       = $8001;
SLCAN_BR_CIA_500K       = $8002;
SLCAN_BR_CIA_250K       = $8003;
SLCAN_BR_CIA_125K       = $8004;
SLCAN_BR_CIA_50K        = $8005;
SLCAN_BR_CIA_20K        = $8006;
SLCAN_BR_CIA_10K        = $8007;
SLCAN_BR_400K           = $8008;
SLCAN_BR_200K           = $8009;
SLCAN_BR_100K           = $800A;
SLCAN_BR_83333          = $800B;
SLCAN_BR_33333          = $800C;
SLCAN_BR_25K            = $800D;
SLCAN_BR_5K             = $800E;
SLCAN_BR_30K            = $800F;
SLCAN_BR_300K           = $8010;
SLCAN_BR_LASTINDEX = SLCAN_BR_33333;

SLCAN_CAP_MODE_NORMAL         = $01;
SLCAN_CAP_MODE_LISTEN_ONLY    = $02;
SLCAN_CAP_MODE_LOOP_BACK      = $04;
SLCAN_CAP_MODE_SLEEP          = $08;

SLCAN_CAP_TXMODE_ONE_SHOT     = $01;
SLCAN_CAP_TXMODE_TIME_STAMP   = $02;

SLCAN_CAP_CONTR_EXTERNAL = $00;
SLCAN_CAP_CONTR_MCP2515  = $01;
SLCAN_CAP_CONTR_SJA1000  = $02;

SLCAN_CAP_CONTR_INTERNAL = $80;
SLCAN_CAP_CONTR_LPC      = $81;
SLCAN_CAP_CONTR_STM32    = $82;
SLCAN_CAP_CONTR_STM8     = $83;
SLCAN_CAP_CONTR_PIC      = $84;
SLCAN_CAP_CONTR_PIC_ECAN = $85;
```

```
SLCAN_CAP_PHYS_HS    = $01;
SLCAN_CAP_PHYS_LS    = $02;
SLCAN_CAP_PHYS_SW    = $04;
SLCAN_CAP_PHYS_J1708 = $08;
SLCAN_CAP_PHYS_LIN   = $10;
SLCAN_CAP_PHYS_KLINE = $20;

SLCAN_CAP_PHYS_NAGR = $01;

SLCAN_CAP_BITRATE_INDEX  = $01;
SLCAN_CAP_BITRATE_CUSTOM = $02;
SLCAN_CAP_BITRATE_AUTO   = $04;

SLCAN_EVT_LEVEL_RX_MSG     = 0;
SLCAN_EVT_LEVEL_TIME_STAMP = 1;
SLCAN_EVT_LEVEL_TX_MSG     = 2;
SLCAN_EVT_LEVEL_BUS_STATE  = 3;
SLCAN_EVT_LEVEL_COUNTS     = 4;
SLCAN_EVT_LEVEL_ERRORS     = 5;

SLCAN_EVT_TYPE_RX                = $0;
SLCAN_EVT_TYPE_START_TX          = $1;
SLCAN_EVT_TYPE_END_TX            = $2;
SLCAN_EVT_TYPE_ABORT_TX          = $3;
SLCAN_EVT_TYPE_BUS_STATE         = $4;
SLCAN_EVT_TYPE_ERROR_COUNTS      = $5;
SLCAN_EVT_TYPE_BUS_ERROR         = $6;
SLCAN_EVT_TYPE_ARBITRATION = $7;
SLCAN_EVT_TYPE_STAMP_INC         = $F;

SLCAN_BUS_STATE_ERROR_ACTIVE      = $00;
SLCAN_BUS_STATE_ERROR_ACTIVE_WARN = $01;
SLCAN_BUS_STATE_ERROR_PASSIVE     = $02;
SLCAN_BUS_STATE_BUSOFF            = $03;

SLCAN_MES_INFO_EXT     = $01;
SLCAN_MES_INFO_RTR     = $02;
SLCAN_MES_INFO_ONESHOT = $04;

SLCAN_DEVOP_CREATE        = $00000000;
SLCAN_DEVOP_CREATEHANDLE  = $00000001;
SLCAN_DEVOP_OPEN          = $00000002;
SLCAN_DEVOP_CLOSE         = $00000003;
SLCAN_DEVOP_DESTROYHANDLE = $00000004;
SLCAN_DEVOP_DESTROY       = $00000005;
SLCAN_DEVOP_INFO          = $00000006;
SLCAN_DEVOP_USER          = $00000007;

SLCAN_INVALID_HANDLE_ERROR          = $E0001001;
SLCAN_DEVICE_INVALID_HANDLE_ERROR   = $E0001120;
SLCAN_HANDLE_INIT_ERROR             = $E0001017;
SLCAN_DEVICE_NOTOPEN_ERROR          = $E0001121;

SLCAN_EVT_ERR_TYPE_BIT              = $00;
SLCAN_EVT_ERR_TYPE_FORM             = $01;
SLCAN_EVT_ERR_TYPE_STUFF            = $02;
SLCAN_EVT_ERR_TYPE_OTHER            = $03;

SLCAN_EVT_ERR_DIR_TX                = $00;
SLCAN_EVT_ERR_DIR_RX                = $01;

SLCAN_EVT_ERR_FRAME_SOF             = $03;
SLCAN_EVT_ERR_FRAME_ID28_ID21       = $02;
SLCAN_EVT_ERR_FRAME_ID20_ID18       = $06;
```

```
   SLCAN_EVT_ERR_FRAME_SRTR            = $04;
   SLCAN_EVT_ERR_FRAME_IDE             = $05;
   SLCAN_EVT_ERR_FRAME_ID17_ID13       = $07;
   SLCAN_EVT_ERR_FRAME_ID12_ID5        = $0F;
   SLCAN_EVT_ERR_FRAME_ID4_ID0         = $0E;
   SLCAN_EVT_ERR_FRAME_RTR             = $0C;
   SLCAN_EVT_ERR_FRAME_RSRV0           = $0D;
   SLCAN_EVT_ERR_FRAME_RSRV1           = $09;
   SLCAN_EVT_ERR_FRAME_DLC             = $0B;
   SLCAN_EVT_ERR_FRAME_DATA            = $0A;
   SLCAN_EVT_ERR_FRAME_CRC_SEQ         = $08;
   SLCAN_EVT_ERR_FRAME_CRC_DEL         = $18;
   SLCAN_EVT_ERR_FRAME_ACK_SLOT        = $19;
   SLCAN_EVT_ERR_FRAME_ACK_DEL         = $1B;
   SLCAN_EVT_ERR_FRAME_EOF             = $1A;
   SLCAN_EVT_ERR_FRAME_INTER           = $12;
   SLCAN_EVT_ERR_FRAME_AER_FLAG        = $11;
   SLCAN_EVT_ERR_FRAME_PER_FLAG        = $16;
   SLCAN_EVT_ERR_FRAME_TDB             = $13;
   SLCAN_EVT_ERR_FRAME_ERR_DEL         = $17;
   SLCAN_EVT_ERR_FRAME_OVER_FLAG       = $1C;

   SLCAN_TX_STATUS_OK                  =$00;
   SLCAN_TX_STATUS_TIMEOUT             =$01;
   SLCAN_TX_STATUS_BUSOFF              =$02;
   SLCAN_TX_STATUS_ABORT               =$03;
   SLCAN_TX_STATUS_NOT_ENA             =$04;
   SLCAN_TX_STATUS_ERROR_ONE_SHOT      =$05;
   SLCAN_TX_STATUS_INVALID_MODE        =$06;
   SLCAN_TX_STATUS_UNKNOWN             =$0F;

   SLCAN_PURGE_TX_ABORT                =$01;
   SLCAN_PURGE_RX_ABORT                =$02;
   SLCAN_PURGE_TX_CLEAR                =$04;
   SLCAN_PURGE_RX_CLEAR                =$08;

 type
{$IFDEF CompilerVersion}
{$IF CompilerVersion >= 16 }
TSlCanDevice = NativeUInt;
{$ELSE}
TSlCanDevice = longword;
{$ENDIF}
{$ELSE}
TSlCanDevice = longword;
{$ENDIF}

PSlCanCapabilities = ^TSlCanCapabilities;
TSlCanCapabilities = packed record
   bModes:byte;
   bTXModes:byte;
   bMaxEventLevel:byte;
   bController:byte;
   bPhysical:byte;
   bPhysicalNAGr:byte;
   bBitrates:byte;
   bAdvancedModes:byte;
   dwCanBaseClk:longword;
   dwTimeStampClk:longword;
   wMaxBrp:word;
end;

PSlCanState = ^TSlCanState;
TSlCanState = packed record
```

```
   BusState:byte;
...Dummy1:byte;
   ErrCountRX:byte;
   ErrCountTX:byte;
end;

PSlCanMessage = ^TSlCanMessage;
TSlCanMessage = packed record
   Info:byte;
   ID:longword;
   DataCount:byte;
   Data: array [0 .. 7] of byte;
end;

PSlCanTxMessage = ^TSlCanTxMessage;
TSlCanTxMessage = packed record
   dwDelay:longint;
   Msg:TSlCanMessage;
end;


PSlCanEvent = ^TSlCanEvent;
TSlCanEvent = packed record
   EventType:byte;
   TimeStampLo:longword;
   case integer of
   0:(Msg:TSlCanMessage);
   1:(TimeStamp: array [0 .. 1] of longword);
   2:(BusMode:byte;
      bDummy1:byte;
      ErrCountRX:byte;
      ErrCountTX:byte;
      ErrType:byte;
      ErrDir:byte;
      ErrFrame:byte;
      LostArbitration:byte);
end;

PSlCanBitRate = ^TSlCanBitRate;
TSlCanBitRate = packed record
   BRP:word;
   TSEG1:byte;
   TSEG2:byte;
   SJW:byte;
   SAM:byte;
end;

PSlCanTimeStamp = ^TSlCanTimeStamp;
TSlCanTimeStamp = packed record
   case integer of
   0:(Value:int64);
   1:(dwValue: array [0 .. 1] of longword);
   2:(wValue: array [0 .. 3] of word);
   3:(bValue: array [0 .. 7] of byte);
end;

TSlCan_DeviceCallBack = procedure(
   hDevice:       TSlCanDevice;
   dwIndex:       longword;
   Operation:     longword;
   pContext:      pointer;
   dwContextSize: longword
   );stdcall;
```

```
TSlCan_DeviceListCallBack = procedure(
   hDevice:       TSlCanDevice;
   dwIndex:       longword;
   pContext:      pointer;
   dwContextSize: longword
   );stdcall;

function SlCan_Load(
   DeviceProc:       TSlCan_DeviceCallBack;
   DeviceListProc:   TSlCan_DeviceListCallBack
   ):longbool;stdcall;

function SlCan_Free(
   bDoCallBack:longbool
   ):longbool;stdcall;

function SlCan_Update:longbool;stdcall;

function SlCan_GetDeviceCount:longword;stdcall;

function SlCan_GetDevice(
   dwIndex:longword
   ):TSlCanDevice;stdcall;

function SlCan_DeviceGetHandle(
   hDevice:TSlCanDevice
   ):THandle;stdcall;

function SlCan_DeviceGetProperty(
   hDevice: TSlCanDevice;
   dwIndex: longword;
   pBuf:    pAnsiChar;
   dwSize:  longword
   ):longword;stdcall;

function SlCan_DeviceGetPropertyW(
   hDevice: TSlCanDevice;
   dwIndex: longword;
   pBuf:    PWideChar;
   dwSize:  longword
   ):longword;stdcall;

function SlCan_DeviceGetRegKey(
   hDevice: TSlCanDevice;
   dwIndex: longword
   ):HKEY;stdcall;

function SlCan_DeviceSetContext(
   hDevice:TSlCanDevice;
   pBuf:     pointer;
   dwBufSize:longword
   ):pointer;stdcall;

function SlCan_DeviceGetContext(hDevice:TSlCanDevice):pointer;stdcall;

function SlCan_DeviceGetContextSize(
   hDevice: TSlCanDevice
   ):longword;stdcall;

function SlCan_DeviceSetAlias(
   hDevice: TSlCanDevice;
   pBuf:    PAnsiChar
   ):longbool;stdcall;
```

```
function SlCan_DeviceGetAlias(
   hDevice:TSlCanDevice;
   pBuf:    PAnsiChar;
   dwSize:  longword
   ):longword;stdcall;

function SlCan_DeviceSetAliasW(
   hDevice: TSlCanDevice;
   pBuf:    PWideChar
   ):longbool;stdcall;

function SlCan_DeviceGetAliasW(
   hDevice: TSlCanDevice;
   pBuf:    PWideChar;
   dwSize:  longword
   ):longword;stdcall;

function SlCan_DeviceGetCapabilities(
   hDevice: TSlCanDevice;
   pCap:    PSlCanCapabilities
   ):longbool;stdcall;

function SlCan_DeviceOpen(
   hDevice:TSlCanDevice
   ):longbool;stdcall;

function SlCan_DeviceClose(
   hDevice:TSlCanDevice
   ):longbool;stdcall;

function SlCan_DeviceSetMode(
   hDevice: TSlCanDevice;
   dwMode:  longword
   ):longbool;stdcall;

function SlCan_DeviceGetMode(
   hDevice: TSlCanDevice;
   pdwMode: Plongword
   ):longbool;stdcall;

function SlCan_DeviceGetState(
   hDevice:TSlCanDevice;
   pState: PSlCanState
   ):longbool;stdcall;

function SlCan_DeviceSetTXTimeOut(
   hDevice: TSlCanDevice;
   dwValue: longword):longbool;stdcall;

function SlCan_DeviceGetTXTimeOut(
   hDevice: TSlCanDevice;
   pdwValue:Plongword
   ):longbool;stdcall;

function SlCan_DeviceGetBitRate(
   hDevice: TSlCanDevice;
   pBitRate:PSlCanBitRate
   ):longbool;stdcall;

function SlCan_DeviceSetBitRate(
   hDevice: TSlCanDevice;
   pBitRate:PSlCanBitRate
   ):longbool;stdcall;
```

```
function SlCan_DeviceEnaRec(
   hDevice: TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;

function SlCan_DeviceSetLatency(
   hDevice: TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;

function SlCan_DeviceGetLatency(
   hDevice: TSlCanDevice;
   pbValue: pbyte
   ):longbool;stdcall;

function SlCan_DevicePurge(
   hDevice:TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;

function SlCan_DeviceSetEventLevel(
   hDevice:TSlCanDevice;
   bValue:  byte
   ):longbool;stdcall;

function SlCan_DeviceGetEventLevel(
   hDevice:TSlCanDevice;
   pbValue: pbyte
   ):longbool;stdcall;

function SlCan_DeviceSetStartTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;

function SlCan_DeviceGetStartTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;

function SlCan_DeviceGetTimeStamp(
   hDevice: TSlCanDevice;
   pValue:  PSlCanTimeStamp
   ):longbool;stdcall;

function SlCan_DeviceSetTimeStampPeriod(
   hDevice: TSlCanDevice;
   lValue:  longwint
   ):longbool;stdcall;

function SlCan_DeviceGetTimeStampPeriod(
   hDevice: TSlCanDevice;
   plValue:Plongint
   ):longbool;stdcall;

function SlCan_DeviceSetExMode(
   hDevice: TSlCanDevice;
   bMode:   byte
   ):longbool;stdcall;

function SlCan_DeviceGetExMode(
   hDevice:TSlCanDevice;
   pbMode:  pbyte
   ):longbool;stdcall;
```

```
function SlCan_DeviceWriteMessages(
   hDevice: TSlCanDevice;
   pMsg:    PSlCanMessage;
   Count:   longword;
   pStatus: pbyte
   ):longbool;stdcall;

function SlCan_DeviceWriteMessagesEx(
   hDevice: TSlCanDevice;
   pMsg:    PSlCanTxMessage;
   Count:   longword;
   pStatus: pbyte;
   pdwCount:Plongword
   ):longbool;stdcall;

function SlCan_DeviceReadMessages(
   hDevice:  TSlCanDevice;
   dwTimeOut:longword;
   pMsg:     PSlCanMessage;
   Count:    longword;
   pCount:   Plongword
   ):longbool;stdcall;

function SlCan_DeviceReadEvents(
   hDevice:  TSlCanDevice;
   dwTimeOut:longword;
   pEvent:   PSlCanEvent;
   Count:    longword;
   pCount:   Plongword
   ):longbool;stdcall;

implementation

const
   Dll = 'slcan.dll';

function SlCan_Load;stdcall;external Dll;
function SlCan_Free;stdcall;external Dll;
function SlCan_Update;stdcall;external Dll;
function SlCan_GetDeviceCount;stdcall;external Dll;
function SlCan_GetDevice;stdcall;external Dll;

function SlCan_DeviceGetHandle;stdcall;external Dll;
function SlCan_DeviceOpen;stdcall;external Dll;
function SlCan_DeviceClose;stdcall;external Dll;

function SlCan_DeviceGetProperty;stdcall;external Dll;
function SlCan_DeviceGetPropertyW;stdcall;external Dll;
function SlCan_DeviceGetRegKey;stdcall;external Dll;
function SlCan_DeviceSetAlias;stdcall;external Dll;
function SlCan_DeviceGetAlias;stdcall;external Dll;
function SlCan_DeviceSetAliasW;stdcall;external Dll;
function SlCan_DeviceGetAliasW;stdcall;external Dll;
function SlCan_DeviceGetCapabilities;stdcall;external Dll;

function SlCan_DeviceSetContext;stdcall;external Dll;
function SlCan_DeviceGetContext;stdcall;external Dll;
function SlCan_DeviceGetContextSize;stdcall;external Dll;

function SlCan_DeviceSetMode;stdcall;external Dll;
function SlCan_DeviceGetMode;stdcall;external Dll;
function SlCan_DeviceGetState;stdcall;external Dll;
function SlCan_DeviceSetTXTimeOut;stdcall;external Dll;
```

```
function SlCan_DeviceGetTXTimeOut;stdcall;external Dll;

function SlCan_DeviceGetBitRate;stdcall;external Dll;
function SlCan_DeviceSetBitRate;stdcall;external Dll;
function SlCan_DeviceEnaRec;stdcall;external Dll;
function SlCan_DeviceSetLatency;stdcall;external Dll;
function SlCan_DeviceGetLatency;stdcall;external Dll;
function SlCan_DevicePurge;stdcall;external Dll;
function SlCan_DeviceSetEventLevel;stdcall;external Dll;
function SlCan_DeviceGetEventLevel;stdcall;external Dll;
function SlCan_DeviceSetStartTimeStamp;stdcall;external Dll;
function SlCan_DeviceGetStartTimeStamp;stdcall;external Dll;
function SlCan_DeviceGetTimeStamp;stdcall;external Dll;
function SlCan_DeviceSetTimeStampPeriod;stdcall;external Dll;
function SlCan_DeviceGetTimeStampPeriod;stdcall;external Dll;
function SlCan_DeviceSetExMode;stdcall;external Dll;
function SlCan_DeviceGetExMode;stdcall;external Dll;

function SlCan_DeviceWriteMessages;stdcall;external Dll;
function SlCan_DeviceWriteMessagesEx;stdcall;external Dll;
function SlCan_DeviceReadMessages;stdcall;external Dll;
function SlCan_DeviceReadEvents;stdcall;external Dll;

end.
```

slcanclass.cs

```
using System;
using System.Text;
using System.Runtime.InteropServices;

public static class SlCan
{

    public const int PROPERTY_INDEX_LINKNAME = 0;
    public const int PROPERTY_INDEX_INSTANCEID = 1;
    public const int PROPERTY_INDEX_DEVICEDESC = 2;
    public const int PROPERTY_INDEX_FRIENDLYNAME = 3;
    public const int PROPERTY_INDEX_PHOBJECTNAME = 4;
    public const int PROPERTY_INDEX_MFG = 5;
    public const int PROPERTY_INDEX_LOCATIONINFO = 6;
    public const int PROPERTY_INDEX_ENUMERATOR = 7;
    public const int PROPERTY_INDEX_CLASS = 8;
    public const int PROPERTY_INDEX_CLASSGUID = 9;
    public const int PROPERTY_INDEX_SERVICE = 10;
    public const int PROPERTY_INDEX_DRIVER = 11;
    public const int PROPERTY_INDEX_PORTNAME = 12;
    public const int PROPERTY_INDEX_PRODUCT = 13;
    public const int PROPERTY_INDEX_MANUFACTURER = 14;
    public const int PROPERTY_INDEX_CONFIGURATION = 15;
    public const int PROPERTY_INDEX_INTERFACE = 16;
    public const int PROPERTY_INDEX_SERIAL = 17;
    public const int PROPERTY_INDEX_ALIAS = 18;
    public const int PROPERTY_INDEX_CHANNELLINK = 19;
    public const int PROPERTY_INDEX_SERIALID = 20;

    public const int MODE_CONFIG = 0x00;
    public const int MODE_NORMAL = 0x01;
    public const int MODE_LISTENONLY = 0x02;
    public const int MODE_LOOPBACK = 0x03;
    public const int MODE_SLEEP = 0x04;

    public const ushort BR_CIA_1000K = 0x8000;
    public const ushort BR_CIA_800K = 0x8001;
    public const ushort BR_CIA_500K = 0x8002;
```

```
public const ushort BR_CIA_250K = 0x8003;
public const ushort BR_CIA_125K = 0x8004;
public const ushort BR_CIA_50K = 0x8005;
public const ushort BR_CIA_20K = 0x8006;
public const ushort BR_CIA_10K = 0x8007;
public const ushort BR_400K = 0x8008;
public const ushort BR_200K = 0x8009;
public const ushort BR_100K = 0x800A;
public const ushort BR_83333 = 0x800B;
public const ushort BR_33333 = 0x800C;
public const ushort BR_25K = 0x800D;
public const ushort BR_5K = 0x800E;
public const ushort BR_30K = 0x800F;
public const ushort BR_300K = 0x8010;
public const ushort BR_LASTINDEX = BR_300K;

public const byte CAP_MODE_NORMAL       = 0x01;
public const byte CAP_MODE_LISTEN_ONLY =  0x02;
public const byte CAP_MODE_LOOP_BACK    = 0x04;
public const byte CAP_MODE_SLEEP        = 0x08;

public const byte CAP_TXMODE_ONE_SHOT   = 0x01;
public const byte CAP_TXMODE_TIMESTAMP  = 0x02;


public const byte CAP_CONTR_EXTERNAL    = 0x00;
public const byte CAP_CONTR_MCP2515     = 0x01;
public const byte CAP_CONTR_SJA1000     = 0x02;

public const byte CAP_CONTR_INTERNAL    = 0x80;
public const byte CAP_CONTR_LPC         = 0x81;
public const byte CAP_CONTR_STM32       = 0x82;
public const byte CAP_CONTR_STM8        = 0x83;
public const byte CAP_CONTR_PIC         = 0x84;
public const byte CAP_CONTR_PIC_ECAN    = 0x85;

public const byte CAP_PHYS_HS           = 0x01;
public const byte CAP_PHYS_LS           = 0x02;
public const byte CAP_PHYS_SW           = 0x04;
public const byte CAP_PHYS_J1708        = 0x08;
public const byte CAP_PHYS_LIN          = 0x10;
public const byte CAP_PHYS_KLINE        = 0x20;

public const byte CAP_PHYS_LOAD         = 0x01;

public const byte CAP_BITRATE_INDEX     = 0x01;
public const byte CAP_BITRATE_CUSTOM    = 0x02;
public const byte CAP_BITRATE_AUTOMATIC = 0x04;


public const byte EVT_LEVEL_RX_MSG      = 0;
public const byte EVT_LEVEL_TIME_STAMP  = 1;
public const byte EVT_LEVEL_TX_MSG      = 2;
public const byte EVT_LEVEL_BUS_STATE   = 3;
public const byte EVT_LEVEL_COUNTS      = 4;
public const byte EVT_LEVEL_ERRORS      = 5;

public const byte EVT_TYPE_RX           = 0x0;
public const byte EVT_TYPE_START_TX     = 0x1;
public const byte EVT_TYPE_END_TX       = 0x2;
public const byte EVT_TYPE_ABORT_TX     = 0x3;
public const byte EVT_TYPE_BUS_STATE    = 0x4;
public const byte EVT_TYPE_ERROR_COUNTS = 0x5;
public const byte EVT_TYPE_BUS_ERROR    = 0x6;
```

```
public const byte EVT_TYPE_ARBITRATION   =0x7;
public const byte EVT_STAMP_INC         = 0xF;

public const byte BUS_STATE_ERROR_ACTIVE= 0x00;
public const byte BUS_STATE_ERROR_ACTIVE_WARN  = 0x01;
public const byte BUS_STATE_ERROR_PASSIVE     =  0x02;
public const byte BUS_STATE_BUSOFF      = 0x03;

public const byte MES_INFO_EXT       = 0x01;
public const byte MES_INFO_RTR       = 0x02;
public const byte MES_INFO_ONESHOT   = 0x04;

public const int DEVOP_CREATE = 0x00000000;
public const int DEVOP_CREATEHANDLE = 0x00000001;
public const int DEVOP_OPEN = 0x00000002;
public const int DEVOP_CLOSE = 0x00000003;
public const int DEVOP_DESTROYHANDLE = 0x00000004;
public const int DEVOP_DESTROY = 0x00000005;

public const uint INVALID_HANDLE_ERROR           = 0xE0001001;
public const uint DEVICE_INVALID_HANDLE_ERROR    = 0xE0001120;
public const uint HANDLE_INIT_ERROR              = 0xE0001017;
public const uint DEVICE_NOTOPEN_ERROR           = 0xE0001121;

public const byte EVT_ERR_TYPE_BIT              = 0x00;
public const byte EVT_ERR_TYPE_FORM             = 0x01;
public const byte EVT_ERR_TYPE_STUFF            = 0x02;
public const byte EVT_ERR_TYPE_OTHER            = 0x03;

public const byte EVT_ERR_DIR_TX                = 0x00;
public const byte EVT_ERR_DIR_RX                = 0x01;

public const byte EVT_ERR_FRAME_SOF             = 0x03;
public const byte EVT_ERR_FRAME_ID28_ID21       = 0x02;
public const byte EVT_ERR_FRAME_ID20_ID18       = 0x06;
public const byte EVT_ERR_FRAME_SRTR            = 0x04;
public const byte EVT_ERR_FRAME_IDE             = 0x05;
public const byte EVT_ERR_FRAME_ID17_ID13       = 0x07;
public const byte EVT_ERR_FRAME_ID12_ID5        = 0x0F;
public const byte EVT_ERR_FRAME_ID4_ID0         = 0x0E;
public const byte EVT_ERR_FRAME_RTR             = 0x0C;
public const byte EVT_ERR_FRAME_RSRV0           = 0x0D;
public const byte EVT_ERR_FRAME_RSRV1           = 0x09;
public const byte EVT_ERR_FRAME_DLC             = 0x0B;
public const byte EVT_ERR_FRAME_DATA            = 0x0A;
public const byte EVT_ERR_FRAME_CRC_SEQ         = 0x08;
public const byte EVT_ERR_FRAME_CRC_DEL         = 0x18;
public const byte EVT_ERR_FRAME_ACK_SLOT        = 0x19;
public const byte EVT_ERR_FRAME_ACK_DEL         = 0x1B;
public const byte EVT_ERR_FRAME_EOF             = 0x1A;
public const byte EVT_ERR_FRAME_INTER           = 0x12;
public const byte EVT_ERR_FRAME_AER_FLAG        = 0x11;
public const byte EVT_ERR_FRAME_PER_FLAG        = 0x16;
public const byte EVT_ERR_FRAME_TDB             = 0x13;
public const byte EVT_ERR_FRAME_ERR_DEL         = 0x17;
public const byte EVT_ERR_FRAME_OVER_FLAG       = 0x1C;

public const byte TX_STATUS_OK                  = 0x00;
public const byte TX_STATUS_TIMEOUT             = 0x01;
public const byte TX_STATUS_BUSOFF              = 0x02;
public const byte TX_STATUS_ABORT               = 0x03;
public const byte TX_STATUS_NOT_ENA             = 0x04;
public const byte TX_STATUS_ERROR_ONE_SHOT      = 0x05;
public const byte TX_STATUS_INVALID_MODE        = 0x06;
```

```
public const byte TX_STATUS_UNKNOWN             = 0x0F;

public const byte PURGE_TX_ABORT                = 0x01;
public const byte PURGE_RX_ABORT                = 0x02;
public const byte PURGE_TX_CLEAR                = 0x04;
public const byte PURGE_RX_CLEAR                = 0x08;

public const uint INVALID_HANDLE_VALUE = 0xFFFFFFFF;

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Capability
{
    public byte bModes;
    public byte bTXModes;
    public byte bMaxEventLevel;
    public byte bController;
    public byte bPhysical;
    public byte bPhysicalNAGr;
    public byte bBitrates;
    public byte bAdvancedModes;
    public uint dwCanBaseClk;
    public uint dwTimeStampClk;
    public ushort wMaxBRP;
}
[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct BitRate
{
    public ushort BRP;
    public byte TSEG1;
    public byte TSEG2;
    public byte SJW;
    public byte SAM;
}

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct Message
{
    public byte Info;
    public UInt32 ID;
    public byte DataCount;
    public byte Data0;
    public byte Data1;
    public byte Data2;
    public byte Data3;
    public byte Data4;
    public byte Data5;
    public byte Data6;
    public byte Data7;
}

[StructLayout(LayoutKind.Sequential, Pack = 1)]
public struct TxMessage
{
    public int  Delay;
    public byte Info;
    public UInt32 ID;
    public byte DataCount;
    public byte Data0;
    public byte Data1;
    public byte Data2;
    public byte Data3;
    public byte Data4;
    public byte Data5;
    public byte Data6;
```

```
        public byte Data7;
    }
    [StructLayout(LayoutKind.Explicit, Size=19, Pack = 1)]
    public struct Event
    {
        [FieldOffset(0)]
        public byte EventType;
        [FieldOffset(1)] public uint TimeStampLo;
        [FieldOffset(5)]
        public byte Info;
        [FieldOffset(6)]
        public UInt32 ID;
        [FieldOffset(10)]
        public byte DataCount;
        [FieldOffset(11)]
        public byte Data0;
        [FieldOffset(12)]
        public byte Data1;
        [FieldOffset(13)]
        public byte Data2;
        [FieldOffset(14)]
        public byte Data3;
        [FieldOffset(15)]
        public byte Data4;
        [FieldOffset(16)]
        public byte Data5;
        [FieldOffset(17)]
        public byte Data6;
        [FieldOffset(18)]
        public byte Data7;

        [FieldOffset(5)]
        public UInt64 TimeStamp;

        [FieldOffset(5)]
        public byte BusMode;
        [FieldOffset(6)]
        public byte Dummy1;
        [FieldOffset(7)]
        public byte ErrCountRx;
        [FieldOffset(8)]
        public byte ErrCountTx;
        [FieldOffset(9)]
        public byte ErrType;
        [FieldOffset(10)]
        public byte ErrDir;
        [FieldOffset(11)]
        public byte ErrFrame;
        [FieldOffset(12)]
        public byte LostArbitration;
    }

    [StructLayout(LayoutKind.Sequential, Pack = 1)]
    public struct State
    {
        public byte BusMode;
        public byte Dummy1;
        public byte ErrCountRX;
        public byte ErrCountTX;

    }

    public delegate void DeviceListCallBack(IntPtr hDevice, uint dwIndex, IntPtr
pContext, uint dwContextSize);
```

```
    public delegate void DeviceCallBack(IntPtr hDevice, uint dwIndex, uint
dwOperation, IntPtr pContext, uint dwContextSize);

    [DllImport("slcan.dll", EntryPoint = "SlCan_Load", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool Load(DeviceCallBack DeviceProc, DeviceListCallBack
DeviceListProc);

    [DllImport("slcan.dll", EntryPoint = "SlCan_Free", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool Free(bool bDoCallBack);

    [DllImport("slcan.dll", EntryPoint = "SlCan_Update", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool Update();

    [DllImport("slcan.dll", EntryPoint = "SlCan_GetDeviceCount", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern uint GetDeviceCount();

    [DllImport("slcan.dll", EntryPoint = "SlCan_GetDevice", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern IntPtr GetDevice(uint dwIndex);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetProperty", CallingConvention
= CallingConvention.StdCall, SetLastError = true)]
    public static extern uint DeviceGetProperty(IntPtr hDevice, uint dwIndex,
StringBuilder pInfo, int dwInfoSize);

    [DllImport("slcan.dll", EntryPoint = "Slcan_DeviceSetContext", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern IntPtr DeviceSetContext(IntPtr hDevice, IntPtr pBuf, uint
dwBufSize);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetContext", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern IntPtr DeviceGetContext(IntPtr hDevice);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetContextSize",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern uint DeviceGetContextSize(IntPtr hDevice);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetCapabilities",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetCapabilities(IntPtr hDevice, out Capability
Cap);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceOpen", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceOpen(IntPtr hDevice);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceClose", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceClose(IntPtr hDevice);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetMode", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetMode(IntPtr hDevice, uint dwMode);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetMode", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetMode(IntPtr hDevice, out uint dwMode);
```

```
    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetState", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetState(IntPtr hDevice, out uint dwState);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetTXTimeOut", CallingConvention
= CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetTXTimeOut(IntPtr hDevice, uint dwMilliseconds);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetTXTimeOut", CallingConvention
= CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetTXTimeOut(IntPtr hDevice, out uint
dwMilliseconds);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetBitRate", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetBitRate(IntPtr hDevice, out BitRate bitrate);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetBitRate", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetBitRate(IntPtr hDevice, ref BitRate bitrate);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceEnaRec", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceEnaRec(IntPtr hDevice, byte bValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetLatency", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetLatency(IntPtr hDevice, byte bValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetLatency", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetLatency(IntPtr hDevice, out byte pbValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DevicePurge", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DevicePurge(IntPtr hDevice, byte bValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetEventLevel",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetEventLevel(IntPtr hDevice, byte bValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetEventLevel",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetEventLevel(IntPtr hDevice, out byte pbValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetStartTimeStamp",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetStartTimeStamp(IntPtr hDevice, ref UInt64
Value);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetStartTimeStamp",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetStartTimeStamp(IntPtr hDevice, out UInt64
pValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetTimeStamp", CallingConvention
= CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetTimeStamp(IntPtr hDevice, out UInt64 pValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetTimeStampPeriod",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetTimeStampPeriod(IntPtr hDevice, int lValue);
```

```
    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetTimeStampPeriod",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetTimeStampPeriod(IntPtr hDevice, out int
lValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceSetExMode", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceSetEXMode(IntPtr hDevice, byte bValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceGetExMode", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceGetExMode(IntPtr hDevice, out byte pbValue);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceWriteMessages",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceWriteMessages(IntPtr hDevice, [In, Out]
SlCan.Message[] pMsg, uint dwCount, out byte pbStatus);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceReadMessages", CallingConvention
= CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceReadMessages(IntPtr hDevice, uint dwTimeOut, [In,
Out] SlCan.Message[] pMsg, uint dwCount, out uint pdwCount);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceWriteMessagesEx",
CallingConvention = CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceWriteMessagesEx(IntPtr hDevice, [In, Out]
SlCan.TxMessage[] pMsg, uint dwCount, out byte pbStatus,out uint pdwCount);

    [DllImport("slcan.dll", EntryPoint = "SlCan_DeviceReadEvents", CallingConvention =
CallingConvention.StdCall, SetLastError = true)]
    public static extern bool DeviceReadEvents(IntPtr hDevice, uint dwTimeOut, [In,
Out] SlCan.Event[] pEvent, uint dwCount, out uint pdwCount);

    public static string DeviceGetProperty(IntPtr hDevice, uint infoindex)
    {
        StringBuilder buf = new StringBuilder(255);
        uint len, pdwInfo;
        len = DeviceGetProperty(hDevice, infoindex, buf, 250, out pdwInfo);
        return buf.ToString();
    }

}
```

slcanclass.vb

```
Imports System
Imports System.Text
Imports System.Runtime.InteropServices

Public Module SlCan
    Public Const PROPERTY_INDEX_LINKNAME As Integer = 0
    Public Const PROPERTY_INDEX_INSTANCEID As Integer = 1
    Public Const PROPERTY_INDEX_DEVICEDESC As Integer = 2
    Public Const PROPERTY_INDEX_FRIENDLYNAME As Integer = 3
    Public Const PROPERTY_INDEX_PHOBJECTNAME As Integer = 4
    Public Const PROPERTY_INDEX_MFG As Integer = 5
    Public Const PROPERTY_INDEX_LOCATIONINFO As Integer = 6
    Public Const PROPERTY_INDEX_ENUMERATOR As Integer = 7
    Public Const PROPERTY_INDEX_CLASS As Integer = 8
    Public Const PROPERTY_INDEX_CLASSGUID As Integer = 9
    Public Const PROPERTY_INDEX_SERVICE As Integer = 10
    Public Const PROPERTY_INDEX_DRIVER As Integer = 11
    Public Const PROPERTY_INDEX_PORTNAME As Integer = 12
```

```
Public Const PROPERTY_INDEX_PRODUCT As Integer = 13
Public Const PROPERTY_INDEX_MANUFACTURER As Integer = 14
Public Const PROPERTY_INDEX_CONFIGURATION As Integer = 15
Public Const PROPERTY_INDEX_INTERFACE As Integer = 16
Public Const PROPERTY_INDEX_SERIAL As Integer = 17
Public Const PROPERTY_INDEX_ALIAS As Integer = 18
Public Const PROPERTY_INDEX_CHANNELLINK As Integer = 19
Public Const PROPERTY_INDEX_SERIALID As Integer = 20

Public Const MODE_CONFIG As Integer = &H0
Public Const MODE_NORMAL As Integer = &H1
Public Const MODE_LISTENONLY As Integer = &H2
Public Const MODE_LOOPBACK As Integer = &H3
Public Const MODE_SLEEP As Integer = &H4

Public Const BR_CIA_1000K As UShort = &H8000
Public Const BR_CIA_800K As UShort = &H8001
Public Const BR_CIA_500K As UShort = &H8002
Public Const BR_CIA_250K As UShort = &H8003
Public Const BR_CIA_125K As UShort = &H8004
Public Const BR_CIA_50K As UShort = &H8005
Public Const BR_CIA_20K As UShort = &H8006
Public Const BR_CIA_10K As UShort = &H8007
Public Const BR_400K As UShort = &H8008
Public Const BR_200K As UShort = &H8009
Public Const BR_100K As UShort = &H800A
Public Const BR_83333 As UShort = &H800B
Public Const BR_33333 As UShort = &H800C
Public Const BR_25K As UShort = &H800D
Public Const BR_5K As UShort = &H800E
Public Const BR_30K As UShort = &H800F
Public Const BR_300K As UShort = &H8010
Public Const BR_LASTINDEX As UShort = BR_300K

Public Const CAP_MODE_NORMAL As Byte = &H1
Public Const CAP_MODE_LISTEN_ONLY As Byte = &H2
Public Const CAP_MODE_LOOP_BACK As Byte = &H4
Public Const CAP_MODE_SLEEP As Byte = &H8

Public Const CAP_TXMODE_ONE_SHOT As Byte = &H1
Public Const CAP_TXMODE_TIMESTAMP As Byte = &H2


Public Const CAP_CONTR_EXTERNAL As Byte = &H0
Public Const CAP_CONTR_MCP2515 As Byte = &H1
Public Const CAP_CONTR_SJA1000 As Byte = &H2

Public Const CAP_CONTR_INTERNAL As Byte = &H80
Public Const CAP_CONTR_LPC As Byte = &H81
Public Const CAP_CONTR_STM32 As Byte = &H82
Public Const CAP_CONTR_STM8 As Byte = &H83
Public Const CAP_CONTR_PIC As Byte = &H84
Public Const CAP_CONTR_PIC_ECAN As Byte = &H85

Public Const CAP_PHYS_HS As Byte = &H1
Public Const CAP_PHYS_LS As Byte = &H2
Public Const CAP_PHYS_SW As Byte = &H4
Public Const CAP_PHYS_J1708 As Byte = &H8
Public Const CAP_PHYS_LIN As Byte = &H10
Public Const CAP_PHYS_KLINE As Byte = &H20

Public Const CAP_PHYS_LOAD As Byte = &H1

Public Const CAP_BITRATE_INDEX As Byte = &H1
```

```
Public Const CAP_BITRATE_CUSTOM As Byte = &H2
Public Const CAP_BITRATE_AUTOMATIC As Byte = &H4

Public Const EVT_LEVEL_RX_MSG As Byte = 0
Public Const EVT_LEVEL_TIME_STAMP As Byte = 1
Public Const EVT_LEVEL_TX_MSG As Byte = 2
Public Const EVT_LEVEL_BUS_STATE As Byte = 3
Public Const EVT_LEVEL_COUNTS As Byte = 4
Public Const EVT_LEVEL_ERRORS As Byte = 5

Public Const EVT_TYPE_RX As Byte = &H0
Public Const EVT_TYPE_START_TX As Byte = &H1
Public Const EVT_TYPE_END_TX As Byte = &H2
Public Const EVT_TYPE_ABORT_TX As Byte = &H3
Public Const EVT_TYPE_BUS_STATE As Byte = &H4
Public Const EVT_TYPE_ERROR_COUNTS As Byte = &H5
Public Const EVT_TYPE_BUS_ERROR As Byte = &H6
Public Const EVT_TYPE_ARBITRATION_ERROR As Byte = &H7
Public Const EVT_STAMP_INC As Byte = &HF

Public Const BUS_STATE_ERROR_ACTIVE As Byte = &H0
Public Const BUS_STATE_ERROR_ACTIVE_WARN As Byte = &H1
Public Const BUS_STATE_ERROR_PASSIVE As Byte = &H2
Public Const BUS_STATE_BUSOFF As Byte = &H3

Public Const MES_INFO_EXT As Byte = &H1
Public Const MES_INFO_RTR As Byte = &H2
Public Const MES_INFO_ONESHOT As Byte = &H4

Public Const DEVOP_CREATE As UInteger = &H0
Public Const DEVOP_CREATEHANDLE As UInteger = &H1
Public Const DEVOP_OPEN As UInteger = &H2
Public Const DEVOP_CLOSE As UInteger = &H3
Public Const DEVOP_DESTROYHANDLE As UInteger = &H4
Public Const DEVOP_DESTROY As UInteger = &H5

Public Const INVALID_HANDLE_ERROR As Integer = &HE0001001
Public Const DEVICE_INVALID_HANDLE_ERROR As Integer = &HE0001120
Public Const HANDLE_INIT_ERROR As Integer = &HE0001017
Public Const DEVICE_NOTOPEN_ERROR As Integer = &HE0001121

Public Const EVT_ERR_TYPE_BIT As Byte = &H0
Public Const EVT_ERR_TYPE_FORM As Byte = &H1
Public Const EVT_ERR_TYPE_STUFF As Byte = &H2
Public Const EVT_ERR_TYPE_OTHER As Byte = &H3

Public Const EVT_ERR_DIR_TX As Byte = &H0
Public Const EVT_ERR_DIR_RX As Byte = &H1

Public Const EVT_ERR_FRAME_SOF As Byte = &H3
Public Const EVT_ERR_FRAME_ID28_ID21 As Byte = &H2
Public Const EVT_ERR_FRAME_ID20_ID18 As Byte = &H6
Public Const EVT_ERR_FRAME_SRTR As Byte = &H4
Public Const EVT_ERR_FRAME_IDE As Byte = &H5
Public Const EVT_ERR_FRAME_ID17_ID13 As Byte = &H7
Public Const EVT_ERR_FRAME_ID12_ID5 As Byte = &HF
Public Const EVT_ERR_FRAME_ID4_ID0 As Byte = &HE
Public Const EVT_ERR_FRAME_RTR As Byte = &HC
Public Const EVT_ERR_FRAME_RSRV0 As Byte = &HD
Public Const EVT_ERR_FRAME_RSRV1 As Byte = &H9
Public Const EVT_ERR_FRAME_DLC As Byte = &HB
Public Const EVT_ERR_FRAME_DATA As Byte = &HA
Public Const EVT_ERR_FRAME_CRC_SEQ As Byte = &H8
Public Const EVT_ERR_FRAME_CRC_DEL As Byte = &H18
```

```
Public Const EVT_ERR_FRAME_ACK_SLOT As Byte = &H19
Public Const EVT_ERR_FRAME_ACK_DEL As Byte = &H1B
Public Const EVT_ERR_FRAME_EOF As Byte = &H1A
Public Const EVT_ERR_FRAME_INTER As Byte = &H12
Public Const EVT_ERR_FRAME_AER_FLAG As Byte = &H11
Public Const EVT_ERR_FRAME_PER_FLAG As Byte = &H16
Public Const EVT_ERR_FRAME_TDB As Byte = &H13
Public Const EVT_ERR_FRAME_ERR_DEL As Byte = &H17
Public Const EVT_ERR_FRAME_OVER_FLAG As Byte = &H1C

Public Const TX_STATUS_OK As Byte = &H0
Public Const TX_STATUS_TIMEOUT As Byte = &H1
Public Const TX_STATUS_BUSOFF As Byte = &H2
Public Const TX_STATUS_ABORT As Byte = &H3
Public Const TX_STATUS_NOT_ENA As Byte = &H4
Public Const TX_STATUS_ERROR_ONE_SHOT As Byte = &H5
Public Const TX_STATUS_INVALID_MODE As Byte = &H6
Public Const TX_STATUS_UNKNOWN As Byte = &HF

Public Const PURGE_TX_ABORT As Byte = &H1
Public Const PURGE_RX_ABORT As Byte = &H2
Public Const PURGE_TX_CLEAR As Byte = &H4
Public Const PURGE_RX_CLEAR As Byte = &H8

Public Const INVALID_HANDLE_VALUE As Integer = &HFFFFFFFF

<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure Capability
    Public bModes As Byte
    Public bTXModes As Byte
    Public bMaxEventLevel As Byte
    Public bController As Byte
    Public bPhysical As Byte
    Public bPhysicalLoad As Byte
    Public bBitrates As Byte
    Public bAdvancedModes As Byte
    Public dwCanBaseClk As UInteger
    Public dwTimeStampClk As UInteger
    Public wMaxBRP As UShort
End Structure
<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure BitRate
    Public BRP As UShort
    Public TSEG1 As Byte
    Public TSEG2 As Byte
    Public SJW As Byte
    Public SAM As Byte
End Structure

<StructLayout(LayoutKind.Sequential, Pack:=1)> _
Public Structure Message
    Public Info As Byte
    Public ID As UInt32
    Public DataCount As Byte
    Public Data0 As Byte
    Public Data1 As Byte
    Public Data2 As Byte
    Public Data3 As Byte
    Public Data4 As Byte
    Public Data5 As Byte
    Public Data6 As Byte
    Public Data7 As Byte
End Structure
```

```
    <StructLayout(LayoutKind.Sequential, Pack:=1)> _
    Public Structure TxMessage
        Public Delay As Int32
        Public Info As Byte
        Public ID As UInt32
        Public DataCount As Byte
        Public Data0 As Byte
        Public Data1 As Byte
        Public Data2 As Byte
        Public Data3 As Byte
        Public Data4 As Byte
        Public Data5 As Byte
        Public Data6 As Byte
        Public Data7 As Byte
    End Structure

    <StructLayout(LayoutKind.Explicit, Size:=19, Pack:=1)> _
    Public Structure CEvent

        <FieldOffset(0)> Public EventType As Byte
        <FieldOffset(1)> Public TimeStampLo As UInt32
        <FieldOffset(5)> Public Info As Byte
        <FieldOffset(6)> Public ID As UInt32
        <FieldOffset(10)> Public DataCount As Byte
        <FieldOffset(11)> Public Data0 As Byte
        <FieldOffset(12)> Public Data1 As Byte
        <FieldOffset(13)> Public Data2 As Byte
        <FieldOffset(14)> Public Data3 As Byte
        <FieldOffset(15)> Public Data4 As Byte
        <FieldOffset(16)> Public Data5 As Byte
        <FieldOffset(17)> Public Data6 As Byte
        <FieldOffset(18)> Public Data7 As Byte

        <FieldOffset(5)> Public TimeStamp As UInt64

        <FieldOffset(5)> Public BusMode As Byte
        <FieldOffset(6)> Public Dummy1 As Byte
        <FieldOffset(7)> Public ErrCountRx As Byte
        <FieldOffset(8)> Public ErrCountTx As Byte
        <FieldOffset(9)> Public ErrType As Byte
        <FieldOffset(10)> Public ErrDir As Byte
        <FieldOffset(11)> Public ErrFrame As Byte
        <FieldOffset(12)> Public ErrArbitration As Byte
    End Structure

    <StructLayout(LayoutKind.Sequential, Pack:=1)> _
    Public Structure State
        Public BusMode As Byte
        Public Dummy1 As Byte
        Public ErrCountRX As Byte
        Public ErrCountTX As Byte

    End Structure



    Public Delegate Sub DeviceCallBack(ByVal hDevice As IntPtr, ByVal dwIndex As
UInteger, ByVal dwOperation As UInteger, ByVal pContext As IntPtr, ByVal dwContextSize
As UInteger)
    Public Delegate Sub DeviceListCallBack(ByVal hDevice As IntPtr, ByVal dwIndex As
UInteger, ByVal pContext As IntPtr, ByVal dwContextSize As UInteger)
    <DllImport("slcan.dll", EntryPoint:="SlCan_Load",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
```

```
    Public Function Load(ByVal DeviceProc As DeviceCallBack, ByVal DeviceListProc As
DeviceListCallBack) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_Free",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function Free(ByVal bDoCallBack As Boolean) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_Update",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function Update() As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_GetDeviceCount",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function GetDeviceCount() As UInt32
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_GetDevice",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function GetDevice(ByVal dwIndex As Integer) As IntPtr
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetProperty",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetProperty(ByVal hDevice As IntPtr, ByVal dwIndex As
UInteger, ByVal pInfo As StringBuilder, ByVal dwSize As Integer) As UInteger
    End Function

    <DllImport("slcan.dll", EntryPoint:="Slcan_DeviceSetContext",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)>
    Public Function DeviceSetContext(ByVal hDevice As IntPtr, ByVal pBuf As IntPtr,
ByVal dwBufSize As Integer) As IntPtr
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetContext",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetContext(ByVal hDevice As IntPtr) As IntPtr
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetContextSize",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetContextSize(ByVal hDevice As IntPtr) As Integer
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetCapabilities",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetCapabilities(ByVal hDevice As IntPtr, ByRef Cap As
Capability) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceOpen",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceOpen(ByVal hDevice As IntPtr) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceClose",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceClose(ByVal hDevice As IntPtr) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetMode",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
```

```
    Public Function DeviceSetMode(ByVal hDevice As IntPtr, ByVal dwMode As Integer) As
Boolean
    End Function


    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetMode",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetMode(ByVal hDevice As IntPtr, ByRef dwMode As Integer) As
Boolean

    End Function


    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetState",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetState(ByVal hDevice As IntPtr, ByRef dwState As UInteger)
As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetTXTimeOut",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetTXTimeOut(ByVal hDevice As IntPtr, ByVal dwMilliseconds
As UInteger) As Boolean
    End Function


    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetTXTimeOut",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetTXTimeOut(ByVal hDevice As IntPtr, ByRef dwMilliseconds
As UInteger) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetBitRate",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetBitRate(ByVal hDevice As IntPtr, ByRef bitrate As
BitRate) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetBitRate",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetBitRate(ByVal hDevice As IntPtr, ByRef bitrate As
BitRate) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceEnaRec",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceEnaRec(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetLatency",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetLatency(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetLatency",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetLatency(ByVal hDevice As IntPtr, ByRef pbValue As Byte)
As Boolean
    End Function
```

```
    <DllImport("slcan.dll", EntryPoint:="SlCan_DevicePurge",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DevicePurge(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetEventLevel",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetEventLevel(ByVal hDevice As IntPtr, ByVal bValue As Byte)
As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetEventLevel",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetEventLevel(ByVal hDevice As IntPtr, ByRef pbValue As
Byte) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetStartTimeStamp",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetStartTimeStamp(ByVal hDevice As IntPtr, ByRef Value As
UInt64) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetStartTimeStamp",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetStartTimeStamp(ByVal hDevice As IntPtr, ByRef pValue As
UInt64) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetTimeStamp",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetTimeStamp(ByVal hDevice As IntPtr, ByRef pValue As
UInt64) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetTimeStampPeriod",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetTimeStampPeriod(ByVal hDevice As IntPtr, ByVal lValue As
Integer) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetTimeStampPeriod",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetTimeStampPeriod(ByVal hDevice As IntPtr, ByRef lValue As
Integer) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceSetExMode",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceSetEXMode(ByVal hDevice As IntPtr, ByVal bValue As Byte) As
Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceGetExMode",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceGetExMode(ByVal hDevice As IntPtr, ByRef pbValue As Byte) As
Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceWriteMessages",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceWriteMessages(ByVal hDevice As IntPtr, <[In](), Out()> ByVal
pMsg() As Message, ByVal dwCount As UInteger, ByRef pbStatus As Byte) As Boolean
```

```
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceReadMessages",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceReadMessages(ByVal hDevice As IntPtr, ByVal dwTimeOut As
Integer, <[In](), Out()> ByVal pMsg() As Message, ByVal dwCount As UInteger, ByRef
pdwCount As UInteger) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceWriteMessagesEx",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceWriteMessagesEx(ByVal hDevice As IntPtr, <[In](), Out()>
ByVal pMsg() As TxMessage, ByVal dwCount As UInteger, ByRef pbStatus As Byte, ByRef
pdwCount As UInteger) As Boolean
    End Function

    <DllImport("slcan.dll", EntryPoint:="SlCan_DeviceReadEvents",
CallingConvention:=CallingConvention.StdCall, SetLastError:=True)> _
    Public Function DeviceReadEvents(ByVal hDevice As IntPtr, ByVal dwTimeOut As
UInteger, <[In](), Out()> ByVal pEvent() As CEvent, ByVal dwCount As UInteger, ByRef
pdwCount As UInteger) As Boolean
    End Function

    Public Function DeviceGetProperty(ByVal hDevice As IntPtr, ByVal infoindex As
Integer) As String

        Dim buf As New StringBuilder(255)
        Dim len As Integer
        len = DeviceGetProperty(hDevice, infoindex, buf, 250)
        Return buf.ToString()
    End Function


End Module
```