# USB5529 数据采集卡

## WIN2000/XP 驱动程序使用说明书

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



**提醒用户：**

通常情况下，WINDOWS 系统在安装时自带的 DLL 库和驱动不全，所以您不管使用那种语言编程，请您最好先安装上 Visual C++6.0 版本的软件，方可使我们的驱动程序有更完备的运行环境。

**有关设备驱动安装和产品二次发行请参考 USB5529Inst.doc 文档。**

# 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 USB5529_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、如何用读取计数器

读取计时器比较简单，我们先调用CreateDevice来创建一设备对象，创建成功后可以通过EnableDeviceCNT使能计数器，再用SetDeviceCNT来设置加计数或减计数，然后就可以按要求调用GetDeviceCNT取得所有计数器的状态信息，DisableDeviceCNT则禁止计数器。最后调用ReleaseDevice释放掉设备（必需的）。

**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。**

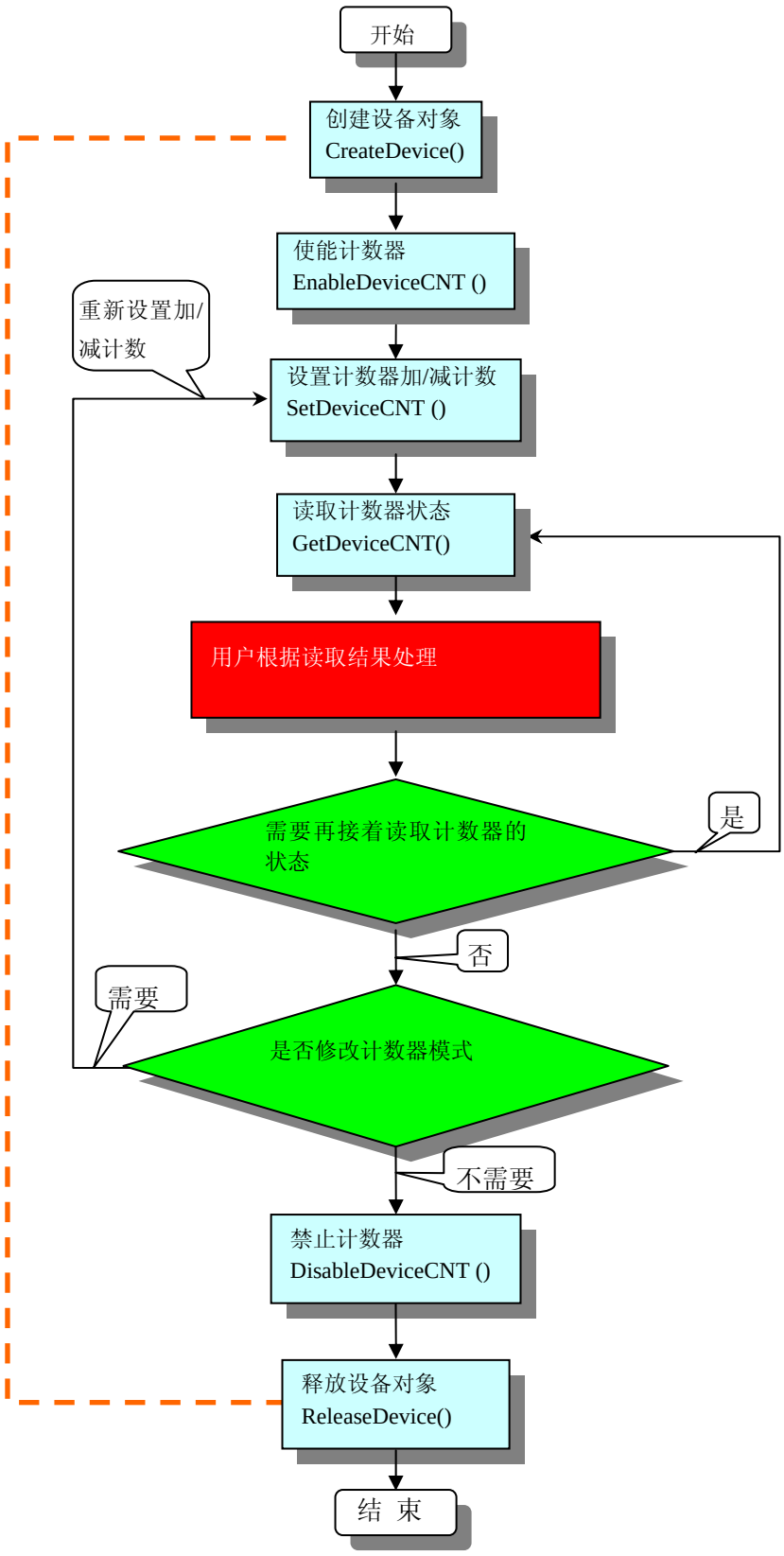



图 2.1 读取计数器过程

## 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“USB5529_”）

**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) | |
| 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 | |
| GetDeviceCurrentID | 取得设备当前 ID 号 | |
| 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 | |
| ReleaseDevice | 关闭设备，且释放 USB 总线设备对象 | |
| *② 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 |
| RetDeviceDO | 回读数字量输出状态 | 上层用户 |
| *③ CNT 计数与定时器操作函数* | | |
| EnableDeviceCNT | 使能计数器 | |
| SetDeviceCNT | 设置加计数或减计数 | |
| GetDeviceCNT | 取得所有计数器的状态信息 | |
| DisableDeviceCNT | 禁止计数器 | |

**使用需知**

***Visual C++：***

首先将 USB5529.h 和 USB5529.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(USB5529.lib)加入到您的工程中。

#include “USB5529.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 USB5529.h 和 USB5529.lib 两个文件均可在演示程序文件夹下面找到。

***Visual Basic:***

首先将 USB5529.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，在弹出的对话中选择 USB5529.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 程序均是需要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：

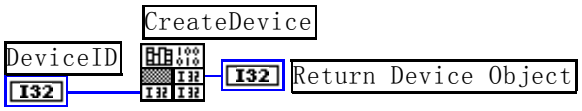
***Visual C++：***

HANDLE CreateDevice(int DeviceLgcID = 0)

***Visual Basic：***

Declare Function CreateDevice Lib "USB5529" (Optional ByVal DeviceLgcID As Integer = 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 USB5529 AD 模板时，系统则以“USB5529”作为基本名称，再以 DeviceLgcID 的初值组合成该设备的标识符“USB5529-0”来确认和管理这第一个设备，若用户接着再添加第二个 USB5529 AD 模板时，则系统将以“USB5529-1”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 "USB5529" (ByVal DevicePhysID As Integer = 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 USB5529 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，请阅读下以内容。

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==INVALIDE_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 "USB5529" (ByVal hDevice As Long ) As Integer

***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GetLastErrorEx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 "USB5529" (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 USB5529 AD 模板时，系统则以“USB5529”作为基本名称，再以 DeviceLgcID 的初值组合成该设备的标识符“USB5529-0”来确认和管理这第一个设备，若用户接着再添加第二个 USB5529 AD 模板时，则系统将以“USB5529-1”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 号能让用户准确的辩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 USB5529 设备各种配置信息**

函数原型：

***Visual C++:***

BOOL ListDeviceDlg (void)

***Visual Basic***:

Declare Function ListDeviceDlg Lib "USB5529" ( )As Boolean

***LabVIEW:***

请参考相关演示程序。

功能：列表系统中 USB5529 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 USB5529 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 "USB5529" (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：

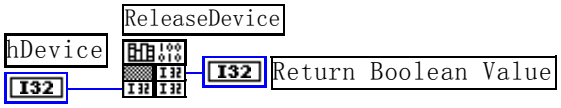
***Visual C++:***

BOOL ReleaseDevice(HANDLE hDevice)

***Visual Basic:***

Declare Function ReleaseDevice Lib "USB5529" (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、CNT 计数与定时器操作函数原型说明

- **使能计数器**

函数原型：

***Visual C++:***

```
BOOL EnableDeviceCNT (HANDLE hDevice)
```

***Visual Basic:***

```
Declare Function EnableDeviceCNT Lib "USB5529" (ByVal hDevice As Long) As Boolean
```

***LabVIEW:***

请参考相关演示程序。

功能：使能计数器。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice DisableDeviceCNT ReleaseDevice

- **设置加计数或减计数**

函数原型：

***Visual C++:***

```
BOOL SetDeviceCNT(HANDLE hDevice,
                  UINT CountMode)
```

***Visual Basic:***

```
Declare Function SetDeviceCNT Lib "USB5529" (ByVal hDevice As Long, _
                                             ByVal CountMode As Long) As Boolean
```

***LabVIEW:***

请参考相关演示程序。

功能：设置加计数或减计数。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

CountMode 方式控制字。其选项值如下表：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| USB5529_CNT_MODE_DEC | 0x0000 | 减计数 |
| USB5529_CNT_MODE_ADD | 0x0001 | 加计数 |

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice GetDeviceCNT ReleaseDevice

- **取得所有计数器的状态信息**

函数原型：

***Visual C++:***

BOOL GetDeviceCNT (HANDLE hDevice,
PLONG CNTValue)

***Visual Basic:***

Declare Function GetDeviceCNT Lib "USB5529" (ByVal hDevice As Long, _
ByRef CNTValue As Long) As Boolean

***LabVIEW:***

请参考相关演示程序。

功能：取得所有计数器的状态信息。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

CNTValue 所有计数器的状态信息。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice SetDeviceCNT ReleaseDevice

- **禁止计数器**

函数原型：

***Visual C++:***

BOOL DisableDeviceCNT (HANDLE hDevice)

***Visual Basic:***

Declare Function DisableDeviceCNT Lib "USB5529" (ByVal hDevice As Long) As Boolean

***Delphi:***

Function DisableDeviceCNT (hDevice : Integer) : Boolean;
StdCall; External 'USB5529' Name ' DisableDeviceCNT ';

***LabVIEW:***

请参考相关演示程序。

功能：禁止计数器。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

返回值：若成功，返回 TRUE，否则返回 FALSE。

**相关函数：** CreateDevice EnableDeviceCNT ReleaseDevice

## 第四节、DIO 数字开关量输入输出简易操作函数原型说明

- **八路开关量输入**

函数原型：

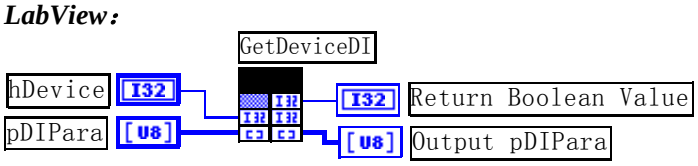
***Visual C++:***

BOOL GetDeviceDI (HANDLE hDevice,
BYTE bDISts[8])

***Visual Basic:***

Declare Function GetDeviceDI Lib "USB5529" (ByVal hDevice As Long, _
ByVal bDISts(0 to 7 ) 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八路开关量输入状态的参数结构，共有 8 个成员变量，分别对应于DI0～DI7 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

◆ **八路开关量输出**

函数原型：

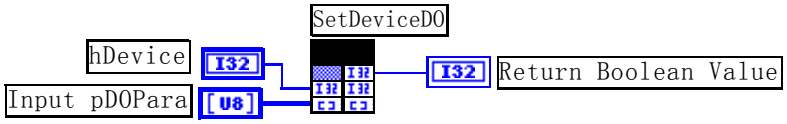
***Visual C++:***

```
BOOL SetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[8])
```

***Visual Basic:***

```
Declare Function SetDeviceDO Lib "USB5529" ( ByVal hDevice As Long, _
                                             ByVal bDOSts(0 to 7 )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。

bDOSts 八路开关量输出状态的参数结构，共有 8 个成员变量，分别对应于DO0～DO7 路开关量输出状态位。比如置bDOSts[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DO7 共 8 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

◆ **回读数字量输出状态**

函数原型：

***Visual C++:***

```
BOOL RetDeviceDO (HANDLE hDevice,
                  BYTE bDOSts[8])
```

***Visual Basic:***

```
Declare Function RetDeviceDOLib "USB5529" (ByVal hDevice As Long, _
                                           ByVal bDOSts(0 to 7) As Byte) As Boolean
```

***LabVIEW:***

请参考相关演示程序。

功能：负责将 USB 设备上的输出开关量置成由 bDOSts[x]指定的相应状态。

参数：

hDevice设备对象句柄，它应由CreateDevice创建。

bDOSts 获得开关输出状态(注意：必须定义为 8 个字节元素的数组)。

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 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SetDeviceCNT函数进行计数操作**

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 USB5529.h 和 Sys.cpp)。

**[程序]**丨**[阿尔泰测控演示系统]**丨**[USB5529 8 路开关量和 1 路计数器卡]**丨**[Microsoft Visual C++]**丨**[简易代码演示]**丨**[计数器演示源程序]**

**其简易程序默认存放路径为：系统盘\ART\USB5529\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 USB5529.h 和 Sys.cpp)。

**[程序]**丨**[阿尔泰测控演示系统]**丨**[USB5529 8 路开关量和 1 路计数器卡]**丨 **[Microsoft Visual C++]**丨**[简易代码演示]**丨**[开关量演示源程序]**

**其默认存放路径为：系统盘\ART\USB5529\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 USB5529.h 和 DADoc.cpp)。

**[程序]**丨**[阿尔泰测控演示系统]**丨**[USB5529 8 路开关量和 1 路计数器卡]**丨**[Microsoft Visual C++]**丨**[高级代码演示]**

**其默认存放路径为：系统盘\ART\USB5529\SAMPLES\VC\ADVANCED**

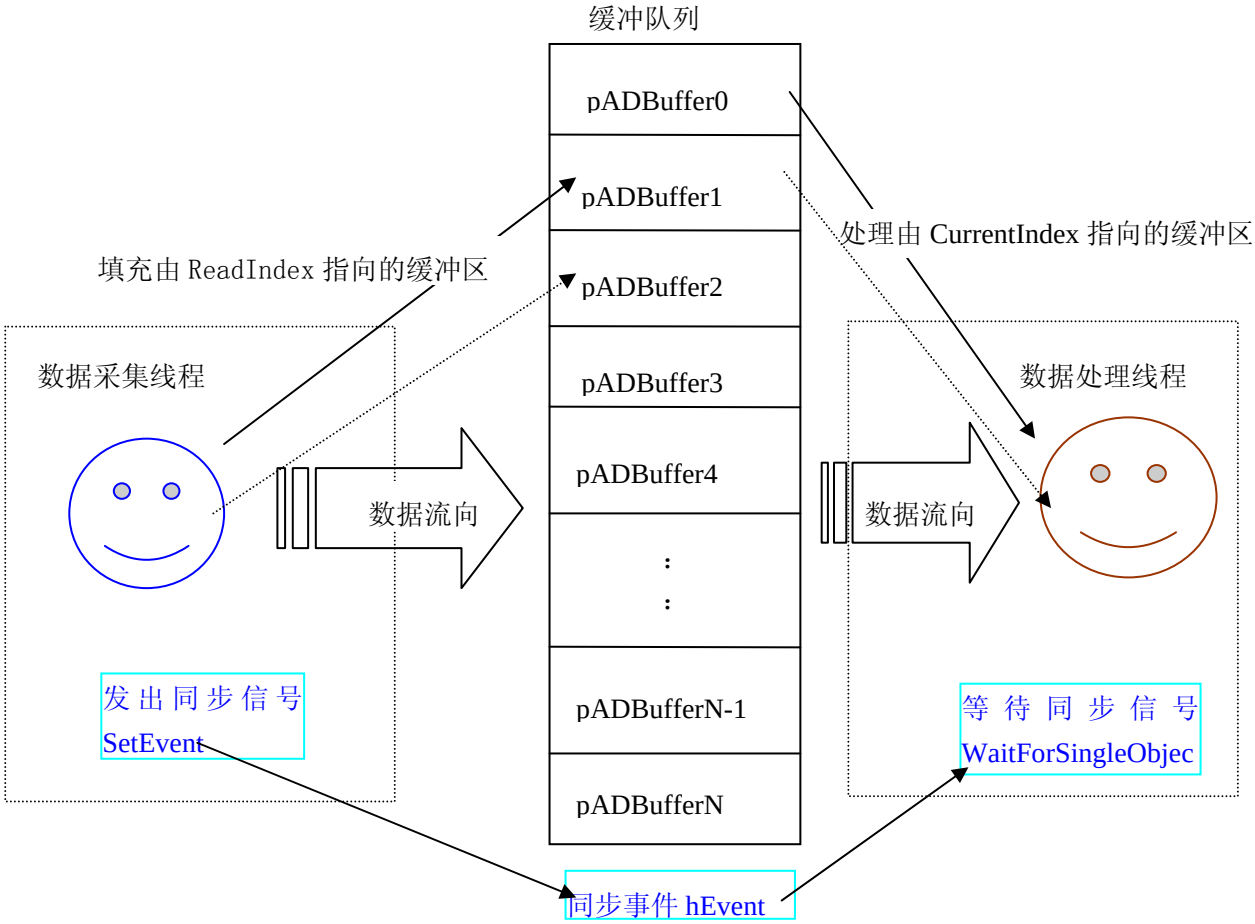
**其他语言的演示可以用上面类似的方法找到。**

# 第五章　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中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两维数组如 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面被堵塞，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]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[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]丨[阿尔泰测控演示系统]丨[Microsoft Visual C++]丨[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
```

# 第六章 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“USB5529_”）

| 函数名 | 函数功能 | 备注 |
|---|---|---|
| ***① 创建 Visual Basic 子线程，线程数量可达 32 个以上*** | | |
| CreateVBThread | 在 VB 环境中建立子线程对象 | 在 VB 中可实现多线程 |

| | | |
|---|---|---|
| TerminateVBThread | 终止 VB 的子线程 | |
| 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 | |
| ***② 文件对象操作函数*** | | |
| GetDevVersion | 获取设备固件及程序版本 | |
| CreateFileObject | 初始设备文件对象 | |
| WriteFile | 请求文件对象写用户数据到磁盘文件 | |
| ReadFile | 请求文件对象读数据到用户空间 | |
| SetFileOffset | 设置文件指针偏移 | |
| GetFileLength | 取得文件长度 | |
| ReleaseFile | 释放已有的文件对象 | |
| GetDiskFreeBytes | 取得指定磁盘的可用空间(字节) | 适用于所有设备 |
| ***③ 各种参数保存和读取函数*** | | |
| SaveParaInt | 保存整型参数到注册表 | |
| LoadParaInt | 从注册表中读取整型参数值 | |
| SaveParaString | 保存字符参数到注册表 | |
| LoadParaString | 从注册表中读取字符参数值 | |
| ***④ 其他函数*** | | |
| kbhit | 探测用户是否有击键动作 | |
| getch | 等待并获取用户击键值 | |
| GetLastErrorEx | 取得驱动函数错误信息 | |

## 第二节、线程操作函数原型说明

（如果您的 VB6.0 中线程无法正常运行，可能是 VB6.0 语言本身的问题，请选用 VB5.0）

♦ **在 VB 环境中，创建子线程对象，以实现多线程操作**

***Visual C++:***

```
BOOL CreateVBThread(HANDLE *hThread,
                    LPTHREAD_START_ROUTINE RoutineAddr)
```

***Visual Basic：***

```
Declare Function CreateVBThread Lib "USB5529" (ByRef hThread As Long, _
                                               ByVal RoutineAddr As Long ) As Boolean
```

功能：该函数在 VB 环境中解决了不能实现或不能很好地实现多线程的问题。通过该函数用户可以很轻松地实现多线程操作。

参数：

hThread 若成功创建子线程，该参数将返回所创建的子线程的句柄，当用户操作这个子线程时将用到这个句柄，如启动线程、暂停线程以及删除线程等。

RoutineAddr 作为子线程运行的函数的地址，在实际使用时，请用AddressOf关键字取得该子线程函数的地址，再传递给CreateVBThread函数。

返回值：当成功创建子线程时，返回TRUE，且所创建的子线程为挂起状态，用户需要用Win32 API函数ResumeThread函数启动它。若失败，则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

**相关函数：** CreateVBThread TerminateVBThread

**注意: RoutineAddr 指向的函数或过程必须放在 VB 的模块文件中，如 USB5529.Bas 文件中。**

***Visual Basic 程序举例:***

```
' 在模块文件中定义子线程函数(注意参考实例)
Function NewRoutine() As Long        ' 定义子线程函数
        :                            ' 线程运行代码
NewRoutine = 1   ' 返回成功码
End Function
'-------------------------------------------------------------------
' 在窗体文件中创建子线程
      :
Dim hNewThread As Long
    If Not CreateVBThread(hNewThread, AddressOf NewRoutine) Then ' 创建新子线程
        MsgBox "创建子线程失败"
        Exit Sub
    End If
    ResumeThread (hNewThread)   '启动新线程          :
     :
```

♦ **在 VB 中，删除子线程对象**

***Visual C++:***

BOOL TerminateVBThread(HANDLE hThread)

***Visual Basic:***

Declare Function TerminateVBThread Lib "USB5529" (ByVal hThread As Long) As Boolean

功能：在VB中删除由CreateVBThread创建的子线程对象。

参数：hThread 指向需要删除的子线程对象的句柄，它应由CreateVBThread创建。

返回值：当成功删除子线程对象时，返回TRUE，否则返回FALSE，用户可用GetLastErrorEx捕获当前错误码。

**相关函数：** CreateVBThread TerminateVBThread

***Visual Basic 程序举例:***

```
      :
If Not TerminateVBThread (hNewThread)   ' 终止子线程
    MsgBox "创建子线程失败"
    Exit Sub
End If
      :
```

♦ **创建内核系统事件**

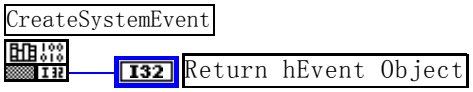
***Visual C++:***

HANDLE CreateSystemEvent(void)

***Visual Basic:***

Declare Function CreateSystemEvent Lib " USB5529 " ( ) 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 " USB5529 " (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。

## 第三节、文件对象操作函数原型说明

◆ **创建文件对象**

函数原型：

***Visual C++:***

```
HANDLE CreateFileObject (HANDLE hDevice,
                         LPCTSTR strFileName,
                         int Mode)
```

***Visual Basic:***

```
Declare Function CreateFileObject Lib "USB5529" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer) As Long
```

***LabVIEW:***

请参见相关演示程序。

功能：初始化设备文件对象， 以期待 WriteFile 请求准备文件对象进行文件操作。
参数：
hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。
strFileName 与新文件对象关联的磁盘文件名，可以包括盘符和路径等信息。在 C 语言中，其语法格式如：“C:\\USB5529\\Data.Dat”，在 Basic 中，其语法格式如：“C:\USB5529\Data.Dat”。
Mode　文件操作方式，所用的文件操作方式控制字定义如下(可通过或指令实现多种方式并操作)：

| 常量名 | 常量值 | 功能定义 |
|---|---|---|
| USB5529_modeRead | 0x0000 | 只读文件方式 |
| USB5529_modeWrite | 0x0001 | 只写文件方式 |
| USB5529_modeReadWrite | 0x0002 | 既读又写文件方式 |
| USB5529_modeCreate | 0x1000 | 如果文件不存在可以创建该文件， 如果存在，则重建此文件，且清 0 |

返回值：若成功，则返回文件对象句柄。
**相关函数：** CreateDevice　CreateFileObject　WriteFile

ReadFile ReleaseFile ReleaseDevice

♦ **通过设备对象，往指定磁盘上写入用户空间的采样数据**

函数原型：

***Visual C++:***

```
BOOL WriteFile(HANDLE hFileObject,
               PVOID pDataBuffer,
               LONG nWriteSizeBytes)
```

***Visual Basic:***

```
Declare Function WriteFile Lib "USB5529" (ByVal hFileObject As Long,_
                                          ByRef pDataBuffer As Byte,_
                                          ByVal nWriteSizeBytes As Long) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：通过向设备对象发送“写磁盘消息”，设备对象便会以最快的速度完成写操作。注意为了保证写入的数据是可用的，这个操作将与用户程序保持同步，但与设备对象中的环形内存池操作保持异步，以得到更高的数据吞吐量，其文件名及路径应由CreateFileObject函数中的strFileName指定。

参数：

hFileObject 设备对象句柄，它应由CreateFileObject创建。

pDataBuffer 用户数据空间地址，可以是用户分配的数组空间。

nWriteSizeBytes 告诉设备对象往磁盘上一次写入数据的长度(以字节为单位)。

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：** CreateFileObject WriteFile ReadFile ReleaseFile

♦ **通过设备对象，从指定磁盘文件中读采样数据**

函数原型：

***Visual C++:***

```
BOOL ReadFile( HANDLE hFileObject,
               PVOID pDataBuffer,
               LONG OffsetBytes,
               LONG nReadSizeBytes)
```

***Visual Basic:***

```
Declare Function ReadFile Lib "USB5529" ( ByVal hFileObject As Long, _
                                          ByRef pDataBuffer As Integer, _
                                          ByVal OffsetBytes As Long, _
                                          ByVal nReadSizeBytes As Long) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：将磁盘数据从指定文件中读入用户内存空间中，其访问方式可由用户在创建文件对象时指定。

参数：

hFileObject 设备对象句柄，它应由CreateFileObject创建。

pDataBuffer 用于接受文件数据的用户缓冲区指针，可以是用户分配的数组空间。

OffsetBytes 指定从文件开始端所偏移的读位置。

nReadSizeBytes 告诉设备对象从磁盘上一次读入数据的长度(以字为单位)。

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：** CreateFileObject WriteFile ReadFile ReleaseFile

- **设置文件偏移位置**

函数原型：

***Visual C++:***

```
BOOL SetFileOffset (HANDLE hFileObject,
                    LONG nOffsetBytes)
```

***Visual Basic:***

```
Declare Function SetFileOffset Lib "USB5529" (ByVal hFileObject As Long,
                                              ByVal nOffsetBytes As Long) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：设置文件偏移位置，用它可以定位读写起点。

参数：hFileObject 文件对象句柄，它应由CreateFileObject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：** CreateFileObject WriteFile ReadFile ReleaseFile

- **取得文件长度（字节）**

函数原型：

***Visual C++:***

```
ULONG GetFileLength (HANDLE hFileObject)
```

***Visual Basic:***

```
Declare Function GetFileLength Lib "USB5529" (ByVal hFileObject As Long) As Long
```

***LabVIEW:***

详见相关演示程序。

功能：取得文件长度。

参数：hFileObject 设备对象句柄，它应由CreateFileObject创建。

返回值：若成功，则返回>1，否则返回 0，用户可以用GetLastErrorEx捕获错误码。

**相关函数：** CreateFileObject WriteFile ReadFile ReleaseFile

- **释放设备文件对象**

函数原型：

***Visual C++:***

```
BOOL ReleaseFile(HANDLE hFileObject)
```

***Visual Basic:***

```
Declare Function ReleaseFile Lib "USB5529" (ByVal hFileObject As Long) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：释放设备文件对象。

参数：hFileObject 设备对象句柄，它应由CreateFileObject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：** CreateFileObject WriteFile ReadFile ReleaseFile

◆ **取得指定磁盘的可用空间**

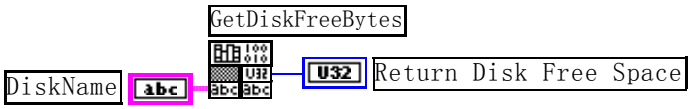
***Visual C++:***

```
LONGLONG GetDiskFreeBytes(LPCTSTR strDiskName )
```

***Visual Basic:***

```
Declare Function GetDiskFreeBytes Lib "USB5529" (ByVal strDiskName As String ) As Currency
```

***LabVIEW:***




功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：strDiskName 需要访问的盘符，若为 C 盘为"C:\\"， D 盘为"D:\\"，以此类推。

返回值：若成功，返回大于或等于 0 的长整型值，否则返回零值，用户可用GetLastErrorEx捕获错误码。注意使用 64 位整型变量。

## 第四节、各种参数保存和读取函数原型说明

◆ **将整型变量的参数值保存在系统注册表中**

函数原型：

***Visual C++:***

```
BOOL SaveParaInt( HANDLE hDevice,
                  LPCTSTR strParaName,
                  int nValue)
```

***Visual Basic:***

```
Declare Function SaveParaInt Lib "USB5529" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nValue As Integer) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB5529\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nValue 整型参数值。它保存在由 strParaName 命名的键项里。

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：** SaveParaInt LoadParaInt SaveParaString LoadParaString

◆ **将整型变量的参数值从系统注册表中读出**

函数原型：

***Visual C++:***

```
UINT LoadParaInt( HANDLE hDevice,
                  LPCTSTR strParaName,
                  int nDefaultVal)
```

***Visual Basic:***

```
Declare Function LoadParaInt Lib "USB5529" (ByVal hDevice As Long,_
                                            ByVal strParaName As String,_
                                            ByVal nDefaultVal As Integer) As Long
```

***LabVIEW:***

详见相关演示程序。

功能：将整型变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB5529\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

nDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

返回值：若指定的整型参数项存在，则返回其整型值。否则返回由 nDefaultVal 指定的默认值。

**相关函数：** SaveParaInt LoadParaInt SaveParaString LoadParaString

**♦将字符变量的参数值保存在系统注册表中**

函数原型：

***Visual C++:***

```
BOOL SaveParaString ( HANDLE hDevice,
                      LPCTSTR strParaName,
                      LPCTSTR strParaVal)
```

***Visual Basic:***

```
Declare Function SaveParaString Lib "USB5529" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：将整型变量的参数值保存在系统注册表中。具体保存位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB5529\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 整型参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 字符参数值。它保存在由 strParaName 命名的键项里。

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：** SaveParaInt LoadParaInt SaveParaString LoadParaString

**♦将字符变量的参数值从系统注册表中读出**

函数原型：

***Visual C++:***

```
BOOL LoadParaString ( HANDLE hDevice,
                      LPCTSTR strParaName,
                      LPCTSTR strParaVal,
                      LPCTSTR strDefaultVal)
```

***Visual Basic:***

```
Declare Function LoadParaString Lib "USB5529" (ByVal hDevice As Long,_
                                               ByVal strParaName As String,_
                                               ByVal strParaVal As String,_
                                               ByVal strDefaultVal As String) As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：将字符变量的参数值从系统注册表中读出。读出参数值的具体位置视设备逻辑号而定。如逻辑号为“0”的其他参数保存位置为：HKEY_CURRENT_USER\Software\Art\USB5529\Device-0\Others。

参数：

hDevice设备对象句柄，它应由CreateDevice或CreateDeviceEx创建。

strParaName 字符参数字符名。它指名该参数在注册表中的字符键项。

strParaVal 取得 strParaName 指定的键项的字符值。

strDefaultVal 若 strParaName 指定的键项不存在，则由该参数指定的默认值返回。

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：** SaveParaInt LoadParaInt SaveParaString LoadParaString

## 第五节、其他函数原型说明

- **探测用户是否有按键动作**

函数原型：

***Visual C++:***

```
BOOL kbhit (void)
```

***Visual Basic:***

```
Declare Function kbhit Lib "USB5529" () As Boolean
```

***LabVIEW:***

详见相关演示程序。

功能：探测用户是否用键盘按键动作，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若自上次探测过后，若用户有键盘按键动作，则返回 TRUE，否则返回 FALSE。

**相关函数：** getch kbhit

- **等待按键动作并返回按键值**

函数原型：

***Visual C++:***

```
char getch (void)
```

***Visual Basic:***

```
Declare Function getch Lib "USB5529" () As String
```

***LabVIEW:***

详见相关演示程序。

功能：探等待用户键盘按键并以字符方式返回按键值，主要应在基于 VB、DELPHI 等控制台应用程序中。

参数：无。

返回值：若用户没有按键动作，此函数一直不返回，一旦用户有按键动作，便立即返回，且返回其当前按键值(ACII 码)。

**相关函数：** getch kbhit

**♦怎样获取驱动函数错误信息**

函数原型：

***Visual C++:***

```
DWORD GetLastErrorEx (LPCTSTR strFuncName,
                      LPCTSTR strErrorMsg)
```

***Visual Basic:***

```
Declare Function GetLastErrorEx Lib "USB5529" (ByVal strFuncName As String,_
                                               ByVal strErrorMsg As String) As Long
```

***LabVIEW:***

详见相关演示程序。

功能：将当某个驱动函数出错时，可以调用此函数获得具体的错误和错误信息字串。

参数：

strFuncName 出错函数的名称。注意此函数必须是完整名称，如 AD 初始化函数 USB5529_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“USB5529_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串。

返回值：返回错误码。

**相关函数：** 无。