

# PCI2307 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

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、使用上层用户函数，高效、简单	2
第二节、如何管理 PCI 设备	2
第三节、如何实现开关量的简便操作	2
第四节、哪些函数对您不是必须的	2
第三章 PCI 即插即用设备操作函数接口介绍	3
第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI2307_”）	4
第二节、设备对象管理函数原型说明	5
第三节、中断操作函数原型说明	7
第四节、DIO 数字量输入输出开关量操作函数原型说明	8
第四章 上层用户函数接口应用实例	9
第一节、怎样使用GetDeviceDI函数进行更便捷的数字开关量输入操作	9
第二节、怎样使用 SetDeviceDO 函数进行更便捷的数字开关量输出操作	9
第五章 共用函数介绍	9
第一节、公用接口函数总列表（每个函数省略了前缀“PCI2307_”）	9
第二节、IO 端口读写函数原型说明	9
第三节、线程操作函数原型说明	13
第四节、文件对象操作函数原型说明	15

## 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阿尔泰科技发展有限公司所有，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、我公司授权的代理商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。若您需要我公司产品及相关信息请及时与当地代理商联系或直接与我们联系，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 PCIxxxx\_ 则被省略。如 PCI2307\_CreateDevice 则写为 CreateDevice。

二、函数名及参数中各种关键字缩写

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			

## 第二章 使用纲要

### 第一节、使用上层用户函数，高效、简单

如果您只关心通道及频率等基本参数，而不必了解复杂的硬件知识和控制细节，那么我们强烈建议您使用上层用户函数，它们就是几个简单的形如 Win32 API 的函数，具有相当的灵活性、可靠性和高效性。诸如 [SetDeviceDO](#) 等。而底层用户函数如 [WritePortByte](#)、[ReadPortByte](#)……则是满足了解硬件知识和控制细节、且又需要特殊复杂控制的用户。但不管怎样，我们强烈建议您使用上层函数（在这些函数中，您见不到任何设备地址、寄存器端口、中断号等物理信息，其复杂的控制细节完全封装在上层用户函数中。）对于上层用户函数的使用，您基本上不必参考硬件说明书，除非您需要知道板上 D 型插座等管脚分配情况。

### 第二节、如何管理 PCI 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绝对控制权。然后将此句柄作为参数传递给相应的驱动函数，如 [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

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。如果您使用上层用户函数访问设备，那么 [GetDeviceAddr](#) 等函数您可完全不必理会，除非您是作为底层用户管

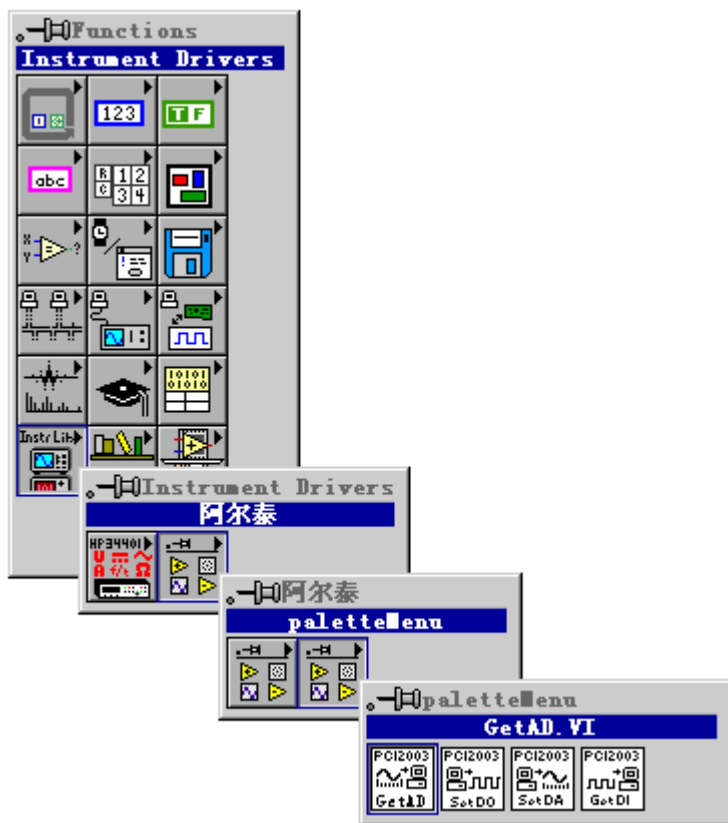
理设备。而[WritePortByte](#)、[WritePortWord](#)、[WritePortULong](#)、[ReadPortByte](#)、[ReadPortWord](#)、[ReadPortULong](#)则对PCI用户来讲，可以说完全是辅助性，它们只是对我公司驱动程序的一种功能补充，对用户额外提供的，它们可以帮助您在NT、Win2000等操作系统中实现对您原有传统设备如ISA卡、串口卡、并口卡的访问，而没有这些函数，您可能在基于Windows NT架构的操作系统中无法继续使用您原有的老设备。

### 第三章 PCI 即插即用设备操作函数接口介绍

由于我公司的设备应用于各种不同的领域，有些用户可能根本不关心硬件设备的控制细节，只关心AD的首末通道、采样频率等，然后就能通过一两个简易的采集函数便能轻松得到所需要的AD数据。这方面的用户我们称之为上层用户。那么还有一部分用户不仅对硬件控制熟悉，而且由于应用对象的特殊要求，则要直接控制设备的每一个端口，这是一种复杂的工作，但又是必须的工作，我们则把这一群用户称之为底层用户。因此总的看来，上层用户要求简单、快捷，他们最希望在软件操作上所面对的全是他们最关心的问题，比如在正式采集数据之前，只须用户调用一个简易的初始化函数（告诉设备我要使用多少个通道，采样频率是多少赫兹等，然后便可以用函数指定每次采集的点数，即可实现数据连续不间断采样。而关于设备的物理地址、端口分配及功能定义等复杂的硬件信息则与上层用户无任何关系。那么对于底层用户则不然。他们不仅要关心设备的物理地址，还要关心虚拟地址、端口寄存器的功能分配，甚至每个端口的Bit位都要了如指掌，看起来这是一项相当复杂、繁琐的工作。但是这些底层用户一旦使用我们提供的技术支持，则不仅可以让您不必熟悉PCI总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，比如您不用去了解PCI的资源配置空间、PNP即插即用管理，而只须用[GetDeviceAddr](#)函数便可以同时取得指定设备的物理基地址和虚拟线性基地址。这个时候您便可以用这个虚拟线性基地址，再根据硬件使用说明书中的各端口寄存器的功能说明，然后对这些端口寄存器进行32位模式的读写操作，即可实现设备的所有控制。

综上所述，用户使用我公司提供的驱动程序软件包将极大的方便和满足您的各种需求。但为了您更省心，别忘了在您正式阅读下面的函数说明时，先明白自己是上层用户还是底层用户，因为在《[设备驱动接口函数总列表](#)》中的备注栏里明确注明了适用对象。

另外需要申明的是，在本章和下一章中列明的关于LabView的接口，均属于外挂式驱动接口，他是通过LabView的Call Library Function功能模板实现的。它的特点是除了自身的语法略有不同以外，每一个基于LabView的驱动图标与Visual C++、Visual Basic、Delphi等语言中每个驱动函数是一一对应的，其调用流程和功能是完全相同的。那么相对于外挂式驱动接口的另一种方式是内嵌式驱动。这种驱动是完全作为LabView编程环境中的紧密耦合的一部分，它可以直接从LabView的Functions模板中取得，如下图所示。此种方式更适合上层用户的需要，它的最大特点是方便、快捷、简单，而且可以取得它的在线帮助。关于LabView的外挂式驱动和内嵌式驱动更详细的叙述，请参考LabView的相关演示。



LabView 内嵌式驱动接口的获取方法

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PCI2307\_”）

函数名	函数功能	备注
<b>① 设备对象操作函数</b>		
<a href="#">CreateDevice</a>	创建 PCI 设备对象(用设备逻辑号)	上层及底层用户
<a href="#">GetDeviceCurrentID</a>	取得指定设备的逻辑 ID 和物理 ID	上层及底层用户
<a href="#">GetDeviceCount</a>	取得同一种 PCI 设备的总台数	上层及底层用户
<a href="#">ListDeviceDlg</a>	列表所有同一种 PCI 设备的各种配置	上层及底层用户
<a href="#">ReleaseDevice</a>	关闭设备，且释放 PCI 总线设备对象	上层及底层用户
<b>② 中断操作函数</b>		
<a href="#">InitDeviceInt</a>	初始化中断	
<a href="#">GetDeviceIntCount</a>	在中断初始化后,取得中断服务程序产生的次数	
<a href="#">ReleaseDeviceInt</a>	释放中断资源	
<b>③ DIO 开关量简易操作函数</b>		
<a href="#">GetDeviceDI</a>	开关输入函数	上层用户
<a href="#">SetDeviceDO</a>	开关输出函数	上层用户

使用需知:

Visual C++:

要使用如下函数关键的问题是:

首先，必须在您的源程序中包含如下语句:

```
#include "C:\Art\PCI2307\INCLUDE\PCI2307.H"
```

注：以上语句采用默认路径和默认板号，应根据您的板号和安装情况确定 PCI2307.H 文件的正确路径，当然也可以把此文件拷到您的源程序目录中。然后加入如下语句:

```
#include "PCI2307.H"
```

另外，要在 VB 环境中用子线程以实现高速、连续数据采集与存盘，请务必使用 VB5.0 版本。当然如果您有 VB6.0 的最新版，也可以实现子线程操作。

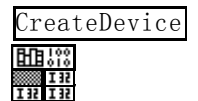
### Visual Basic:

要使用如下函数一个关键的问题是首先必须将我们提供的模块文件(\*.Bas)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 PCI2307.Bas 模块文件, 该文件的路径为用户安装驱动程序后其子目录 Samples\VB 下面。

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, 我们不能保证完全顺利运行。

### 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 的进一步介绍请见本文最后一部分关于 LabView 的专述。其驱动程序接口单元模块的使用方法如下:



- 一、在 LabView 中打开 PCI2307.VI 文件, 用鼠标单击接口单元图标, 比如 CreateDevice 图标  
然后按 Ctrl+C 或选择 LabView 菜单 Edit 中的 Copy 命令, 接着进入用户的应用程序 LabView 中, 按 Ctrl+V 或选择 LabView 菜单 Edit 中的 Paste 命令, 即可将接口单元加入到用户工程中, 然后按以下函数原型说明或演示程序的说明连接该接口模块即可顺利使用。
- 二、根据 LabView 语言本身的规定, 接口单元图标以黑色的较粗的中间线为中心, 以左边的方格为数据输入端, 右边的方格为数据的输出端, 设备对象句柄、用户分配的数据缓冲区、要求采集的数据长度等信息从接口单元左边输入端进入单元, 待单元接口被执行后, 需要返回给用户的数据从接口单元右边的输出端输出, 其他接口完全同理。
- 三、在单元接口图标中, 凡标有 "I32" 为有符号长整型 32 位数据类型, "U16" 为无符号短整型 16 位数据类型, "[U16]" 为无符号 16 位短整型数组或缓冲区或指针, "[U32]" 与 "[U16]" 同理, 只是位数不一样。

## 第二节、设备对象管理函数原型说明

### ◆ 创建设备对象函数 (逻辑号)

函数原型:

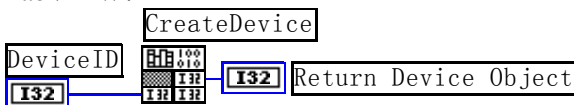
**Visual C++:**

`HANDLE CreateDevice (int DeviceID)`

**Visual Basic:**

`Declare Function CreateDevice Lib "PCI2307" (ByVal DeviceID As Integer) As Long`

**LabVIEW:**



**功能:** 该函数使用逻辑号创建设备对象, 并返回其设备对象句柄 hDevice。只有成功获取 hDevice, 您才能实现对该设备所有功能的访问。

**参数:** DeviceID 设备 ID 标识号。

**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:** [CreateDevice](#)      [GetDeviceCount](#)      [ListDeviceDlg](#)  
[ReleaseDevice](#)

### Visual C++ 程序举例:

```

HANDLE hDevice; // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = CreateDevice ( DeviceLgcID ); // 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
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = CreateDevice ( DeviceLgcID ) ' 创建设备对象,并取得设备对象句柄
If hDevice = INVALID_HANDLE_VALUE Then ' 判断设备对象句柄是否有效
    MsgBox "创建设备对象失败"
    Exit Sub ' 退出该过程
End If
:

```

◆ 取得本计算机系统中 PCI2307 设备的总数量

函数原型:

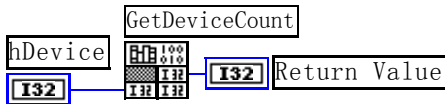
**Visual C++:**

```
int GetDeviceCount (HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function GetDeviceCount Lib "PCI2307" (ByVal hDevice As Long ) As Integer
```

**LabVIEW:**



功能: 取得 PCI2307 设备的数量。

参数: hDevice设备对象句柄, 它应由CreateDevice创建。

返回值: 返回系统中 PCI2307 的数量。

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

◆ 用对话框控件列表计算机系统中所有 PCI2307 设备各种配置信息

函数原型:

**Visual C++:**

```
BOOL ListDeviceDlg (HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function ListDeviceDlg Lib "PCI2307" (ByVal hDevice As Long ) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

功能: 列表系统中 PCI2307 的硬件配置信息。

参数: hDevice设备对象句柄, 它应由CreateDevice创建。

返回值: 若成功, 则弹出对话框控件列表所有 PCI2307 设备的配置情况。

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:

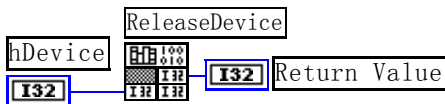
**Visual C++:**

```
BOOL ReleaseDevice(HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function ReleaseDevice Lib "PCI2307" (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](#)后, 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DMA控制器、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、中断操作函数原型说明

#### ◆ 初始化中断

函数原型:

**Visual C++:**

```
BOOL InitDeviceInt (HANDLE hDevice,  
                   HANDLE hEventInt)
```

**Visual Basic:**

```
Declare Function InitDeviceInt Lib "PCI2307" (ByVal hDevice As Long, _  
                                             ByVal hEventInt As Long) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 它负责初始化设备对象的硬件中断的方式工作。

**参数:**

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

**hEventInt** 中断事件对象句柄, 它应由[CreateSystemEvent](#)函数创建。它被创建时是一个不发信号且自动复位的内核系统事件对象。当硬件中断发生, 这个内核系统事件被触发。用户应在数据采集子线程中使用WaitForSingleObject这个Win32 函数来接管这个内核系统事件。当中断没有到来时, WaitForSingleObject将使所在线程进入睡眠状态, 此时, 它不同于程序轮询方式, 它并不消耗CPU时间。当hEventInt事件被触发成发信号状态, 那么WaitForSingleObject将唤醒所在线程, 可以工作了, 比如取FIFO中的数据、分析数据等, 且复位该内核系统事件对象, 使其处于不发信号状态, 以便在取完FIFO数据等工作后, 让所在线程再次进入睡眠状态。所以利用中断方式采集数据, 其效率是最高的。

**返回值:** 若成功, 返回TRUE, 否则返回FALSE, 用户可用GetLastErrorEx捕获当前错误码, 并加以分析。

**相关函数:** [CreateDevice](#)            [InitDeviceInt](#)            [GetDeviceIntCount](#)  
[ReleaseDeviceInt](#)            [ReleaseDevice](#)

#### ◆ 取得中断服务程序产生的次数

函数原型:

**Visual C++:**

```
LONG GetDeviceIntCount (HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function GetDeviceIntCount Lib "PCI2307" (ByVal hDevice As Long) As Long
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 在中断初始化后, 用它取得中断服务程序产生的次数。

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

**返回值:** 若成功, 返回取得中断服务程序产生的次数, 否则返回FALSE, 用户可用GetLastErrorEx捕获当前错误码, 并加以分析。

**相关函数:** [CreateDevice](#)            [InitDeviceInt](#)            [GetDeviceIntCount](#)  
[ReleaseDeviceInt](#)            [ReleaseDevice](#)

#### ◆ 释放中断资源

函数原型:

**Visual C++:**

```
BOOL ReleaseDeviceInt (HANDLE hDevice)
```

**Visual Basic:**

```
Declare Function ReleaseDeviceInt Lib "PCI2307" (ByVal hDevice As Long) As Boolean
```



**LabVIEW:**

请参考相关演示程序。

**功能:** 释放中断资源。

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

**返回值:** 若成功, 返回TRUE, 否则返回FALSE。用户可用GetLastErrorEx捕获当前错误码, 并加以分析。

**相关函数:** [CreateDevice](#)      [InitDeviceInt](#)      [GetDeviceIntCount](#)  
[ReleaseDeviceInt](#)      [ReleaseDevice](#)

#### 第四节、DIO 数字量输入输出开关量操作函数原型说明

##### ◆ 开关量输入

函数原型:

**Visual C++:**

```
BOOL GetDeviceDI ( HANDLE hDevice,
                  PPCCI2307_PARA_DI pDIPara)
```

**Visual Basic:**

```
Declare Function GetDeviceDI Lib "PCI2307" ( ByVal hDevice As Long, _
                                           ByVal pDIPara As PCCI2307_PARA_DI) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 负责将 PCI 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](#)创建。

pDIPara 十六路开关量输入状态的参数结构, 共有 16 个元素, 分别对应于 DI0~DI15 路开关量输入状态位。

如果 bDISts[0]等于“1”则表示 0 通道处于开状态, 若为“0”则 0 通道为关状态。其他同理。

**返回值:** 若成功, 返回 TRUE, 其 pPara 中的值有效; 否则返回 FALSE, 其 pPara 中的值无效。

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

##### ◆ 开关量输出

函数原型:

**Visual C++:**

```
BOOL SetDeviceDO (HANDLE hDevice,
                  PPCCI2307_PARA_DO pDOPara)
```

**Visual Basic:**

```
Declare Function SetDeviceDO Lib "PCI2307" (ByVal hDevice As Long, _
                                           ByVal pDOPara As PCCI2307_PARA_DO) As Boolean
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 负责将 PCI 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](#)创建。

pDOPara 十六路开关量输出状态的参数结构, 共有 16 个成员变量, 分别对应于 DO0~DO15 路开关量输出状态位。比如置 pPara->DO0 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 DO15 共 16 个成员变量赋初值, 其值必须为“1”或“0”。

**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。

## 第四章 上层用户函数接口应用实例

### 第一节、怎样使用[GetDeviceDI](#)函数进行更便捷的数字开关量输入操作

**Visual C++:**

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。

[程序] | [阿尔泰测控演示系统] | [PCI2307 继电器、开关量输入及中断卡] | [Microsoft Visual C++] | [简易代码演示]

### 第二节、怎样使用[SetDeviceD0](#)函数进行更便捷的数字开关量输出操作

**Visual C++:**

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。

[程序] | [阿尔泰测控演示系统] | [PCI2307 继电器、开关量输入及中断卡] | [Microsoft Visual C++] | [简易代码演示]

## 第五章 共用函数介绍

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“PCI2307\_”）

函数名	函数功能	备注
<b>① ISA 总线 I/O 端口操作函数</b>		
<a href="#">GetDeviceAddr</a>	取得指定 PCI 设备寄存器操作基地址	底层用户
<a href="#">WritePortByte</a>	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
<a href="#">WritePortWord</a>	以字(16Bit)方式写 I/O 端口	用户程序操作端口
<a href="#">WritePortULong</a>	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
<a href="#">ReadPortByte</a>	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
<a href="#">ReadPortWord</a>	以字(16Bit)方式读 I/O 端口	用户程序操作端口
<a href="#">ReadPortULong</a>	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
<b>② 创建 Visual Basic 子线程，线程数量可达 32 个以上</b>		
<a href="#">CreateVBThread</a>	在 VB 环境中建立子线程对象	
<a href="#">TerminateVBThread</a>	终止 VB 的子线程	
<a href="#">CreateSystemEvent</a>	创建系统内核事件对象	用于线程同步或中断
<a href="#">ReleaseSystemEvent</a>	释放系统内核事件对象	
<a href="#">DelayTimeUs</a>	高效高精度延时函数	不消耗 CPU 时间
<b>③ 文件对象操作函数</b>		
<a href="#">CreateFileObject</a>	初始设备文件对象	
<a href="#">WriteFile</a>	请求文件对象写用户数据到磁盘文件	
<a href="#">ReadFile</a>	请求文件对象读数据到用户空间	
<a href="#">SetFileOffset</a>	设置文件指针偏移	
<a href="#">GetFileLength</a>	取得文件长度	
<a href="#">ReleaseFile</a>	释放已有的文件对象	
<a href="#">GetDiskFreeBytes</a>	取得指定磁盘的可用空间(字节)	适用于所有设备

### 第二节、IO 端口读写函数原型说明

注意：若您想在 WIN2K 系统的 User 模式中直接访问 I/O 端口，那么您可以安装光盘中 ISA\CommUser 目录下的公用驱动，然后调用其中的 WritePortByteEx 或 ReadPortByteEx 等有“Ex”后缀的函数即可。

◆ 取得指定内存映射寄存器的线性地址和物理地址

函数原型:

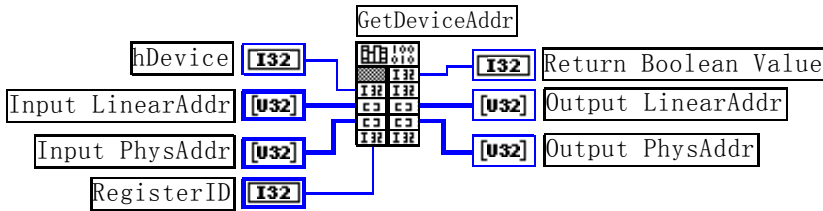
Visual C++:

```
BOOL GetDeviceAddr( HANDLE hDevice,
                   PULONG LinearAddr,
                   PULONG PhysAddr,
                   int RegisterID)
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PCI2307" (ByVal hDevice As Long, _
                                             ByRef LinearAddr As Long, _
                                             ByRef PhysAddr As Long, _
                                             ByVal RegisterID As Integer) As Boolean
```

LabVIEW:



功能: 取得 PCI 设备指定的内存映射寄存器的线性地址。

参数:

hDevice设备对象句柄, 它应由CreateDevice创建。

LinearAddr 指针参数, 用于取得的映射寄存器指向的线性地址, RegisterID 指定的寄存器组属于 MEM 模式时该值不应为零, 也就是说它可用于 WriteRegisterX 或 ReadRegisterX (X 代表 Byte、ULong、Word) 等函数, 以便于访问设备寄存器。它指明该设备位于系统空间的虚拟位置。但如果 RegisterID 指定的寄存器组属于 I/O 模式时该值通常为零, 您不能通过以上函数访问设备。

PhysAddr 指针参数, 用于取得的映射寄存器指向的物理地址, 它指明该设备位于系统空间的物理位置。如果由 RegisterID 指定的寄存器组属于 I/O 模式, 则可用于 WritePortX 或 ReadPortX (X 代表 Byte、ULong、Word) 等函数, 以便于访问设备寄存器。

RegisterID 指定映射寄存器的 ID 号, 其取值范围为[0, 5], 通常情况下, 用户应使用 0 号映射寄存器, 特殊情况下, 我们为用户加以申明。

返回值: 如果执行成功, 则返回TRUE, 它表明由RegisterID指定的映射寄存器的无符号 32 位线性地址和物理地址被正确返回, 否则会返回FALSE, 同时还要检查其LinearAddr和PhysAddr是否为 0, 若为 0 则依然视为失败。用户可用GetLastErrorEx捕获当前错误码, 并加以分析。

- 相关函数: CreateDevice, GetDeviceAddr, WritePortByte, WritePortWord, WritePortULong, ReadPortByte, ReadPortWord, ReadPortULong, ReleaseDevice

Visual C++ 程序举例:

```
:
HANDLE hDevice;
ULONG LinearAddr, PhysAddr;
hDevice = CreateDevice(0);
if(!GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0))
{
    AfxMessageBox("取得设备地址失败...");
}
:
```

Visual Basic 程序举例:

```
:
Dim hDevice As Long
Dim LinearAddr, PhysAddr As Long
hDevice = CreateDevice(0)
if Not GetDeviceAddr(hDevice, LinearAddr, PhysAddr, 0) then
    MsgBox "取得设备地址失败..."
End If
:
```

◆ 以单字节(8Bit)方式写 I/O 端口

函数原型:

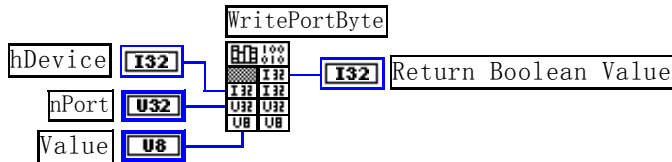
**Visual C++:**

BOOL WritePortByte (HANDLE hDevice,  
                          UINT nPort,  
                          BYTE Value)

**Visual Basic:**

Declare Function WritePortByte Lib "PCI2307" ( ByVal hDevice As Long, \_  
  ByVal nPort As Long, \_  
  ByVal Value As Byte) As Boolean

**LabVIEW:**



功能: 以单字节(8Bit)方式写 I/O 端口。

参数:

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

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值: 若成功, 返回TRUE, 否则返回FALSE, 用户可用GetLastErrorEx捕获当前错误码。

相关函数: [CreateDevice](#)                    [GetDeviceAddr](#)                    [WritePortByte](#)  
              [WritePortWord](#)                    [WritePortULong](#)                    [ReadPortByte](#)  
              [ReadPortWord](#)                    [ReadPortULong](#)                    [ReleaseDevice](#)

◆ 以双字(16Bit)方式写 I/O 端口

函数原型:

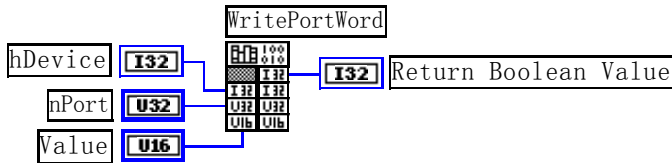
**Visual C++:**

BOOL WritePortWord (HANDLE hDevice,  
                          UINT nPort,  
                          WORD Value)

**Visual Basic:**

Declare Function WritePortWord Lib "PCI2307" ( ByVal hDevice As Long, \_  
  ByVal nPort As Long, \_  
  ByVal Value As Integer) As Boolean

**LabVIEW:**



功能: 以双字(16Bit)方式写 I/O 端口。

参数:

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

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值: 若成功, 返回TRUE, 否则返回FALSE, 用户可用GetLastErrorEx捕获当前错误码。

相关函数: [CreateDevice](#)                    [GetDeviceAddr](#)                    [WritePortByte](#)  
              [WritePortWord](#)                    [WritePortULong](#)                    [ReadPortByte](#)  
              [ReadPortWord](#)                    [ReadPortULong](#)                    [ReleaseDevice](#)

◆ 以四字节(32Bit)方式写 I/O 端口

函数原型:

**Visual C++:**

```

BOOL WritePortULong(HANDLE hDevice,
                    UINT nPort,
                    ULONG Value)

```

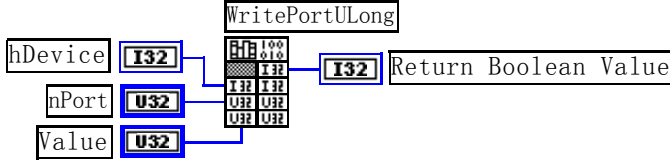
**Visual Basic:**

```

Declare Function WritePortULong Lib "PCI2307" (ByVal hDevice As Long, _
                                                ByVal nPort As Long, _
                                                ByVal Value As Long ) As Boolean

```

**LabVIEW:**



功能: 以四字节(32Bit)方式写 I/O 端口。

参数:

hDevice 设备对象句柄, 它应由>CreateDevice创建。

nPort 设备的 I/O 端口号。

Value 写入由 nPort 指定端口的值。

返回值: 若成功, 返回TRUE, 否则返回FALSE, 用户可用GetLastErrorEx捕获当前错误码。

相关函数: [CreateDevice](#)            [GetDeviceAddr](#)            [WritePortByte](#)  
[WritePortWord](#)            [WritePortULong](#)            [ReadPortByte](#)  
[ReadPortWord](#)            [ReadPortULong](#)            [ReleaseDevice](#)

◆ 以单字节(8Bit)方式读 I/O 端口

函数原型:

**Visual C++:**

```

BYTE ReadPortByte( HANDLE hDevice,
                   UINT nPort)

```

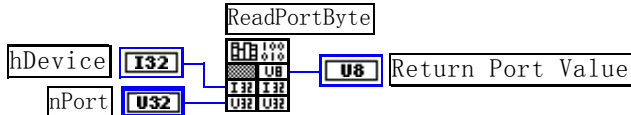
**Visual Basic:**

```

Declare Function ReadPortByte Lib "PCI2307" (ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Byte

```

**LabVIEW:**



功能: 以单字节(8Bit)方式读 I/O 端口。

参数:

hDevice设备对象句柄, 它应由>CreateDevice创建。

nPort 设备的 I/O 端口号。

返回值: 返回由 nPort 指定的端口的值。

相关函数: [CreateDevice](#)            [GetDeviceAddr](#)            [WritePortByte](#)  
[WritePortWord](#)            [WritePortULong](#)            [ReadPortByte](#)  
[ReadPortWord](#)            [ReadPortULong](#)            [ReleaseDevice](#)

◆ 以双字节(16Bit)方式读 I/O 端口

函数原型:

**Visual C++:**

```

WORD ReadPortWord(HANDLE hDevice,
                  UINT nPort)

```

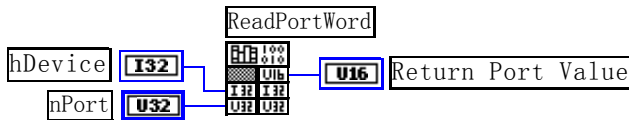
**Visual Basic:**

```

Declare Function ReadPortWord Lib "PCI2307" ( ByVal hDevice As Long, _
                                              ByVal nPort As Long ) As Integer

```

**LabVIEW:**



功能：以双字节(16Bit)方式读 I/O 端口。

参数：

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

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定的端口的值。

相关函数：[CreateDevice](#)                      [GetDeviceAddr](#)                      [WritePortByte](#)  
[WritePortWord](#)                      [WritePortULong](#)                      [ReadPortByte](#)  
[ReadPortWord](#)                      [ReadPortULong](#)                      [ReleaseDevice](#)

◆ 以四字节(32Bit)方式读 I/O 端口

函数原型：

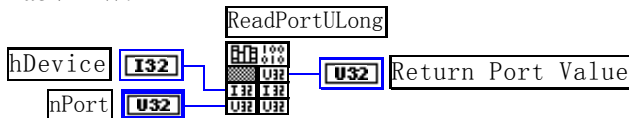
**Visual C++:**

ULONG ReadPortULong(HANDLE hDevice,  
 UINT nPort)

**Visual Basic:**

Declare Function ReadPortULong Lib "PCI2307" ( ByVal hDevice As Long, \_  
 ByVal nPort As Long ) As Long

**LabVIEW:**



功能：以四字节(32Bit)方式读 I/O 端口。

参数：

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

nPort 设备的 I/O 端口号。

返回值：返回由 nPort 指定端口的值。

相关函数：[CreateDevice](#)                      [GetDeviceAddr](#)                      [WritePortByte](#)  
[WritePortWord](#)                      [WritePortULong](#)                      [ReadPortByte](#)  
[ReadPortWord](#)                      [ReadPortULong](#)                      [ReleaseDevice](#)

第三节、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

函数原型：

**Visual C++:**

BOOL CreateVBThread(HANDLE \*hThread,  
 LPTHREAD\_START\_ROUTINE StartThread)

**Visual Basic:**

Declare Function CreateVBThread Lib "PCI2307" ( ByRef hThread As Long, \_  
 ByVal StartThread 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。

StartThread 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: StartThread 指向的函数或过程必须放在 VB 的模块文件中，如 PCI2307.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

函数原型:

**Visual C++:**

**BOOL TerminateVBThread(HANDLE hThreadHandle)**

**Visual Basic:**

**Declare Function TerminateVBThread Lib "PCI2307" (ByVal hThreadHandle As Long) As Boolean**

功能: 在VB中删除由[CreateVBThread](#)创建的子线程对象。

参数: hThreadHandle 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

函数原型:

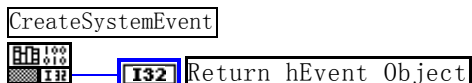
**Visual C++:**

**HANDLE CreateSystemEvent(void)**

**Visual Basic:**

**Declare Function CreateSystemEvent Lib "PCI2307" () 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

函数原型:

**Visual C++:**

BOOL ReleaseSystemEvent(HANDLE hEvent)

**Visual Basic:**

Declare Function ReleaseSystemEvent Lib "PCI2307" (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。

#### ◆ 高效高精度延时

函数原型:

**Visual C++:**

BOOL DelayTime (HANDLE hDevice,  
LONG nTime

**Visual Basic:**

Declare Function DelayTime Lib "PCI2307" (ByVal hDevice As Long, \_  
ByVal nTime As Long) As Boolean

**LabVIEW:**

请参考相关演示程序。

**功能:** 微秒级延时函数。

**参数:**

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

nTime 时间常数。单位 1 微秒。

**返回值:** 若成功, 返回 TRUE, 否则返回 FALSE, 用户可用 GetLastErrorEx 捕获错误码。

**相关函数:** 无。

### 第四节、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 NewFileName,  
int Mode)

**Visual Basic:**

Declare Function CreateFileObject Lib "PCI2307" (ByVal hDevice As Long, \_  
ByVal NewFileName 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](#)创建。

NewFileName 新文件名。

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
PCI2307_modeRead	0x0000	只读文件方式
PCI2307_modeWrite	0x0001	只写文件方式
PCI2307_modeReadWrite	0x0002	既读又写文件方式
PCI2307_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,
               PWORD pUserRegion,
               ULONG nWriteSizeBytes)
```

**Visual Basic:**

```
Declare Function WriteFile Lib "PCI2307" ( ByVal hFileObject As Long, _
                                           ByRef pUserRegion As Integer, _
                                           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。

pUserRegion 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,
               PWORD pFileUserRegion,
               ULONG OffsetBytes,
               ULONG nReadSizeBytes)
```

**Visual Basic:**

```
Declare Function ReadFile Lib "PCI2307" ( ByVal hFileObject As Long, _
                                           ByRef pFileUserRegion 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。

pFileUserRegion 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,  
ULONG nOffsetBytes)

**Visual Basic:**

Declare Function SetFileOffset Lib "PCI2307" (ByVal hFileObject As Long,  
ByVal nOffsetBytes As Long) As Boolean

**LabVIEW:**

请参考相关演示程序。

**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 "PCI2307" (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 "PCI2307" (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

函数原型:

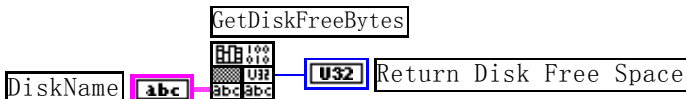
**Visual C++:**

ULONG GetDiskFreeBytes(LPCTSTR szDiskName )

**Visual Basic:**

Declare Function GetDiskFreeBytes Lib "PCI2307" (ByVal szDiskName As String ) As Long

**LabVIEW:**



**功能:** 取得指定磁盘的可用剩余空间(以字为单位)。

**参数:** `szDiskName` 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。