

USB5831 数据采集卡

WIN2000/XP 驱动程序使用说明书



北京阿尔泰科技发展有限公司
产品研发部修订

请您务必阅读《[使用纲要](#)》，他会使您事半功倍!

目 录

目 录	1
第一章 版权信息与命名约定	2
第一节、版权信息	2
第二节、命名约定	2
第二章 使用纲要	2
第一节、如何管理 USB 设备	2
第二节、如何批量取得 AD 数据	2
第三节、如何实现开关量的简便操作	4
第四节、哪些函数对您不是必须的	4
第三章 USB 设备专用函数接口介绍	4
第一节、设备驱动接口函数列表（每个函数省略了前缀“USB5831_”）	4
第二节、设备对象管理函数原型说明	5
第三节、AD 采样操作函数原型说明	9
第四节、AD 硬件参数系统保存与读取函数原型说明	11
第五节、DA 模拟量输出操作函数原型说明	13
第六节、DIO 数字开关量输入输出简易操作函数原型说明	14
第四章 硬件参数结构	15
第五章 数据格式转换与排列规则	18
第一节、AD 原始数据 LSB 转换成电压值 Volt 的换算方法	18
第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则	18
第三节、DA 电压值转换成 LSB 原码数据的换算方法	19
第六章 上层用户函数接口应用实例	20
第一节、简易程序演示说明	20
第二节、高级程序演示说明	20
第七章 基于 USB 总线的大容量连续数据采集详述	20
第八章 公共接口函数介绍	22
第一节、公用接口函数总列表（每个函数省略了前缀“USB5831_”）	22
第二节、线程操作函数原型说明	23

第一章 版权信息与命名约定

第一节、版权信息

本软件产品及相关套件均属北京市阿尔泰科贸有限公司所有，其产权受国家法律绝对保护，除非本公司书面允许，其他公司、单位及个人不得非法使用和拷贝，否则将受到国家法律的严厉制裁。您若需要我公司产品及相关信息请及时与我们联系，我们将热情接待。

第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 **USBxxxx_** 则被省略。如 **USB5831_CreateDevice** 则写为 **CreateDevice**。

二、函数名及参数中各种关键字缩写规则

缩写	全称	汉语意思	缩写	全称	汉语意思
Dev	Device	设备	DI	Digital Input	数字量输入
Pro	Program	程序	DO	Digital Output	数字量输出
Int	Interrupt	中断	CNT	Counter	计数器
Dma	Direct Memory Access	直接内存存取	DA	Digital convert to Analog	数模转换
AD	Analog convert to Digital	模数转换	DI	Differential	(双端或差分) 注：在常量选项中
Npt	Not Empty	非空	SE	Single end	单端
Para	Parameter	参数	DIR	Direction	方向
SRC	Source	源	ATR	Analog Trigger	模拟量触发
TRIG	Trigger	触发	DTR	Digital Trigger	数字量触发
CLK	Clock	时钟	Cur	Current	当前的
GND	Ground	地	OPT	Operate	操作
Lgc	Logical	逻辑的	ID	Identifier	标识
Phys	Physical	物理的			

以上规则不局限于该产品。

第二章 使用纲要

第一节、如何管理 USB 设备

由于我们的驱动程序采用面向对象编程，所以要使用设备的一切功能，则必须首先用 [CreateDevice](#) 函数创建一个设备对象句柄 **hDevice**，有了这个句柄，您就拥有了对该设备的控制权。然后将此句柄作为参数传递给其他函数，如 [InitDeviceAD](#) 可以使用 **hDevice** 句柄以初始化设备的 AD 部件并启动 AD 设备，[ReadDeviceAD](#) 函数可以用 **hDevice** 句柄实现对 AD 数据的采样批量读取，[SetDeviceDO](#) 函数可用实现开关量的输出等。最后可以通过 [ReleaseDevice](#) 将 **hDevice** 释放掉。

第二节、如何批量取得 AD 数据

当您有了 **hDevice** 设备对象句柄后，便可用 [InitDeviceAD](#) 函数初始化 AD 部件，关于采样通道、频率等的参数的设置是由这个函数的 **pADPara** 参数结构体决定的。您只需要对这个 **pADPara** 参数结构体的各个成员简单赋值即可实现所有硬件参数和设备状态的初始化，然后这个函数启动 AD 设备。接着便可用 [ReadDeviceAD](#) 反复读取 AD 数据以实现连续不间断采样当您需要关闭 AD 设备时，[ReleaseDeviceAD](#) 便可帮您实现（但设备对象 **hDevice** 依然存在）。（注：[ReadDeviceAD](#) 虽然主要面对批量读取，高速连续采集而设计，但亦可用它以少量点如 32 个点读取 AD 数据，以满足慢速采集需要）。具体执行流程请看下面的图 2.1.1。

注意：图中较粗的虚线表示对称关系。如红色虚线表示>CreateDevice和ReleaseDevice两个函数的关系是：最初执行一次>CreateDevice，在结束是就须执行一次ReleaseDevice。绿色虚线InitDeviceAD与ReleaseDeviceAD成对称方式出现。

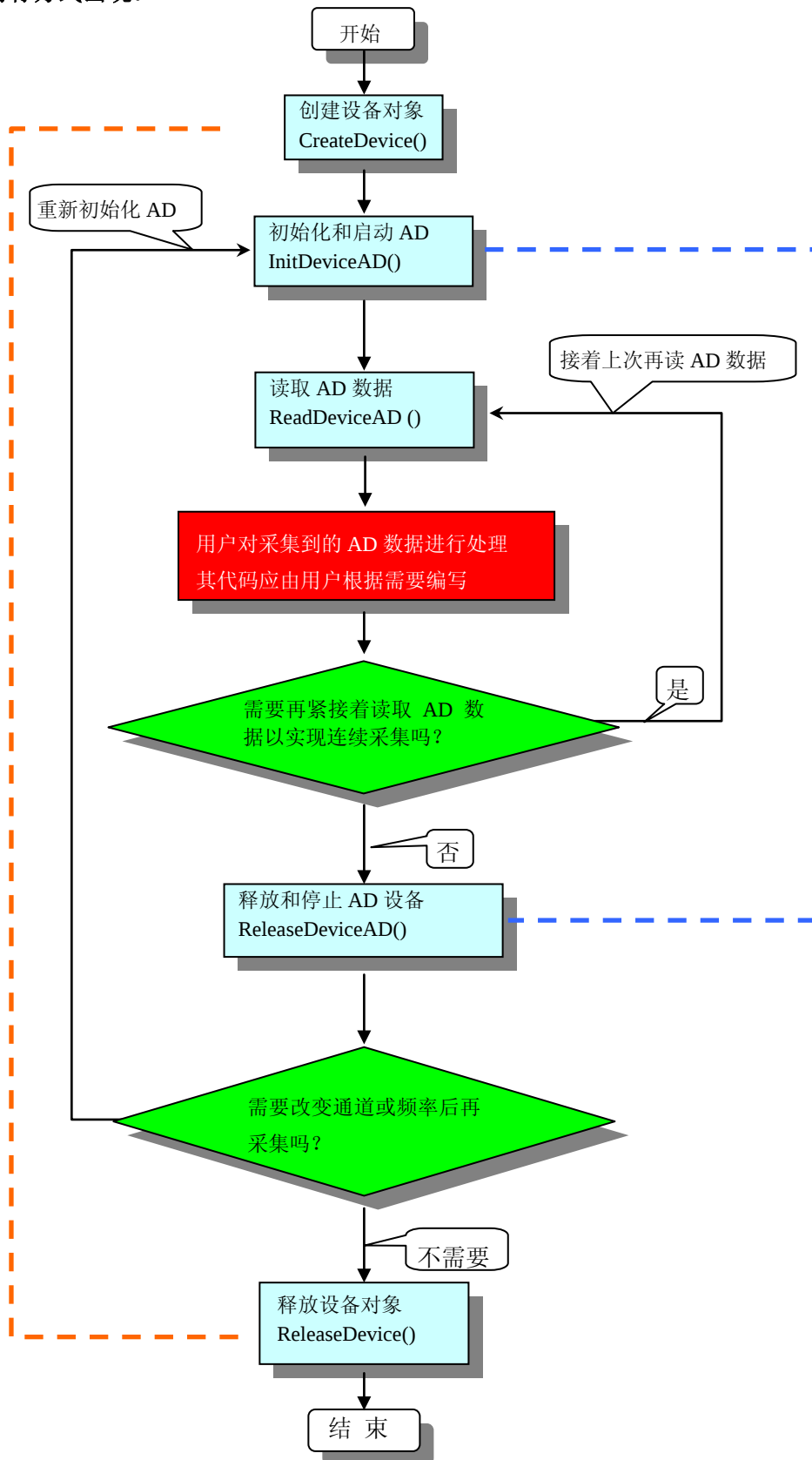


图 2.1.1 AD 采集实现过程

第三节、如何实现开关量的简便操作

当您有了 hDevice 设备对象句柄后，便可用[SetDeviceDO](#)函数实现开关量的输出操作，其各路开关量的输出状态由其 bDOSts[16]中的相应元素决定。由[GetDeviceDI](#)函数实现开关量的输入操作，其各路开关量的输入状态由其 bDISts[16]中的相应元素决定。

第四节、哪些函数对您不是必须的

当公共函数如[CreateFileObject](#)，[WriteFile](#)，[ReadFile](#)等一般来说都是辅助性函数，除非您要使用存盘功能。它们只是对我公司驱动程序的一种功能补充，对用户额外提供的。

第三章 USB 设备专用函数接口介绍

第一节、设备驱动接口函数列表（每个函数省略了前缀“USB5831_”）

本章函数是设备使用 USB 方式传输时所使用的。

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 USB 对象(用设备逻辑号)	上层及底层用户
CreateDeviceEx	创建 USB 对象(用设备物理号)	上层及底层用户
GetDeviceCount	取得设备总数	上层及底层用户
GetDeviceCurrentID	取得设备当前 ID 号	上层及底层用户
ListDeviceDlg	列表所有同一种 USB 各种配置	上层及底层用户
ResetDevice	复位 USB 设备	上层及底层用户
ReleaseDevice	关闭设备，且释放 USB 总线设备对象	上层及底层用户
② AD 采样操作函数		
InitDeviceAD	初始化 USB 设备 AD 部件，准备传数	上层用户
ReadDeviceAD	连续批量读取 USB 设备上的 AD 数据	上层用户
ReleaseDeviceAD	释放 USB 设备对象中的 AD 部件	上层用户
③ 辅助函数（硬件参数设置、保存、读取函数）		
LoadParaAD	从 Windows 系统中读取硬件参数	上层用户
SaveParaAD	往 Windows 系统保存硬件参数	上层用户
ResetParaAD	将注册表中的 AD 参数恢复至出厂默认值	上层用户
④ DA 输出操作函数		
WriteDeviceDA	输出模拟信号到指定通道	上层用户
⑤ 开关量函数		
GetDeviceDI	开关输入函数	上层用户
SetDeviceDO	开关输出函数	上层用户

使用需知

Visual C++:

首先将 USB5831.h 和 USB5831.lib 两个驱动库文件从相应的演示程序文件夹下复制到您的源程序文件夹中，然后在您的源程序头部添加如下语句，以便将驱动库函数接口的原型定义信息和驱动接口导入库 (USB5831.lib) 加入到您的工程中。

```
#include "USB5831.H"
```

在 VC 中，为了使用方便，避免重复定义和包含，您最好将以上语句放在 StdAfx.h 文件。一旦完成了以上工作，那么使用设备的驱动程序接口就跟使用 VC/C++Builder 自身的各种函数，其方法一样简单，毫

无二别。

关于 USB5831.h 和 USB5831.lib 两个文件均可在演示程序文件夹下面找到。

Visual Basic:

首先将 USB5831.Bas 驱动模块头文件从 VB 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的“添加模块”(Add Module)命令，在弹出的对话框中选择 USB5831.Bas 模块文件即可，一旦完成以上工作后，那么使用设备的驱动程序接口就跟使用 VB 自身的各种函数，其方法一样简单，毫无二别。

请注意，因考虑 Visual C++ 和 Visual Basic 两种语言的兼容问题，在下列函数说明和示范程序中，所举的 Visual Basic 程序均是需编译后在独立环境中运行。所以用户若在解释环境中运行这些代码，我们不保证能完全顺利运行。

LabView / CVI:

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的驱动程序接口的详细说明请参考其演示源程序。

第二节、设备对象管理函数原型说明

◆ 创建设备对象函数

函数原型:

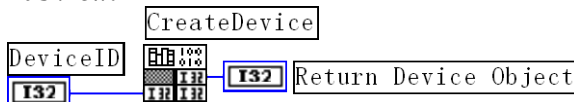
Visual C++:

`HANDLE CreateDevice(int DeviceLgcID = 0)`

Visual Basic:

`Declare Function CreateDevice Lib "USB5831_32" (Optional ByVal DeviceLgcID As Long = 0) As Long`

LabView:



功能: 该函数负责创建设备对象，并返回其设备对象句柄。

参数:

DeviceLgcID 设备逻辑 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 USB 设备时，系统将以该设备的“基本名称”与 DeviceLgcID 标识值为名称后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 USB5831 AD 模板时，系统则以“USB5831”作为基本名称，再以 DeviceLgcID 的初值组合成该设备的标识符“USB5831-0”来确认和管理这第一个设备，若用户接着再添加第二个 USB5831 AD 模板时，则系统将以“USB5831-1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 USB 设备时，DeviceLgcID 应置 0，第二应置 1，也以此类推。默认值为 0。

返回值: 如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数: [ReleaseDevice](#)

◆ 创建设备对象函数(扩展函数)

函数原型：

Visual C++:

HANDLE CreateDeviceEx(int DevicePhysID = 0)

Visual Basic:

Declare Function CreateDeviceEx Lib "USB5831_32" (Optional ByVal DevicePhysID = 0) As Long

LabView:

请参考相关演示程序。

功能：该函数负责创建设备对象，并返回其设备对象句柄。设备对象句柄是访问某一台设备的唯一依据。不同 DevicePhysID 创建的设备对象句柄用于访问不同的设备。只有成功创建 DevicePhysID 指定设备的句柄后，设备对用户来讲才是可用的。因为每一个访问设备的驱动函数接口都需要 hDevice 这个设备句柄参数。

参数：

DevicePhysID 设备物理 ID(Identifier)标识号。如果您使用单个 USB5831 设备，该参数等于 0 即可，且下面的内容您不必阅读。但如果您需要多卡工作时，此参数便大有用武之地，请阅读以下内容。

假如您所选用的产品只有 32 个 AD 通道，而您需要 128 路信号，那么您就需要同时使用四块卡来实现此功能。具体办法是您需要将 128 路信号依次分配到四个卡上，如下表：

卡号	信号通道	物理 ID 号	逻辑 ID 号
第一块	1~32	0	若第二个加载此卡，则为 1，否则为其他号
第二块	33~64	1	若最先加载此卡，则为 0，否则为其他号
第三块	65~96	2	若第四个加载此卡，则为 3，否则为其他号
第四块	97~128	3	若第三个加载此卡，则为 2，否则为其他号

从上表可知，如果使用您物理 ID 号，那么不管怎么安装您的硬件设备，其卡的物理编址不会发生任何变化，在软件上您用相应的物理 ID 号创建的设备对象所访问设备在物理顺序上不会发生变化，它始终对应于您的事先分配的信号通道。而使用逻辑 ID 号则不然，同样的逻辑 ID 值可能在不同的拔插顺序下会指向不同的物理设备而使软件的通道序列与硬件上无法一一对应。为了更好的保证物理通道序列与软件通道序列的对应关系，请务必保证板上的物理 ID 设置不重号。否则，[CreateDevice](#) ()只能创建重号中逻辑号最小的一个设备的对象，且具体创建的是哪一个设备也只能由用户根据采样信号特征等大概地确定。需要注意的是，物理 ID 号的设置有限的。如果实际设备数量超过这个有限值，那么超过的部分的物理 ID 号均设置为最大物理 ID 号，在创建设备对象时自动按逻辑 ID 处理。比如该产品的物理 ID 最多只能管理 16 个设备(ID 值为 0~15)，如果您要使用 18 个设备，那么为第 17、18 个设备创建设备对象时，其 DevicePhysID 应分别等于 16、17。

返回值：如果执行成功，则返回设备对象句柄；如果没有成功，则返回错误码 INVALID_HANDLE_VALUE。由于此函数已带容错处理，即若出错，它会自动弹出一个对话框告诉您出错的原因。您只需要对此函数的返回值作一个条件处理即可，别的任何事情您都不必做。

相关函数：[ReleaseDevice](#)

Visual C++ 程序举例

```

:
HANDLE hDevice; // 定义设备对象句柄
hDevice=CreateDevice(0); // 创建设备对象,并取得设备对象句柄
if(hDevice==INVALID_HANDLE_VALUE) // 判断设备对象句柄是否有效
{
    return; // 退出该函数
}
:
    
```

Visual Basic 程序举例

```

:
Dim hDevice As Long ' 定义设备对象句柄
hDevice = CreateDevice(0) ' 创建设备对象,并取得设备对象句柄,管理第一个 USB 设备
If hDevice = INVALID_HANDLE_VALUE Then ' 判断设备对象句柄是否有效

Else
    
```

```
Exit Sub ' 退出该过程
End If
:
```

◆ 取得在系统中的设备总台数

函数原型:

Visual C++:

`int GetDeviceCount (HANDLE hDevice)`

Visual Basic:

`Declare Function GetDeviceCount Lib "USB5831_32" (ByVal hDevice As Long) As Long`

LabView:

请参考相关演示程序。

功能: 取得在系统中物理设备的总台数。

参数: `hDevice` 设备对象句柄, 它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

返回值: 若成功, 则返回实际设备台数, 否则返回 0, 用户可以用[GetLastError](#)捕获错误码。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 取得当前设备对象句柄指向的设备所在的设备 ID

函数原型:

Visual C++:

`BOOL GetDeviceCurrentID (HANDLE hDevice,
PLONG DeviceLgcID,
PLONG DevicePhysID)`

Visual Basic:

`Declare Function GetDeviceCurrentID Lib "USB5831_32" (_
ByVal hDevice As Long,_
ByRef DeviceLgcID As Long,_
ByRef DevicePhysID As Long) As Boolean`

LabView:

请参考相关演示程序。

功能: 取得指定设备对象所代表的设备在设备链中的物理设备 ID 号和逻辑 ID 号。

参数:

`hDevice` 设备对象句柄, 它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

`DeviceLgcID` 设备逻辑 ID(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的 USB 设备时, 系统将以该设备的“基本名称”与 `DeviceLgcID` 标识值为名称后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 USB5831 AD 模板时, 系统则以“USB5831”作为基本名称, 再以 `DeviceLgcID` 的初值组合成该设备的标识符“USB5831-0”来确认和管理这第一个设备, 若用户接着再添加第二个 USB5831 AD 模板时, 则系统将以“USB5831-1”来确认和管理第二个设备, 若再添加, 则以此类推。所以当用户要创建设备句柄管理和操作第一个 USB 设备时, `DeviceLgcID` 应置 0, 第二应置 1, 也以此类推。默认值为 0。

`DevicePhysID` 指针参数, 取得指定设备的物理 ID。该号可以由用户设置板上 JP1 跳线器获得。当多卡工作时, 该物理 ID 号能让用户准确的辨别每一个卡所处的编址。该编址除了用户能手动改变以外, 不会随着系统加载卸载设备的顺序或者是用户插拔设备的顺序而改变其相应的物理编址, 它只会改变其逻辑编址, 即 `DeviceLgcID` 的值。比如您使用某一产品, 该产品只有 32 个 AD 通道, 而您需要 128 个通道采样, 那么您就需要四块卡来实现此功能。在实际采样中, 您需要对将 128 路信号依次分配到四个卡上, 如下表:

卡号	信号通道	物理 ID 号	逻辑 ID 号
第一块	1~32	0	若第二个加载此卡, 则为 1, 否则为其他号

第二块	33~64	1	若最先加载此卡, 则为 0, 否则为其他号
第三块	65~96	2	若第四个加载此卡, 则为 3, 否则为其他号
第四块	97~128	3	若第三个加载此卡, 则为 2, 否则为其他号

从上表可知, 如果使用您物理 ID 号, 那么不管怎么安装您的硬件设备, 那么其卡的物理编址不会发生任何变化, 在软件上您用相应的物理 ID 号创建的设备对象所访问设备在物理上不会发生变化, 它始终对应于您的事先分配的信号通道。而使用逻辑 ID 号则不然, 同样的逻辑 ID 值可能在不同的拔插顺序下会指向不同的物理设备而使软件的通道序列与硬件上无法一一对应。

返回值: 若成功, 则返回由 hDevice 参数代表的设备在设备链中的设备 ID, 否则返回-1, 用户可以用 [GetLastErrorEx](#) 捕获错误码。注意其返回的 ID 是一定与在 [CreateDevice](#) 函数中指定的 DeviceLgcID 参数值相等。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 用对话框控件列表计算机系统中所有 USB5831 设备各种配置信息

函数原型:

Visual C++:

BOOL ListDeviceDlg (void)

Visual Basic:

Declare Function ListDeviceDlg Lib "USB5831_32" () As Boolean

LabVIEW:

请参考相关演示程序。

功能: 列表系统中 USB5831 的硬件配置信息。

参数: 无。

返回值: 若成功, 则弹出对话框控件列表所有 USB5831 设备的配置情况。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 复位整个 USB 设备

函数原型:

Visual C++:

BOOL ResetDevice (HANDLE hDevice)

Visual Basic:

Declare Function ResetDevice Lib "USB5831_32" (ByVal hDevice As Long) As Boolean

LabView:

请参考相关演示程序。

功能: 复位整个 USB 设备, 相当于它与 PC 机端重新建立。其效果与重新插上 USB 设备等同。一般在出错情况下, 想软复位来建决重连接问题, 就可以调用该函数解决此问题。

参数: hDevice 设备对象句柄, 它应由 [CreateDevice](#) 或 [CreateDeviceEx](#) 创建。由它指向要复位的设备。

返回值: 若成功, 则返回 TRUE, 否则返回 FALSE, 用户可以用 [GetLastErrorEx](#) 捕获错误码。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 释放设备对象所占的系统资源及设备对象

函数原型:

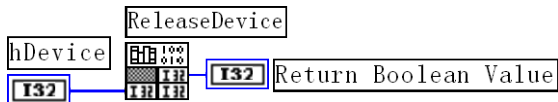
Visual C++:

BOOL ReleaseDevice(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDevice Lib "USB5831_32" (ByVal hDevice As Long) As Boolean

LabView:



功能：释放设备对象所占用的系统资源及设备对象自身。

参数：hDevice 设备对象句柄，它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

返回值：若成功，则返回 TRUE，否则返回 FALSE，用户可以用[GetLastErrorEx](#)捕获错误码。

相关函数：[CreateDevice](#)

应注意的是，[CreateDevice](#)必须和[ReleaseDevice](#)函数一一对应，即当您执行了一次[CreateDevice](#)，再一次执行这些函数前，必须执行一次[ReleaseDevice](#)函数，以释放由[CreateDevice](#)占用的系统软硬件资源，如系统内存等。只有这样，当您再次调用[CreateDevice](#)函数时，那些软硬件资源才可被再次使用。

第三节、AD 采样操作函数原型说明

◆ 初始化设备对象

函数原型：

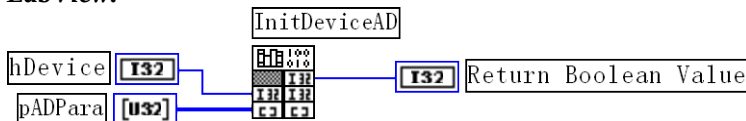
Visual C++:

```
BOOL InitDeviceAD( HANDLE hDevice,
                  PUSHB5831_PARA_AD pADPara )
```

Visual Basic:

```
Declare Function LibInitDeviceAD Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef pADPara As USB5831_PARA_AD) As Boolean
```

LabView:



功能：它负责初始化设备对象中的 AD 部件，为设备操作就绪有关工作，如预置 AD 采集通道，采样频率等，然后启动 AD 设备开始 AD 采集，随后，用户便可以连续调用[ReadDeviceAD](#)读取 USB 设备上的 AD 数据以实现连续采集。

参数：

hDevice 设备对象句柄，它应由 USB 设备的[CreateDevice](#)创建。

pADPara 设备对象参数结构，它决定了设备对象的各种状态及工作方式，如 AD 采样通道、采样频率等。

请参考《[AD 硬件参数介绍](#)》。

返回值：如果初始化设备对象成功，则返回 TRUE，且 AD 便被启动。否则返回 FALSE，用户可用 [GetLastError](#) 捕获当前错误码，并加以分析。

相关函数：[CreateDevice](#) [ReadDeviceAD](#) [ReleaseDevice](#)

注意：该函数将试图占用系统的某些资源，如系统内存区、DMA 资源等。所以当用户在反复进行数据采集之前，只须执行一次该函数即可，否则某些资源将会发生使用上的冲突，便会失败。除非用户执行了[ReleaseDeviceAD](#)函数后，再重新开始设备对象操作时，可以再执行该函数。所以该函数切忌不要单独放在循环语句中反复执行，除非和[ReleaseDeviceAD](#)配对。

◆ 批量读取 USB 设备上的 AD 数据

函数原型：

Visual C++:

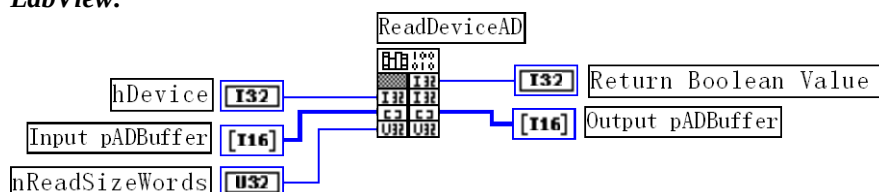
```
BOOL ReadDeviceAD (HANDLE hDevice,
                   USHORT ADBuffer[],
```

LONG nReadSizeWords,
PLONG nRetSizeWords = NULL)

Visual Basic:

Declare Function ReadDeviceAD Lib"USB5831_32" (_
ByVal hDevice As Long, _
ByRef ADBuffer As Integer, _
ByVal nReadSizeWords As Long, _
ByRef nRetSizeWords As Long) As Boolean

LabView:



功能: 读取 USB 设备 AD 部件上的批量数据。它不负责初始化 AD 部件,待读完整过指定长度的数据才返回。它必须在[InitDeviceAD](#)之后, [ReleaseDeviceAD](#)之前调用。

参数:

hDevice 设备对象句柄,它应由[CreateDevice](#)创建。

ADBuffer 用户数据缓冲区地址。接受的是从设备上采集的 LSB 原码数据,关于如何将 LSB 原码数据转换成电压值,请参考《[数据格式转换与排列规则](#)》章节。

nReadSizeWords 读取数据的长度(以字为单位),为了提高读取速率,根据特定要求,其长度必须指定为 256 字的整数倍长,如 256、512、1024 …… 8192 等字长,同时,数据长度也要为采样通道数的整数倍,以便于通道数据对齐处理,所以 nReadSizeWords 为(256*(LastChannel - FirstChannel + 1))的整数倍。否则,USB 设备对象将失败该读操作。

nRetSizeWords 在当前操作中该函数实际读取的点数。只有当函数成功返回时该参数值才有意义,而当函数返回失败时,则该参数的值与调用此函数前的值相等,不会因为函数被调用而改变,因此最好在读取 AD 数据前,将此参数值赋初值 0。需要注意的是在函数成功返回后,若此参数值等于 0,则需要重新调用此函数读取 AD 数据,直到此参数的值不等于 0 为止。

返回值: 若成功,则返回 TRUE,否则返回 FALSE,用户可以用 GetLastError 捕获错误码。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [ReleaseDevice](#)

◆ 释放设备对象中的 AD 部件

函数原型:

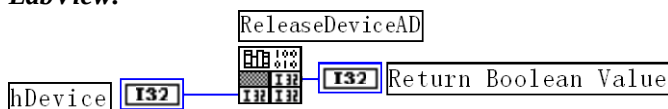
Visual C++:

BOOL ReleaseDeviceAD(HANDLE hDevice)

Visual Basic:

Declare Function ReleaseDeviceAD Lib"USB5831_32" (ByVal hDevice As Long) As Boolean

LabView:



功能: 释放设备对象中的 AD 部件所占用的系统资源。

参数: hDevice 设备对象句柄,它应由[CreateDevice](#)创建。

返回值: 若成功,则返回 TRUE,否则返回 FALSE,用户可以用 GetLastError 捕获错误码。

相关函数: [CreateDevice](#) [InitDeviceAD](#) [ReleaseDevice](#)

应注意的是，[InitDeviceAD](#)必须和[ReleaseDeviceAD](#)函数一一对应，即当您执行了一次[InitDeviceAD](#)，再一次执行这些函数前，必须执行一次[ReleaseDeviceAD](#)函数，以释放由[InitDeviceAD](#)占用的系统软硬件资源，如系统内存等。只有这样，当您再次调用[InitDeviceAD](#)函数时，那些软硬件资源才可被再次使用。这个对应关系对于非连续采样的场合特别适用。比如用户先采集一定长度的数据后，然后对根据这些数据或其他条件，需要改变采样通道或采样频率等配置时，则可以先用[ReleaseDeviceAD](#)释放先已由[InitDeviceAD](#)占用的资源，然后再用[InitDeviceAD](#)重新分配资源和初始化设备状态，即可实现所提到的功能。

❖ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [InitDeviceAD](#)
- ③ [ReadDeviceAD](#)
- ④ [ReleaseDeviceAD](#)
- ⑤ [ReleaseDevice](#)

用户可以反复执行第③步，以实现高速连续不间断数据采集。如果在采集过程中要改变设备状态信息，如采样通道等，则执行到第④步后再回到第②步用新的状态信息重新初始设备。

注意在第③步中，若其[ReadDeviceAD](#)函数成功返回，且 nRetSizeWords 参数值等于 0，则需要重新执行第③步，直到不等于 0 为止。

第四节、AD 硬件参数系统保存与读取函数原型说明

◆ 从 Windows 系统中读入硬件参数函数

函数原型：

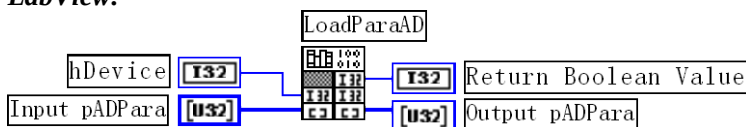
Visual C++:

`BOOL LoadParaAD(HANDLE hDevice,
PUSB5831_PARA_AD pADPara)`

Visual Basic:

`Declare Function LoadParaAD Lib "USB5831_32" (_
ByVal hDevice As Long,_
ByRef pADPara As USB5831_PARA_AD) As Boolean`

LabView:



功能：负责从 Windows 系统中读取设备硬件参数。

参数：

hDevice 设备对象句柄，它应由[CreateDevice](#)创建。

pADPara 属于 PUSB5831_PARA 的结构指针型，它负责返回 USB 硬件参数值，关于结构指针类型 PUSB5831_PARA 请参考相应 USB5831.h 或该结构的帮助文档的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： [CreateDevice](#) [SaveParaAD](#) [ReleaseDevice](#)

Visual C++ 举例：

```

:
USB5831_PARA_AD ADPara;
HANDLE hDevice;
HDevice = CreateDevice(0); // 管理一个 USB 设备
if(!LoadParaAD(hDevice, &ADPara))

```

```

{
    AfxMessageBox("读入硬件参数失败，请确认您的驱动程序是否正确安装");
    Return; // 若错误，则退出该过程
}
:

```

Visual Basic 举例:

```

:
Dim ADPara As USB5831_PARA_AD
Dim hDevice As Long :
hDevice = CreateDevice(0) ' 管理第一个 USB 设备
If Not LoadParaAD(hDevice, ADPara) Then
    MsgBox "读入硬件参数失败，请确认您的驱动程序是否正确安装"
    Exit Sub ' 若错误，则退出该过程
End If
:

```

◆ 往 Windows 系统写入设备硬件参数函数

函数原型:

Visual C++:

```

BOOL SaveParaAD(HANDLE hDevice,
                PUSB5831_PARA_AD pADPara)

```

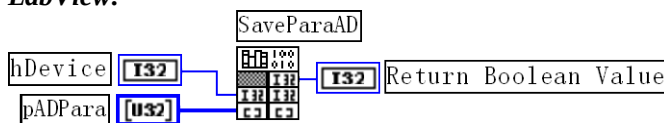
Visual Basic:

```

Declare Function SaveParaAD Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef pADPara As USB5831_PARA_AD) As Boolean

```

LabView:



功能: 负责把用户设置的硬件参数保存在 Windows 系统中，以供下次使用。

参数:

hDevice 设备对象句柄，它应由 [CreateDevice](#) 创建。

pADPara AD 设备硬件参数，请参考《[硬件参数结构](#)》章节。

返回值: 若成功，返回 TRUE，否则返回 FALSE。

相关函数: [CreateDevice](#) [LoadParaAD](#) [ReleaseDevice](#)

◆ AD 采样参数复位至出厂默认值函数

函数原型:

Visual C++:

```

BOOL ResetParaAD (HANDLE hDevice,
                  PUSB5831_PARA_AD pADPara)

```

Visual Basic:

```

Declare Function ResetParaAD Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef pADPara As USB5831_PARA_AD) As Boolean

```

LabVIEW:

请参考相关演示程序。

功能: 将系统中原来的 AD 参数值复位至出厂时的默认值。以防用户不小心将各参数设置错误造成一时无

法确定错误原因的后果。

参数：

hDevice 设备对象句柄，它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

pADPara 设备硬件参数，它负责在参数被复位后返回其复位后的值。关于 USB5831_PARA_AD 的详细介绍请参考 USB5831.h 或 USB5831.Bas 或 USB5831.Pas 函数原型定义文件，也可参考本文《[硬件参数结构](#)》关于该结构的有关说明。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数：[CreateDevice](#) [LoadParaAD](#) [SaveParaAD](#)
[ResetParaAD](#) [ReleaseDevice](#)

注意：在您编写工程应用软件时，若要更方便的保存和读取您特有的软件参数，请不防使用我们为您提供的辅助函数：[SaveParaInt](#)、[LoadParaInt](#)、[SaveParaString](#)、[LoadParaString](#)，详细说明请参考共用函数介绍章节中的《[其他函数原型说明](#)》。

第五节、DA 模拟量输出操作函数原型说明

◆ 输出模拟信号到指定通道

函数原型：

Visual C++:

```
BOOL WriteDeviceDA ( HANDLE hDevice,
                     LONG OutputRange,
                     SHORT nDADData,
                     int nDAChannel)
```

Visual Basic:

```
Declare Function WriteDeviceDA Lib WriteDeviceDA Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByVal OutputRange As Long, _
    ByVal nDADData As Integer, _
    ByVal nDAChannel As Long) As Boolean
```

LabVIEW:

请参考相关演示程序。

功能：输出 DA 原始数据。

参数：

hDevice 设备对象句柄，它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

OutputRange 输出量程。

常量名	常量值	功能定义
USB5831_OUTPUT_0_P5000mV	0x0000	0~5000mV
USB5831_OUTPUT_0_P10000mV	0x0001	0~10000mV
USB5831_OUTPUT_0_P10800mV	0x0002	0~10800mV
USB5831_OUTPUT_N5000_P5000mV	0x0003	±5000mV
USB5831_OUTPUT_N10000_P10000mV	0x0004	±10000mV
USB5831_OUTPUT_N10800_P10800mV	0x0005	±10800mV

nDADData 指输出的 DA 原始码数据，它的取值范围为[0, 4095]，它与实际输出模拟量值的对应关系请参考《[DA 电压值转换成 LSB 原码数据的换算方法](#)》章节。

nDAChannel 需要指定的模拟量通道号，其取值范围为[0, 3]。写入 4，代表四个通道都启动。

返回值: 若成功, 返回 TRUE, 输出 DA 原始数据; 否则返回 FALSE, 您可以调用[GetLastErrorEx](#)函数取得错误或错误字符信息。

相关函数: [CreateDevice](#) [ReleaseDevice](#)

◆ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [WriteDeviceDA](#)
- ③ [ReleaseDevice](#)

用户可以反复执行第②步, 以进行模拟量输出不断变化 (可以和 AD 采样同时进行, 互不影响)。

第六节、DI0 数字开关量输入输出简易操作函数原型说明

◆ 十三路开关量输入

函数原型:

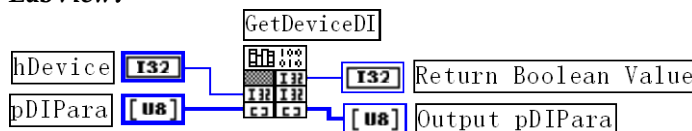
Visual C++:

```
BOOL GetDeviceDI (HANDLE hDevice,
                  BYTE bDIsTs[13])
```

Visual Basic:

```
Declare Function GetDeviceDI Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef bDIsTs As Byte) As Boolean
```

LabView:



功能: 负责将 USB 设备上的输入开关量状态读入内存。

参数:

hDevice 设备对象句柄, 它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

bDIsTs 十三路开关量输入状态的参数结构, 共有 13 个成员变量, 分别对应于 DI0~DI12 路开关量输入状态位。如果 bDIsTs[0]为“1”则使 0 通道处于开状态, 若为“0”则 0 通道为关状态。其他同理。具体定义请参考《[DI 数字开关量输入参数介绍](#)》章节。

返回值: 若成功, 返回 TRUE, 其 bDIsTs[x]中的值有效; 否则返回 FALSE, 其 bDIsTs[x]中的值无效。

相关函数: [CreateDevice](#) [SetDeviceDO](#) [ReleaseDevice](#)

◆ 十三路开关量输出

函数原型:

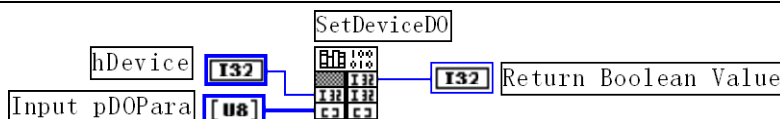
Visual C++:

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[13])
```

Visual Basic:

```
Declare Function SetDeviceDO Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef bDOSts As Byte) As Boolean
```

LabView:



功能：负责将 USB 设备上的输出开关量置成相应的状态。

参数：

hDevice 设备对象句柄，它应由[CreateDevice](#)或[CreateDeviceEx](#)创建。

bDOSets 十三路开关量输出状态的参数结构，共有 13 个成员变量，分别对应于 DO0~DO12 路开关量输出状态位。比如置 bDOSets[0]为“1”则使 0 通道处于“开”状态，若为“0”则置 0 通道为“关”状态。其他同理。请注意，在实际执行这个函数之前，必须对这个参数结构的 DO0 至 DO12 共 13 个成员变量赋初值，其值必须为“1”或“0”。具体定义请参考《[DO 数字开关量输出参数介绍](#)》。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： [CreateDevice](#) [GetDeviceDI](#) [ReleaseDevice](#)

❖ 以上函数调用一般顺序

- ① [CreateDevice](#)
- ② [SetDeviceDO](#)（或[GetDeviceDI](#)，当然这两个函数也可同时进行）
- ③ [ReleaseDevice](#)

用户可以反复执行第②步，以进行数字 I/O 的输入输出（数字 I/O 的输入输出及 AD 采样可以同时进行，互不影响）。

第四章 硬件参数结构

AD 硬件参数介绍（USB5831_PARA_AD）

Visual C++:

```

typedef struct _USB5831_PARA_AD      // 板卡各参数值
{
    LONG CheckStsMode;      // 检查状态模式
    LONG ADMode;            // AD 模式选择(连续采集/分组采集)
    LONG FirstChannel;      // 首通道,取值范围为[0, 15]
    LONG LastChannel;       // 末通道,取值范围为[0, 15]
    LONG Frequency;         // 采集频率,单位为 Hz,取值范围为[31, 250000]
    LONG InputRange;        // 量程选择
    LONG GroupInterval;     // 分组采样时的组间间隔(单位: 微秒),取值范围为[1, 32767]
    LONG LoopsOfGroup;      // 分组采样时, 每组循环次数, 取值范围为[1, 255]
    LONG Gains;             // 增益控制字
    LONG TriggerMode;       // 触发模式选择(软件触发、后触发)
    LONG TriggerSource;     // 触发源选择(ART、DTR)
    LONG TriggerType;       // 触发类型选择(边沿触发/脉冲触发)
    LONG TriggerDir;        // 触发方向选择(正向/负向触发)
    LONG TrigWindow;        // 触发灵敏度(1-255), 单位:0.5 微秒
    LONG GroundingMode;     // 接地方式 (单端或双端选择)
    LONG ClockSource;       // 时钟源选择(内/外时钟源)
    LONG bClockOutput;      // 是否允许本地 AD 转换时钟输出, =TRUE:允许输出到 CN1 上的 CLKOUT,
                             // =FALSE:禁止输出到 CN1 上的 CLKOUT
} USB5831_PARA_AD, *PUSB5831_PARA_AD;
  
```

Visual Basic:

Type USB5831_PARA_AD

CheckStsMode As Long;	' 检查状态模式
ADMode As Long;	' AD 模式选择(连续采集/分组采集)
FirstChannel As Long;	' 首通道,取值范围为[0, 15]
LastChannel As Long;	' 末通道,取值范围为[0, 15]
Frequency As Long;	' 采集频率,单位为 Hz,取值范围为[31, 250000]
InputRange As Long;	' 量程选择
GroupInterval As Long;	' 分组采样时的组间间隔(单位: 微秒),取值范围为[1, 32767]
LoopsOfGroup As Long;	' 分组采样时, 每组循环次数, 取值范围为[1, 255]
Gains As Long;	' 增益控制字
TriggerMode As Long;	' 触发模式选择(软件触发、后触发)
TriggerSource As Long;	' 触发源选择(ART、DTR)
TriggerType As Long;	' 触发类型选择(边沿触发/脉冲触发)
TriggerDir As Long;	' 触发方向选择(正向/负向触发)
TrigWindow As Long;	' 触发灵敏度(1-255), 单位:0.5 微秒
GroundingMode As Long;	' 接地方式 (单端或双端选择)
ClockSource As Long;	' 时钟源选择(内/外时钟源)
bClockOutput As Long;	' 是否允许本地 AD 转换时钟输出, =TRUE: 允许输出到 CN1 上的

CLKOUT, =FALSE:禁止输出到 CN1 上的 CLKOUT

End Type

LabView:

首先请您关注一下这个结构与前面 ISA 总线部分中的硬件参数结构 PARA 比较, 该结构实在太简短了。其原因就是在于 USB 设备是系统全自动管理的设备, 什么端口地址, 中断号, DMA 等将与 USB 设备的用户远告别, 一句话 USB 设备简单得就象使用电源插头一样。

硬件参数说明: 此结构主要用于设定设备硬件参数值, 用这个参数结构对设备进行硬件配置完全由 [InitDeviceAD](#) 函数完成。

[CheckStsMode](#) 检查状态模式。

常量名	常量值	功能定义
USB5831_CHKSTSMODE_HALF	0x00	查询 FIFO 半满标志
USB5831_CHKSTSMODE_NPT	0x01	查询 FIFO 非空标志

[ADMode](#) AD 模式选择(连续采集/分组采集)。

常量名	常量值	功能定义
USB5831_ADMODE_SEQUENCE	0x00	连续采样
USB5831_ADMODE_GROUP	0x01	分组采样

[FirstChannel](#) 首通道值, 取值范围应根据设备的总通道数设定, 本设备的 AD 采样首通道取值范围为 0~15, 要求首通道等于或小于末通道。

[LastChannel](#) 末通道值, 取值范围应根据设备的总通道数设定, 本设备的 AD 采样首通道取值范围为 0~15, 要求末通道大于或等于首通道。

注: 当首通道和末通道相等时, 即为单通道采集。

Frequency 采集频率，单位为 Hz，取值范围为[31, 250000]。

InputRange AD 量程选择：

常量名	常量值	功能定义
USB5831_INPUT_N10000_P10000mV	0x00	±10000mV
USB5831_INPUT_N5000_P5000mV	0x01	±5000mV
USB5831_INPUT_N2500_P2500mV	0x02	±2500mV
USB5831_INPUT_0_P10000mV	0x03	0~10000mV

GroupInterval 分组采样时的组间间隔(单位：微秒)，取值范围为[1, 32767]。

LoopsOfGroup 分组采样时，每组循环次数，取值范围为[1, 255]。

Gains 程控增益放大倍数，被采样的外界信号经通道开关选通后进入一个程控增益放大器如 AD8251 芯片，它可以将原始模拟信号放大指定倍数后再进入 AD 转换器被转换。

常量名	常量值	功能定义
USB5831_GAINS_1MULT	0x00	1 倍增益(使用 AD8251 放大器)
USB5831_GAINS_2MULT	0x01	2 倍增益(使用 AD8251 放大器)
USB5831_GAINS_4MULT	0x02	4 倍增益(使用 AD8251 放大器)
USB5831_GAINS_8MULT	0x03	8 倍增益(使用 AD8251 放大器)

TriggerMode 触发模式选择。

常量名	常量值	功能定义
USB5831_TRIGMODE_SOFT	0x00	软件触发(属于内触发)
USB5831_TRIGMODE_POST	0x01	硬件后触发(属于外触发)

TriggerSource 触发源选择。

常量名	常量值	功能定义
USB5831_TRIGSOURCE_ATR	0x00	ATR 触发
USB5831_TRIGSOURCE_DTR	0x01	DTR 触发

TriggerType 触发类型选择。

常量名	常量值	功能定义
USB5831_TRIGTYPE_EDGE	0x00	边沿触发
USB5831_TRIGTYPE_PULSE	0x01	脉冲触发(电平类型)

TriggerDir 触发方向选择。

常量名	常量值	功能定义
USB5831_TRIGDIR_NEGATIVE	0x00	负向触发(低电平/下降沿触发)
USB5831_TRIGDIR_POSITIVE	0x01	正向触发(高电平/上升沿触发)
USB5831_TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低电平或上升/下降沿触发)

TrigWindow 触发灵敏度(1-255)，单位：0.5 微秒。

GroundingMode AD 接地方式（单端或双端选择）。

常量名	常量值	功能定义
USB5831_GNDMODE_SE	0x00	单端方式(SE:Single end)
USB5831_GNDMODE_DI	0x01	双端方式(DI:Differential)

ClockSource 时钟源选择。

常量名	常量值	功能定义
USB5831_CLOCKSRC_IN	0x00	内部时钟定时触发
USB5831_CLOCKSRC_OUT	0x01	外部时钟定时触发

bClockOutput 是否允许本地 AD 转换时钟输出, =TRUE:允许输出到 CN1 上的 CLKOUT, =FALSE:禁止输出到 CN1 上的 CLKOUT。

相关函数: [InitDeviceAD](#) [LoadParaAD](#) [SaveParaAD](#)

第五章 数据格式转换与排列规则

第一节、AD 原始数据 LSB 转换成电压值 Volt 的换算方法

在换算过程中弄清模板精度 (即 Bit 位数) 是很关键的, 因为它决定了 LSB 数码的总宽度 CountLSB。比如 12 位的模板 CountLSB 为 4096。其他类型同理均按 $2^n = \text{LSB 总数}$ (n 为 Bit 位数) 换算即可。

量程(毫伏)	计算机语言换算公式(标准 C 语法)	Volt 取值范围 mV
±10000	$\text{Volt} = (20000.00/8192) * (\text{ADBuffer}[0] \& 0x1FFF) - 10000.00$	[-10000.00, + 9997.55]
±5000	$\text{Volt} = (10000.00/8192) * (\text{ADBuffer}[0] \& 0x1FFF) - 5000.00$	[-5000.00, + 4998.77]
±2500	$\text{Volt} = (5000.00/8192) * (\text{ADBuffer}[0] \& 0x1FFF) - 2500.00$	[-2500.00, + 2499.38]
0~10000	$\text{Volt} = (10000.00/8192) * (\text{ADBuffer}[0] \& 0x1FFF)$	[0.00, + 9998.77]

换算举例: (设量程为 ±10000mV, 这里只转换第一个点)

Visual C++:

```
USHORT Lsb; // 定义存放标准 LSB 原码的变量
float Volt; // 定义存放转换后的电压值的变量
float PerLsbVolt = 20000.00/8192; // 求出每个 LSB 原码单位电压值
Lsb = ADBuffer[0] & 0x1FFF;
Volt = PerLsbVolt * Lsb - 10000.00; // 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

Visual Basic:

```
Dim Lsb As Long ' 定义存放标准 LSB 原码的变量
Dim Volt As Single ' 定义存放转换后的电压值的变量
Dim PerLsbVolt As Single
PerLsbVolt = 20000.00/8192 ' 求出每个 LSB 原码单位电压值
Lsb = ADBuffer(0) AND &H1FFF ' 将其转换成无符号 16 位有效数据
Volt = PerLsbVolt * Lsb - 10000.00 ' 用单位电压值与 LSB 原码数量相乘减去偏移求得实际电压值
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

当首末通道相等时, 即为单通道采集, 假如 **FirstChannel**=5, **LastChannel**=5, 其排放规则如下:

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
通道号	5	5	5	5	5	5	5	5	5	5	5	5	5	5	5	...

两通道采集(CH0 - CH1)

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
通道号	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	...

四通道采集(CH0 - CH3)

数据缓冲区索引号	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	...
通道号	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	...

其他通道方式以此类推。

如果用户是进行连续不间断循环采集，即用户只进行一次初始化设备操作，然后不停的从设备上读取 AD 数据，那么需要用户特别注意的是应处理好各通道数据排列和对齐问题，尤其任意通道数采集时。否则，用户无法将规则排在缓冲区中的各通道数据正确分离出来。怎样正确处理呢？我们建议的方法是，每次从设备上读取的点数应置为所选通道数量的整数倍长（在 USB 设备上同时也应 32 的整数倍），这样便能保证每读取的这批数据在缓冲区中的相应位置始终固定对应于某一个通道的数据。比如用户要求对 1、2 两个 AD 通道的数据进行连续循环采集，则置每次读取长度为其 2 的整倍长 $2n$ （ n 为每个通道的点数），这里设为 2048。试想，如此一来，每次读取的 2048 个点中的第一个点始终对应于 1 通道数据，第二个点始终对应于 2 通道，第三个点再应于 1 通道，第四个点再对应于 2 通道……以此类推。直到第 2047 个点对应于 1 通道数据，第 2048 个点对应 2 通道。这样一来，每次读取的段长正好包含了从首通道到末通道的完整轮回，如此一来，用户只须按通道排列规则，按正常的处理方法循环处理每一批数据。而对于其他情况也是如此，比如 3 个通道采集，则可以使用 $3n$ （ n 为每个通道的点数）的长度采集。为了更加详细地说明问题，请参考下表（演示的是采集 1、2、3 共三个通道的情况）。由于使用连续采样方式，所以表中的数据序列一行的数字变化说明了数据采样的连续性，即随着时间的延续，数据的点数连续递增，直至用户停止设备为止，从而形成了一个有相当长度的连续不间的多通道数据链。而通道序列一行则说明了随着连续采样的延续，其各通道数据在其整个数据链中的排放次序，这是一种非常规则而又绝对严格的顺序。但是这个相当长度的多通道数据链则不可能一次通过设备对象函数如 [ReadDeviceAD](#) 函数读回，即便不考虑是否能一次读完的问题，但对用户的实时数据处理要求来说，一次性读取那么长的数据，则往往是相当矛盾的。因此我们就得分若干次分段读取。但怎样保证既方便处理，又不易出错，而且还高效。还是正如前面所说，采用通道数的整数倍长读取每一段数据。如表中列举的方法 1（为了说明问题，我们每读取一段数据只读取 $2n$ 即 $3*2=6$ 个数据）。从方法 1 不难看出，每一段缓冲区中的数据在相同缓冲区索引位置都对应于同一个通道。而在方法 2 中由于每次读取的不是通道整数倍长，则出现问题，从表中可以看出，第一段缓冲区中的 0 索引位置上的数据对应的是第 1 通道，而第二段缓冲区中的 0 索引位置上的数据则对应于第 2 通道的数据，而第三段缓冲区中的数据则对应于第 3 通道……，这显然不利于循环有效处理数据。

在实际应用中，我们在遵循以上原则时，应尽可能地使每一段缓冲足够大，这样，可以一定程度上减少数据采集程序和数据处理程序的 CPU 开销量。

数据序列	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	...
通道序列	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	1	2	3	...
方法 1	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	3	4	5	0	1	2	...
缓冲区号	第一段缓冲						第二段缓冲区						第三段缓冲区						第 n 段缓冲			
方法 2	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3	0	...
	第一段缓冲区				第二段缓冲区				第三段缓冲区				第四段缓冲区				第五段缓冲区				第 n 段缓	

第三节、DA 电压值转换成 LSB 原码数据的换算方法

量程(伏)	计算机语言换算公式(标准 C 语法)	Lsb 取值范围
0~5000mV	$Lsb = Volt / (5000.00 / 4096)$	[0, 4095]
0~10000mV	$Lsb = Volt / (10000.00 / 4096)$	[0, 4095]
0~10800mV	$Lsb = Volt / (10800.00 / 4096)$	[0, 4095]
±5000mV	$Lsb = Volt / (10000.00 / 4096) + 2048$	[0, 4095]
±10000mV	$Lsb = Volt / (20000.00 / 4096) + 2048$	[0, 4095]
±10800mV	$Lsb = Volt / (21600.00 / 4096) + 2048$	[0, 4095]

请注意这里求得的 LSB 数据就是用于 [WriteDeviceDA](#) 中的 nDADData 参数的。

第六章 上层用户函数接口应用实例

如果您想快速的了解驱动程序的使用方法和调用流程,以最短的时间建立自己的应用程序,那么我们强烈建议您参考相应的简易程序。此种程序属于工程级代码,可以直接打开不用作任何配置和代码修改即可编译通过,运行编译链接后的可执行程序,即可看到预期效果。

如果您想了解硬件的整体性能、精度、采样连续性等指标以及波形显示、数据存盘与分析、历史数据回放等功能,那么请参考高级演示程序。特别是许多不愿意编写任何程序代码的用户,您可以使用高级程序进行采集、显示、存盘等功能来满足您的要求。甚至可以用我们提供的专用转换程序将高级程序采集的存盘文件转换成相应格式,即可在 Excel、MatLab 第三方软件中分析数据(此类用户请最好选用通过 Visual C++制作的高级演示系统)。

第一节、简易程序演示说明

一、怎样使用 [ReadDeviceAD](#) 函数进行 AD 连续数据采集

其详细应用实例及工程级代码请参考 Visual C++简易演示系统及源程序,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程(主要参考 USB5831.h 和 Sys.cpp)。

[程序] □ [阿尔泰测控演示系统](#) □ [\[USB5831 16 路 AD、4 路 DA、13 路 DIO 卡\]](#) □ [\[Microsoft Visual C++\]](#) □ [\[简易代码演示\]](#) □ [\[AD 采集演示源程序\]](#)

其简易程序默认存放路径为: 系统盘\ART\USB5831\SAMPLES\VC\SIMPLE\AD

二、怎样使用 [SetDeviceDO](#) 和 [GetDeviceDI](#) 函数进行开关量输入输出操作

其详细应用实例及正确代码请参考 Visual C++简易演示系统及源程序,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程(主要参考 USB5831.h 和 Sys.cpp)。

[程序] □ [阿尔泰测控演示系统](#) □ [\[USB5831 16 路 AD、4 路 DA、13 路 DIO 卡\]](#) □ [\[Microsoft Visual C++\]](#) □ [\[简易代码演示\]](#) □ [\[开关量演示源程序\]](#)

其默认存放路径为: 系统盘\ART\USB5831\SAMPLES\VC\SIMPLE\DIO

第二节、高级程序演示说明

高级程序演示了本设备的所有功能,您先点击 Windows 系统的[开始]菜单,再按下列顺序点击,即可打开基于 VC 的 Sys 工程(主要参考 USB5831.h 和 DADoc.cpp)。

[程序] □ [阿尔泰测控演示系统](#) □ [\[USB5831 16 路 AD、4 路 DA、13 路 DIO 卡\]](#) □ [\[Microsoft Visual C++\]](#) □ [\[高级代码演示\]](#)

其默认存放路径为: 系统盘\ART\USB5831\SAMPLES\VC\ADVANCED

其他语言的演示可以用上面类似的方法找到。

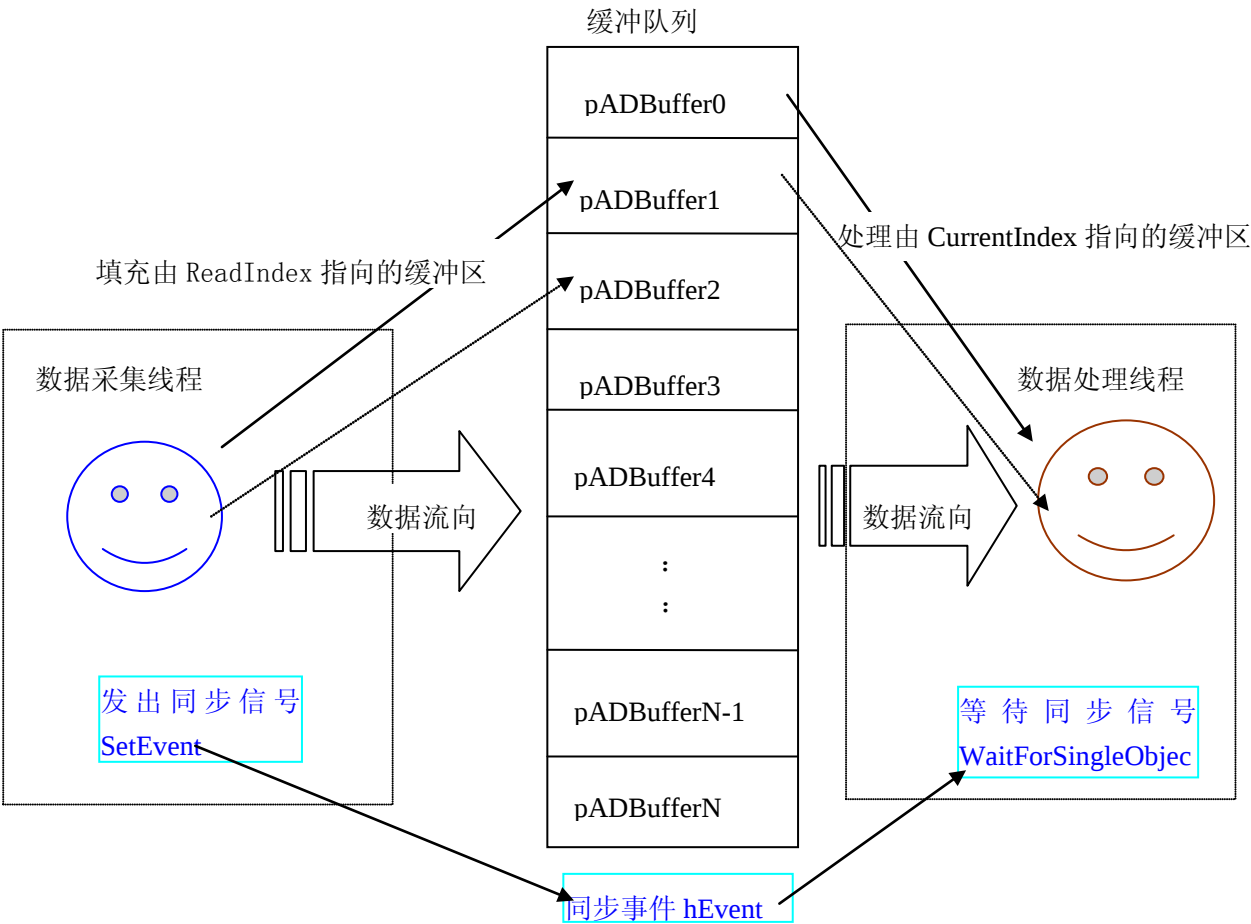
第七章 基于 USB 总线的大容量连续数据采集详述

与 ISA、PCI 设备同理,使用子线程跟踪 AD 转换进度,并进行数据采集是保持数据连续不间断的最佳方案。但是与 ISA 总线设备不同的是,USB 设备在这里不使用动态指针去同步 AD 转换进度,因为 ISA 设备环形内存池的动态指针操作是一种软件化的同步,而 USB 设备不再有软件化的同步,而完全由硬件和驱动程序自动完成。这样一来,用户要用程序方式实现连续数据采集,其软件实现就显得极为容易。每次用 [ReadDeviceAD](#) 函数读取 AD 数据时,那么设备驱动程序会按照 AD 转换进度将 AD 数据一一放进用户数据缓冲区,当完成该次所指定的点数时,它便会返回,当您再次用这个函数读取数据时,它会接着上一次的位置传递数据到用户数据缓冲区。只是要求每两次 [ReadDeviceAD](#) 之间的时间间隔越短越好。

但是由于我们的设备是通常工作在一个单 CPU 多任务的环境中,由于任务之间的调度切换非常平凡,特别是当用户移动窗口、或弹出对话框等,则会使当前线程猛地花掉大量的时间去处理这些图形操作,因此如果处理不当,则将无法实现高速连续不间断采集,那么如何更好的克服这些问题呢?用子线程则是必须的(在这

里我们称之为数据采集线程)，但这还不够，必须要求这个线程是绝对的工作者线程，即这个线程在正常采集不能有任何窗口等图形操作。只有这样，当用户进行任何窗口操作时，这个线程才不会被堵塞，因此可以保证正常连续的数据采集。但是用户可能要问，不能进行任何窗口操作，那么我如何将采集的数据显示在屏幕上呢？其实很简单，再开辟一个子线程，我们称之为数据处理线程，也叫用户界面线程。最初，数据处理线程不做任何工作，而是在 Win32 API 函数 `WaitForSingleObject` 的作用下进入睡眠状态，此时它不消耗 CPU 任何时间，即可保证其他线程代码有充分的运行机会（这里当然主要指数据采集线程），当数据采集线程取得指定长度的数据到用户空间时，则再用 Win32 API 函数 `SetEvent` 将指定事件消息发送给数据处理线程，则数据处理线程即刻恢复运行状态，迅速对这批数据进行处理，如计算、在窗口绘制波形、存盘等操作。

可能用户还要问，既然数据处理线程是非工作者线程，那么如果用户移动窗口等操作堵塞了该线程，而数据采集线程则在不停地采集数据，那数据处理线程难道不会因此而丢失数据采集线程发来的某一段数据吗？如果不另加处理，这个情况肯定有发生的可能。但是，我们采用了一级缓冲队列和二级缓冲队列的设计方案，足以避免这个问题。即假设数据采集线程每一次从设备上取出 8K 数据，那么我们就创建一个缓冲队列，在用户程序中最简单的办法就是开辟一个二维数组如 `ADBuffer[Count][DataLen]`，我们将 `DataLen` 视为数据采集线程每次采集的数据长度，`Count` 则为缓冲队列的成员个数。您应根据您的计算机物理内存大小和总体使用情况来设定这个数。假如我们设成 32，则这个缓冲队列实际上就是数组 `ADBuffer[32][8192]` 的形式。那么如何使用这个缓冲队列呢？方法很简单，它跟一个普通的缓冲区如一维数组差不多，唯一不同是，两个线程首先要通过改变 `Count` 字段的值，即这个下标 `Index` 的值来填充和引用由 `Index` 下标指向某一段 `DataLen` 长度的数据缓冲区。需要注意的两个线程不共用一个 `Index` 下标变量。具体情况是当数据采集线程在 AD 部件被 `InitDeviceAD` 初始化之后，首次采集数据时，则将自己的 `ReadIndex` 下标置为 0，即用第一个缓冲区采集 AD 数据。当采集完后，则向数据处理线程发送消息，且两个线程的公共变量 `SegmentCounts` 加 1，（注意 `SegmentCounts` 变量是用于记录当前时刻缓冲队列中有多少个已被数据采集线程使用了，但是却没被数据处理线程处理掉的缓冲区数量。）然后再接着将 `ReadIndex` 偏移至 1，再用第二个缓冲区采集数据。再将 `SegmentCounts` 加 1，至到 `ReadIndex` 等于 15 为止，然后再回到 0 位置，重新开始。而数据处理线程则在每次接受到消息时判断有多少由于自己被堵塞而没有被处理的缓冲区个数，然后逐一进行处理，最后再从 `SegmentCounts` 变量中减去在所接受到的当前事件下所处理的缓冲区个数。因此，即便应用程序突然很忙，使数据处理线程没有时间处理已到来的数据，但是由于缓冲区队列的缓冲作用，可以让数据采集线程先将数据连续缓存在这个区域中，由于这个缓冲区可以设计得比较大，因此可以缓冲很大的时间，这样即便是数据处理线程由于系统的偶而繁忙而被堵塞，也很难使数据丢失。而且通过这种方案，用户还可以在数据采集线程中对 `SegmentCounts` 加以判断，观察其值是否大小了 32，如果大于，则缓冲区队列肯定因数据处理采集的过度繁忙而被溢出，如果溢出即可报警。因此具有强大的容错处理。



下面只是简要的策略说明，其详细应用实例请参考 Visual C++测试与演示系统，您先点击 Windows 系统的 [开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程。

[程序] □ [阿尔泰测控演示系统] □ [Microsoft Visual C++ □ [高级演示程序]

下面只是基于 C 语言的简要的策略说明，其详细应用实例及正确代码请参考 Visual C++测试与演示系统，您先点击 Windows 系统的 [开始]菜单，再按下列顺序点击，即可打开基于 VC 的 Sys 工程(ADDoc.h 和 ADDoc.cpp)。

[程序] □ [阿尔泰测控演示系统] □ [Microsoft Visual C++ □ [高级演示程序]

然后，您着重参考 ADDoc.cpp 源文件中以下函数：

```
void CADDoc::StartDeviceAD()           // 启动线程函数
UINT ReadDataThread (PVOID hWnd)      // 读数据线程
UINT ProcessDataThread (PVOID hWnd)   // 绘制数据线程
void CADDoc::StopDeviceAD()           // 终止采集函数
```

第八章 公共接口函数介绍

这部分函数不参与本设备的实际操作，它只是为您编写数据采集与处理程序时的有力手段，使您编写应用程序更容易，使您的应用程序更高效。

第一节、公用接口函数总列表（每个函数省略了前缀“USB5831_”）

函数名	函数功能	备注
① USB 总线内存映射寄存器操作函数		
GetDevVersion	获取设备固件及程序版本	底层用户

② 创建 Visual Basic 子线程，线程数量可达 32 个以上		
CreateSystemEvent	创建系统内核事件对象	用于线程同步或中断
ReleaseSystemEvent	释放系统内核事件对象	

第二节、USB 内存映射寄存器操作函数原型说明

◆ 取得指定的指定设备寄存器组 BAR 地址

函数原型:

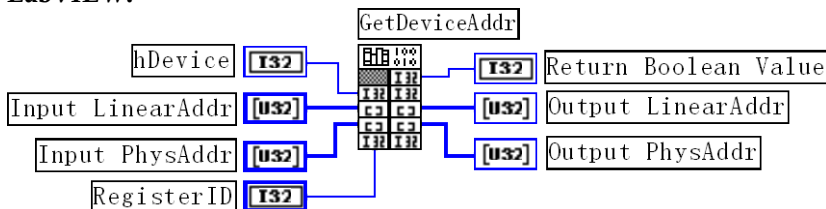
Visual C++:

```
BOOL GetDevVersion (HANDLE hDevice,
                    PULONG pulFmwVersion,
                    PULONG pulDriverVersion)
```

Visual Basic:

```
Declare Function GetDevVersion Lib "USB5831_32" ( _
    ByVal hDevice As Long, _
    ByRef pulFmwVersion As Long, _
    ByRef pulDriverVersion As Long) As Boolean
```

LabVIEW:



功能: 取得指定的指定设备寄存器组 BAR 地址。

参数:

hDevice 设备对象句柄，它应由[CreateDevice](#)创建。

返回值: 如果执行成功，则返回 TRUE, 否则会返回 FALSE。

相关函数: [CreateDevice](#) [GetDevVersion](#)

第三节、线程操作函数原型说明

◆ 创建内核系统事件

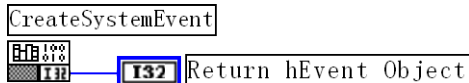
Visual C++:

```
DEVAPI CreateSystemEvent(void)
```

Visual Basic:

```
Declare Function CreateSystemEvent Lib "USB5831_32" () As Long
```

LabVIEW:



功能: 创建系统内核事件对象，它将被用于中断事件响应或数据采集线程同步事件。

参数: 无任何参数。

返回值: 若成功，返回系统内核事件对象句柄，否则返回 -1(或 INVALID_HANDLE_VALUE)。

◆ 释放内核系统事件

Visual C++:

BOOL ReleaseSystemEvent(HANDLE hEvent)

Visual Basic:

Declare Function ReleaseSystemEvent Lib"USB5831_32" (ByVal hEvent As Long) As Boolean

LabVIEW:

请参见相关演示程序。

功能: 释放系统内核事件对象。

参数: hEvent 被释放的内核事件对象。它应由[CreateSystemEvent](#)成功创建的对象。

返回值: 若成功, 则返回 TRUE。