PXI8603 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

第一章 版权信息与命名约定

第一节、版权信息

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 PXIxxxx_则被省略。如 PXI8603_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、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 InitDeviceAD、ReadDeviceProAD_Npt 等。而底层用户函数如 WriteRegisterULong、ReadRegisterULong、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插座等管脚分配情况。

第二节、如何管理设备

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 InitDeviceAD 可以使用 hDevice 句柄以程序查询方式初始化设备的 AD 部件，ReadDeviceProAD_Npt 函数可以用 hDevice 句柄实现对 AD 数据的采样读取等。最后可以通过 ReleaseDevice 将 hDevice 释放掉。

第三节、如何用非空查询方式取得 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。然后用 StartDeviceAD 即可启动 AD 部件，开始 AD 采样，然后便可用 ReadDeviceProAD_Npt 反复读取 AD 数据以实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceAD，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：ReadDeviceProAD_Npt 虽然主要面对批量读取、高速连续采集而设计，但亦可用它以单点或几点的方式读取 AD 数据，以满足慢速、高实时性采集需要）。具体执行流程请看下面的图 2.1.1。

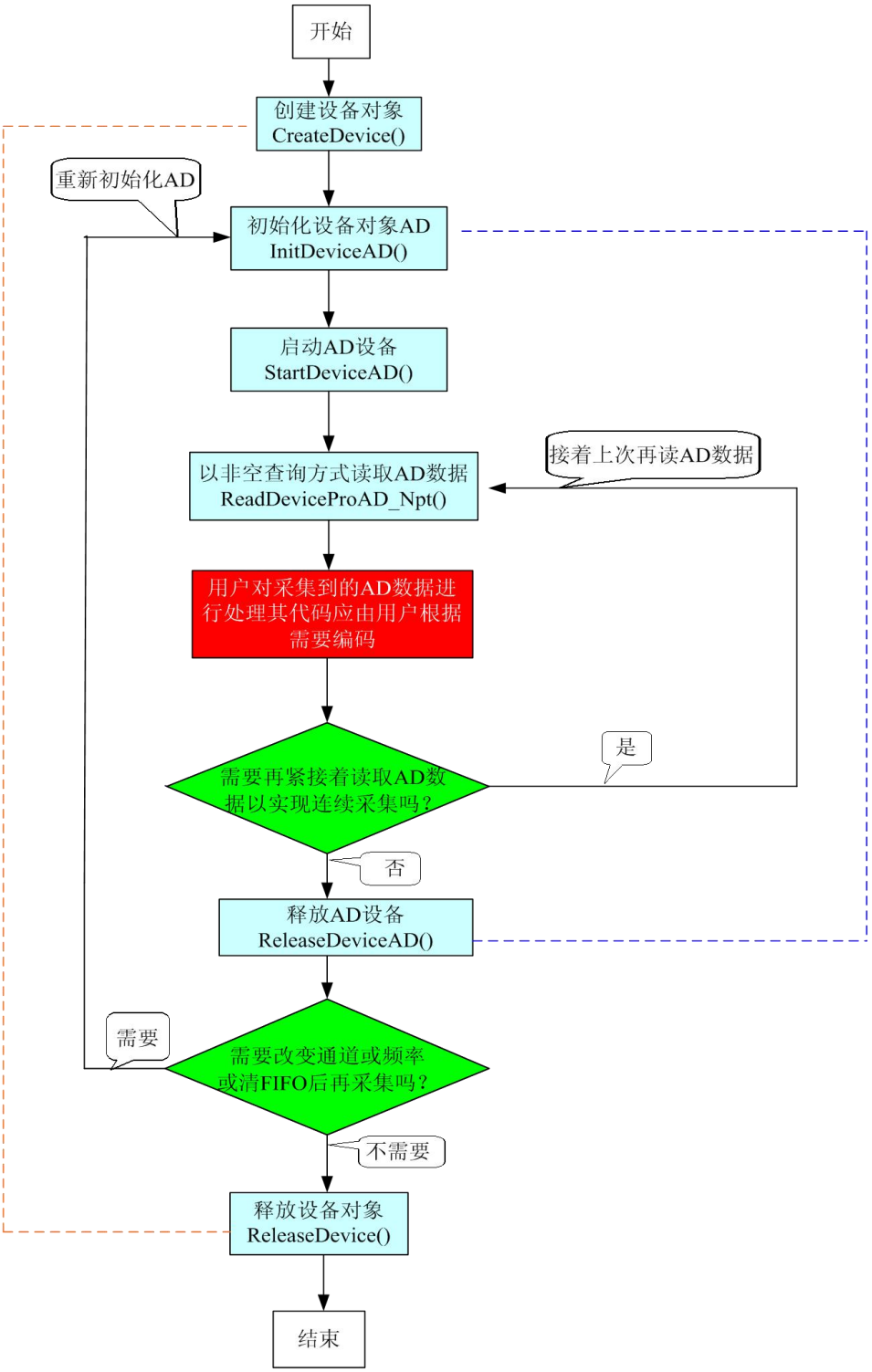


图 2.1.1 非空方式 AD 采样过程

第四节、如何用半满查询方式取得 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。然后用 StartDeviceAD 即可启动 AD 部件，开始 AD 采样，接着调用 GetDevStatusAD 函数以查询 AD 的存储器 FIFO 的半满状态，如果达到半满状态，即可用 ReadDeviceProAD Half 函数读取一批半满长度（或半满以下）的 AD 数据，然后接着在查询 FIFO 的半满状态，若有效再读取，就这样反复查询状态反复读取 AD 数据即可实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceAD，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：ReadDeviceProAD Half 函数在半满状态有效时也可以单点或几点的方式读取 AD 数据，只是到下一次半满信号到来时的时间间隔会变得非常短，而不再是半满间隔。）具体执行流程请看下面的图 2.1.2。

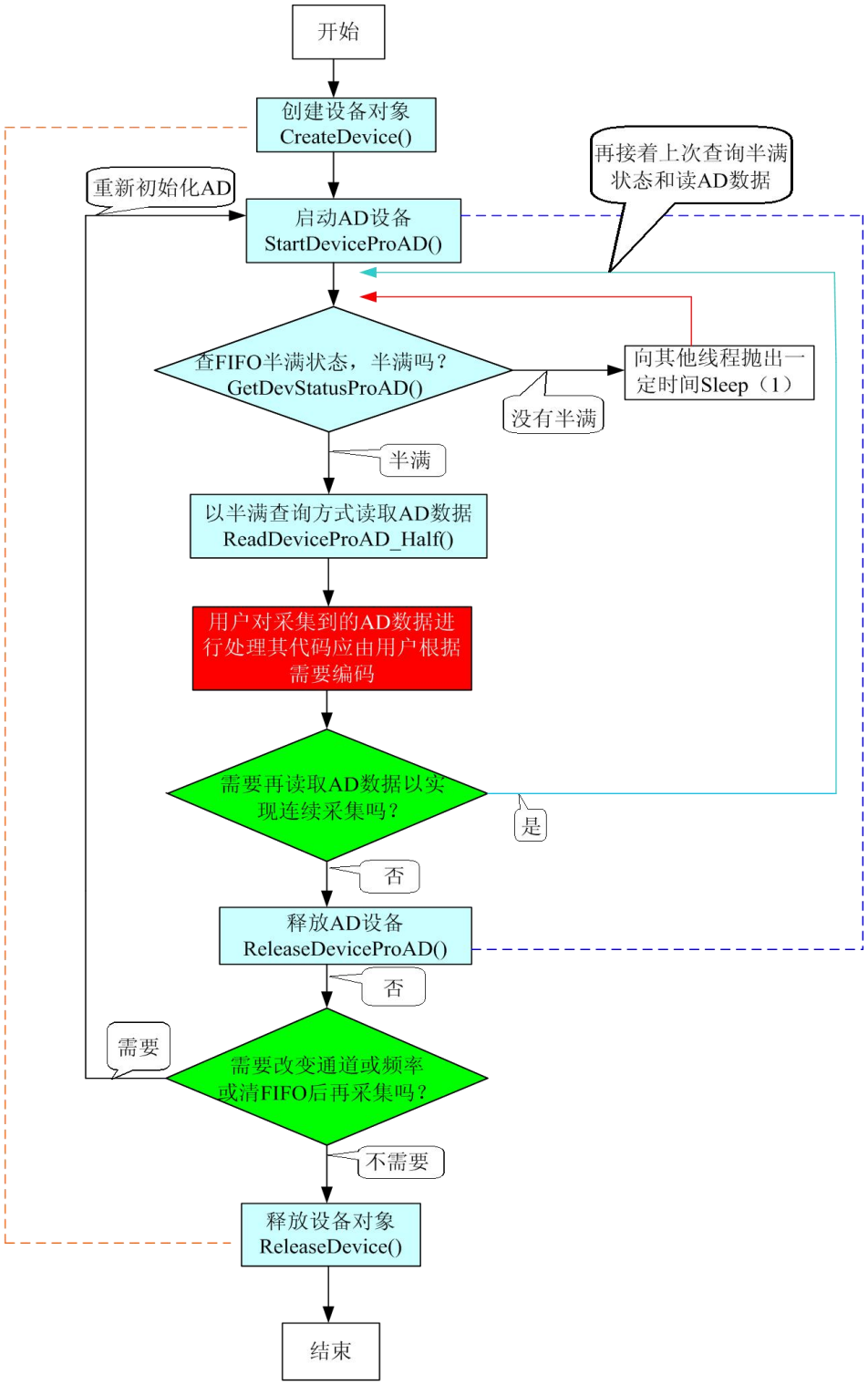


图 2.1.2 半满查询方式 AD 采样过程

第五节、如何用 DMA 方式取得 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。同时应调用 CreateSystemEvent 函数创建一个内核事件对象句柄 hDmaEvent 赋给 InitDeviceAD 的相应参数，它将作为 Dma 事件的变量。然后用 StartDeviceAD 即可启动 AD 部件，开始 AD 采样，接着调用 Win32 API 函数 CreateFileObject 等待 hDmaEvent 事件的发生，当前缓冲段没有被 DMA 完成时，自动使所在线程进入睡眠状态（不消耗 CPU 时间），反之，则立即唤醒所在线程，执行它下面的代码，此时您便可用 GetDevStatusAD 来确定哪一段缓冲是新的数据，即刻处理该数据，至到所有的缓冲段变为旧数据段。然后再回到 CreateFileObject，就这样反复读取 AD 数据即可实现连续不间断采样。当您需要暂停设备时，执行 StopDeviceAD，当您需要关闭 AD 设备时，ReadDeviceAD 便可帮您实现（但设备对象 hDevice 依然存在）。具体执行流程请看图 2.1.3。

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，在结束时就须执行一次 ReleaseDevice。

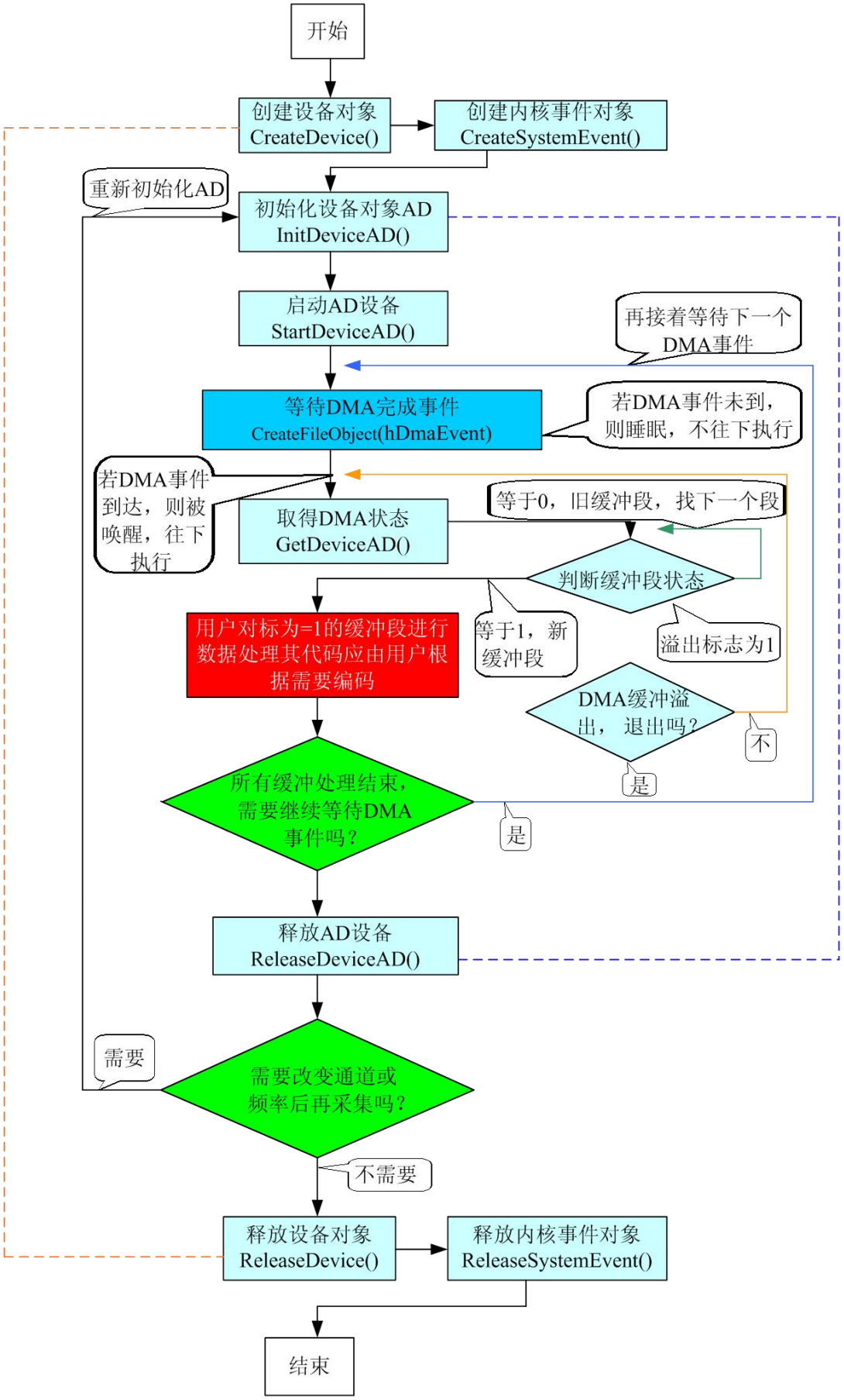


图 2.1.3 DMA 方式 AD 采集实现过程

第六节、如何实现数字量的简便操作

当您有了 hDevice 设备对象句柄后，便可用 SetDeviceDO 函数实现数字量的输出操作，其各路数字量的输出状态由其 bDOSts[8]中的相应元素决定。由 GetDeviceDI 函数实现数字量的输入操作，其各路数字量的输入状态由其 bDISts[8]中的相应元素决定。

第七节、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，WriteRegisterByte，WriteRegisterWord，WriteRegisterULong，ReadRegisterByte，ReadRegisterWord，ReadRegisterULong 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 PXI 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 PXI 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要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（如 InitDeviceAD）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 ReadDeviceProAD_Npt（或 ReadDeviceProAD Half）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 PXI 总线复杂的控制协议，同是还可以省掉您许多繁琐的工作，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，然后使用 ReadRegisterULong 和 WriteRegisterULong 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。

第一节、设备驱动接口函数总列表（每个函数省略了前缀“PXI8603_”）

函数名	函数功能	备注
① 设备对象操作函数		
CreateDevice	创建 PXI 设备对象(用设备逻辑号)	上层及底层用户
GetDeviceCount	取得同一种 PXI 设备的总台数	上层及底层用户
GetDeviceCurrentID	获取设备的 ID 号	上层及底层用户
ListDeviceDlg	列表所有同一种 PXI 设备的各种配置	上层及底层用户
ReleaseDevice	关闭设备，且释放 PXI 总线设备对象	上层及底层用户
②AD 数据读取函数		
GetDevTriggerPos	取得触发位置	上层用户
InitDeviceAD	初始化 AD 部件准备传输	上层用户
StartDeviceAD	启动 AD 设备，开始转换	上层用户
ReadDeviceProAD_Npt	读取设备上的 AD 数据(程序非空方式)	上层用户
GetDevStatusAD	在 AD 采样过程中取得设备的各种状态	上层用户
ReadDeviceProAD_Half	读取设备上的 AD 数据(程序半满方式)	上层用户

ReadDeviceDmaAD	DMA 读 AD 数据	上层用户
StopDeviceAD	暂停设备	上层用户
ReleaseDeviceAD	释放设备上的 AD 部件	上层用户
③ AD 硬件参数操作函数		
SaveParaAD	将当前的 AD 采样参数保存至系统中	上层用户
LoadParaAD	将 AD 采样参数从系统中读出	上层用户
ResetParaAD	将 AD 采样参数恢复至出厂默认值	上层用户
④ DA 数据读取函数		
SetDevTrigLevelDA	设置触发电平(mV)	上层用户
SetDevOutputImpedance	设置输出阻抗	上层用户
SetDevFrequencyDA	在 DA 转换过程中，动态改变采样频率	上层用户
ReadSegmentInfo	读段信息	上层用户
InitDeviceDA	初始化设备，当返回 TRUE 后，设备即准备就绪	上层用户
WriteDeviceBulkDA	在 DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)	上层用户
ReadDeviceBulkDA	用此函数将板载 RAM 中的数据读回计算机(程序方式)	上层用户
EnableDeviceDA	在初始化之后，使能设备	上层用户
SetDeviceTrigDA	当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）	上层用户
GetDevStatusDA	在 DA 采样过程中取得设备的各种状态,返回值表示函数是否成功	上层用户
DisableDeviceDA	在使设备之后，禁止设备	上层用户
ReleaseDeviceDA	禁止 DA 设备，且释放资源	上层用户
WriteDeviceOneDA	DA 单点输出	上层用户
⑤ DA 的硬件参数操作函数		
SaveParaDA	将当前的 DA 采样参数保存至系统中	上层用户
LoadParaDA	将 DA 采样参数从系统中读出	上层用户
ResetParaDA	将 DA 采样参数恢复至出厂默认值	上层用户
⑥ 数字 I/O 输入输出函数		
GetDeviceDI	取得开关量状态	上层用户
SetDeviceDO	输出开关量状态	上层用户
RetDeviceDO	回读开关量输出状态	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\PXI8603\INCLUDE\PXI8603.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 PXI8603.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。

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，在弹出的对话中选择 PXI8603.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，我们不能保证完全顺利运行。

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 DeviceLgcID = 0)

Visual Basic：

Declare Function CreateDevice Lib "PXI8603" (ByVal DeviceLgcID As Integer) As Long

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(Identifier)标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 DeviceID 标识值为名称后缀的标识符来确认和管理该设备。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： GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

Visual C++程序举例

```
    :
HANDLE hDevice;     // 定义设备对象句柄
int DeviceLgcID = 0;
hDevice = PXI8603_CreateDevice (DeviceLgcID); // 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
Dim DeviceLgcID As Long
DeviceLgcID = 0
hDevice = PXI8603_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 PXI8603 设备的总数量**

函数原型：

Visual C++:

int GetDeviceCount (HANDLE hDevice)

Visual Basic:

Declare Function GetDeviceCount Lib "PXI8603" (ByVal hDevice As Long) As Integer

功能：取得 PXI8603 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 PXI8603 的数量。

相关函数： CreateDevice　GetDeviceCurrentID　ListDeviceDlg　ReleaseDevice

♦ **取得该设备当前相应的 ID 号**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceCurrentID Lib "PXI8603" (ByVal hDevice As Long, _
                                                   ByRef DeviceLgcID As Long, _
                                                   ByRef DevicePhysID As Long) As Boolean
```

功能：取得指定设备相应 ID 号。

参数：

hDevice　设备对象句柄，它指向要取得逻辑号的设备，它应由 CreateDevice 创建。

DeviceLgcID　返回设备的逻辑 ID，它的取值范围为[0, 15]。

DevicePhysID　返回设备的物理 ID 号。

返回值：如果初始化设备对象成功，则返回 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　GetDeviceCount　ListDeviceDlg　ReleaseDevice

♦ **用对话框控件列表计算机系统中所有 PXI8603 设备各种配置信息**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function ListDeviceDlg Lib "PXI8603" (ByVal hDevice As Long) As Boolean
```

功能：列表系统中 PXI8603 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 PXI8603 设备的配置情况。

相关函数：CreateDevice　GetDeviceCount　GetDeviceCurrentID　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 "PXI8603" (ByVal hDevice As Long) As Boolean
```

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　GetDeviceCount　GetDeviceCurrentID　ListDeviceDlg

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、AD 数据读取函数原型说明

♦ **取得触发位置**

函数原型：

Visual C++:

BOOL GetDevTriggerPos (HANDLE hDevice,
PULONG nTriggerPos);

Visual Basic:

Declare Function GetDevTriggerPos Lib "PXI8603" (ByVal hDevice As Long,
ByRef nTriggerPos As Long) As Boolean

功能：取得触发位置。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

nTriggerPos　取得触发位置值(只有当 bTriggerFlag 为 TRUE 有效)。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　InitDeviceAD　StartDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　ReadDeviceDmaAD　StopDeviceAD　ReleaseDeviceAD

- **初始化设备**

函数原型：

Visual C++:

BOOL InitDeviceAD(HANDLE hDevice,
PPXI8603_PARA_AD pADPara);

Visual Basic:

Declare Function InitDeviceAD Lib "PXI8603" (ByVal hDevice As Long,
ByRef pADPara As PXI8603_PARA_AD) As Boolean

功能：初始化设备，当返回 TRUE 后，设备即准备就绪。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pADPara　硬件参数, 它仅在此函数中决定硬件状态。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　StartDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　ReadDeviceDmaAD　StopDeviceAD　ReleaseDeviceAD

- **在初始化之后，启动设备**

函数原型：

Visual C++:

BOOL StartDeviceAD(HANDLE hDevice);

Visual Basic:

Declare Function StartDeviceAD Lib "PXI8603" (ByVal hDevice As Long) As Boolean

功能：在初始化之后，启动设备。

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。

相关函数：CreateDevice　InitDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　ReadDeviceDmaAD　StopDeviceAD　ReleaseDeviceAD

- **当 AD 标志有效时，用此函数读取设备上的 AD 数据(程序非空方式)**

函数原型：

Visual C++:

BOOL ReadDeviceProAD_Npt(HANDLE hDevice,
ULONG ADBuffer[],
LONG nReadSizeWords,
PLONG nRetSizeWords);

Visual Basic:

Declare Function ReadDeviceProAD_Npt Lib "PXI8603" (ByVal hDevice As Long,

ByRef ADBuffer As Long,
ByVal nReadSizeWords As Long,
ByRef nRetSizeWords As Long) As Boolean

功能：当 AD 标志有效时，用此函数读取设备上的 AD 数据(程序非空方式)。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
ADBuffer[] 接受原始 AD 数据的用户缓冲区。
nReadSizeWords 相对于偏位点后的数据长度(字)。
nRetSizeWords 返回实际读取的长度(字)
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice InitDeviceAD StartDeviceAD GetDevStatusAD
ReadDeviceProAD_Half ReadDeviceDmaAD StopDeviceAD ReleaseDeviceAD

- **在 AD 采样过程中取得设备的各种状态**

函数原型：
Visual C++:

```
BOOL GetDevStatusAD(HANDLE hDevice,
                    PPXI8603_STATUS_AD pADStatus);
```

Visual Basic:

```
Declare Function GetDevStatusAD Lib "PXI8603" (ByVal hDevice As Long,
                                               ByRef pADStatus As PXI8603_STATUS_AD) As Boolean
```

功能：在 AD 采样过程中取得设备的各种状态,返回值表示函数是否成功。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
pADStatus AD 的各种信息结构体。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice InitDeviceAD StartDeviceAD ReadDeviceProAD_Npt
ReadDeviceProAD_Half ReadDeviceDmaAD StopDeviceAD ReleaseDeviceAD

- **当 AD 标志有效时，用此函数读取设备上的 AD 数据(程序半满方式)**

函数原型：
Visual C++:

```
BOOL ReadDeviceProAD_Half(HANDLE hDevice,
                          ULONG ADBuffer[],
                          LONG nReadSizeWords,
                          PLONG nRetSizeWords);
```

Visual Basic:

```
Declare Function ReadDeviceProAD_Half Lib "PXI8603" (ByVal hDevice As Long,
                                                     ByRef ADBuffer As Long,
                                                     ByVal nReadSizeWords As Long,
                                                     ByRef nRetSizeWords As Long) As Boolean
```

功能：当 AD 标志有效时，用此函数读取设备上的 AD 数据(程序半满方式)。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
ADBuffer[] 接受原始 AD 数据的用户缓冲区。
nReadSizeWords 相对于偏位点后的数据长度(字)。
nRetSizeWords 返回实际读取的长度(字)
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice InitDeviceAD StartDeviceAD ReadDeviceProAD_Npt
GetDevStatusAD ReadDeviceDmaAD StopDeviceAD ReleaseDeviceAD

♦ **DMA 读 AD 数据**

函数原型：

Visual C++:

```
BOOL ReadDeviceDmaAD(HANDLE hDevice,
                     PULONG pADBuffer,
                     ULONG nReadSizeWords,
                     PLONG nRetSizeWords);
```

Visual Basic:

```
Declare Function ReadDeviceDmaAD Lib "PXI8603" (ByVal hDevice As Long,
                                                ByRef pADBuffer As Long,
                                                ByVal nReadSizeWords As Long,
                                                ByRef nRetSizeWords As Long) As Boolean
```

功能：DMA 读 AD 数据。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pADBuffer[]　接受原始 AD 数据的用户缓冲区。

nReadSizeWords　相对于偏位点后的数据长度(字)。

nRetSizeWords　返回实际读取的长度(字)

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　InitDeviceAD　StartDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　StopDeviceAD　ReleaseDeviceAD

♦ **在启动设备之后，暂停设备**

函数原型：

Visual C++:

```
BOOL StopDeviceAD(HANDLE hDevice);
```

Visual Basic:

```
Declare Function StopDeviceAD Lib "PXI8603" (ByVal hDevice As Long) As Boolean
```

功能：在启动设备之后，暂停设备。

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。

相关函数：CreateDevice　InitDeviceAD　StartDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　ReadDeviceDmaAD　ReleaseDeviceAD

♦ **关闭 AD 设备，禁止传输，且释放资源**

函数原型：

Visual C++:

```
BOOL ReleaseDeviceAD(HANDLE hDevice);
```

Visual Basic:

```
Declare Function ReleaseDeviceAD Lib "PXI8603" (ByVal hDevice As Long) As Boolean
```

功能：关闭 AD 设备,禁止传输,且释放资源。

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。

相关函数：CreateDevice　InitDeviceAD　StartDeviceAD　ReadDeviceProAD_Npt　GetDevStatusAD　ReadDeviceProAD_Half　ReadDeviceDmaAD　StopDeviceAD

第四节、AD 数据读取函数原型说明

♦ 往 Windows 系统写入设备硬件参数函数

函数原型：

Viusal C++:

```
BOOL SaveParaAD (HANDLE hDevice,
                 PPXI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function SaveParaAD Lib "PXI8603" (ByVal hDevice As Long, _
                                           ByRef pADPara As PXI8603_PARA_AD) As Boolean
```

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 设备硬件参数，关于 PXI8603_PARA_AD 的详细介绍请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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 ReleaseDevice

♦ 从 Windows 系统中读入硬件参数函数

函数原型：

Visual C++:

```
BOOL LoadParaAD (HANDLE hDevice,
                 PPXI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function LoadParaAD Lib "PXI8603" (ByVal hDevice As Long, _
                                           ByRef pADPara As PXI8603_PARA_AD) As Boolean
```

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pADPara 属于 PXI8603_PARA_AD 的结构指针类型，它负责返回 PXI 硬件参数值，关于结构指针类型 PXI8603_PARA_AD 请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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 ReleaseDevice

♦ AD 采样参数复位至出厂默认值函数

函数原型：

Viusal C++:

```
BOOL ResetParaAD (HANDLE hDevice,
                  PPXI8603_PARA_AD pADPara)
```

Visual Basic:

```
Declare Function ResetParaAD Lib "PXI8603" (ByVal hDevice As Long, _
                                            ByRef pADPara As PXI8603_PARA_AD) As Boolean
```

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 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 PXI8603_PARA_AD 的详细介绍请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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

第五节、DA 数据读取函数原型说明

♦ 设置触发电平(mV)

函数原型：

Visual C++:

```
BOOL SetDevTrigLevelDA(HANDLE hDevice,
                       float fTrigLevelVolt);
```

Visual Basic:

```
Declare Function SetDevTrigLevelDA Lib "PXI8603" (ByVal hDevice As Long,
                                                  ByVal fTrigLevelVolt As Double) As Boolean
```

功能：设置触发电平(mV)。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

fTrigLevelVolt　触发电平,范围为-10000 - +10000mV。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　SetDevOutputImpedance　SetDevFrequencyDA　ReadSegmentInfo　InitDeviceDA　WriteDeviceBulkDA　EnableDeviceDA　SetDeviceTrigDA　GetDevStatusDA　DisableDeviceDA　ReleaseDeviceDA

♦ 设置输出阻抗

函数原型：

Visual C++:

```
BOOL SetDevOutputImpedance(HANDLE hDevice,
                           LONG lImpedance);
```

Visual Basic:

```
Declare Function SetDevOutputImpedance Lib "PXI8603" (ByVal hDevice As Long,
                                                      ByVal Impedance As Long) As Boolean
```

功能：设置输出阻抗。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

lImpedance　阻抗（=0：50R，　=1：75R）。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　SetDevFrequencyDA　ReadSegmentInfo　InitDeviceDA　WriteDeviceBulkDA　EnableDeviceDA　SetDeviceTrigDA　GetDevStatusDA　DisableDeviceDA　ReleaseDeviceDA

♦ 在 DA 转换过程中，动态改变采样频率

函数原型：

Visual C++:

```
BOOL SetDevFrequencyDA(HANDLE hDevice,
                       LONG nFrequency,
                       int nDAChannel);
```

Visual Basic:

```
Declare Function SetDevFrequencyDA Lib "PXI8603" (ByVal hDevice As Long,
                                                  ByVal nFrequency As Long,
                                                  ByVal nDAChannel As Integer) As Boolean
```

功能：在 DA 转换过程中，动态改变采样频率。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

nFrequency　DA 采样频率(Hz)。

nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA EnableDeviceDA SetDeviceTrigDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

◆ **读段信息**

函数原型：

Visual C++:

```
BOOL ReadSegmentInfo(HANDLE hDevice,
                     LONG SegmentCount,
                     PXI8603_SEGMENT_INFO SegmentInfo[],
                     int nDAChannel);
```

Visual Basic:

```
Declare Function ReadSegmentInfo Lib "PXI8603" (ByVal hDevice As Long,
                                                ByVal SegmentCount As Long,
                                                ByRef SegmentInfo As PXI8603_SEGMENT_INFO,
                                                ByVal nDAChannel As Integer) As Boolean
```

功能：读段信息。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
SegmentCount 段循环次数。
SegmentInfo[] 段信息。
nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance SetDevFrequencyDA InitDeviceDA WriteDeviceBulkDA EnableDeviceDA SetDeviceTrigDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

◆ **初始化设备，当返回 TRUE 后，设备即准备就绪**

函数原型：

Visual C++:

```
BOOL InitDeviceDA(HANDLE hDevice,
                  LONG SegmentCount,
                  PXI8603_SEGMENT_INFO SegmentInfo[],
                  PPXI8603_PARA_DA pDAPara,
                  int nDAChannel);
```

Visual Basic:

```
Declare Function InitDeviceDA Lib "PXI8603" (ByVal hDevice As Long,
                                             ByVal SegmentCount As Long,
                                             ByRef SegmentInfo As PXI8603_SEGMENT_INFO,
                                             ByRef pDAPara As PXI8603_PARA_DA,
                                             ByVal nDAChannel As Integer) As Boolean
```

功能：初始化设备，当返回 TRUE 后，设备即准备就绪。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
SegmentCount 段循环次数。
SegmentInfo[] 段信息。
pDAPara 硬件参数，它仅在此函数中决定硬件状态
nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance SetDevFrequencyDA ReadSegmentInfo

WriteDeviceBulkDA EnableDeviceDA SetDeviceTrigDA GetDevStatusDA
DisableDeviceDA ReleaseDeviceDA

♦ **在 DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)**

函数原型：

Visual C++:

```
BOOL WriteDeviceBulkDA(HANDLE hDevice,
                       SHORT DABuffer[],
                       LONG nWriteOffsetWords,
                       LONG nWriteSizeWords,
                       PLONG nRetSizeWords,
                       int nDAChannel);
```

Visual Basic:

```
Declare Function WriteDeviceBulkDA Lib "PXI8603" (ByVal hDevice As Long,
                                                  ByRef DABuffer As Integer,
                                                  ByVal nWriteOffsetWords As Long,
                                                  ByVal nWriteSizeWords As Long,
                                                  ByRef nRetSizeWords As Long,
                                                  ByVal nDAChannel As Integer) As Boolean
```

功能：在 DA 输出前，用此函数将 DA 数据写入板载 RAM 中(程序方式)。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

DABuffer[] 携带原始 DA 数据的用户缓冲区。

nWriteOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)。

nWriteSizeWords 相对于偏位点后写入的数据长度(字)。

nRetSizeWords 返回实际写出的长度(字)

nDAChannel DA 通道号[0, 1]。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice SetDevOutputImpedance SetDevFrequencyDA ReadSegmentInfo
InitDeviceDA EnableDeviceDA SetDeviceTrigDA GetDevStatusDA DisableDeviceDA
ReleaseDeviceDA

♦ **用此函数将板载 RAM 中的数据读回计算机(程序方式)**

函数原型：

Visual C++:

```
BOOL ReadDeviceBulkDA(HANDLE hDevice,
                      SHORT DABuffer[],
                      LONG nReadOffsetWords,
                      LONG nReadSizeWords,
                      PLONG nRetSizeWords,
                      int nDAChannel);
```

Visual Basic:

```
Declare Function ReadDeviceBulkDA Lib "PXI8603" (ByVal hDevice As Long,
                                                 ByRef DABuffer As Integer,
                                                 ByVal nReadOffsetWords As Long,
                                                 ByVal nReadSizeWords As Long,
                                                 ByRef nRetSizeWords As Long,
                                                 ByVal nDAChannel As Integer) As Boolean
```

功能：用此函数将板载 RAM 中的数据读回计算机(程序方式)。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

DABuffer[] 携带原始 DA 数据的用户缓冲区。

nReadOffsetWords 相对于该通道物理 RAM 零位置的偏移点(字)
nReadSizeWords 相对于偏位点后写入的数据长度(字)。
nRetSizeWords 返回实际写出的长度(字)
nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA EnableDeviceDA SetDeviceTrigDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **在初始化之后，使能设备**

函数原型：
Visual C++:

```
BOOL EnableDeviceDA(HANDLE hDevice,
                    int nDAChannel);
```

Visual Basic:

```
Declare Function EnableDeviceDA Lib "PXI8603" (ByVal hDevice As Long,
                                               ByVal nDAChannel As Integer) As Boolean
```

功能：在初始化之后，使能设备。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA SetDeviceTrigDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）**

函数原型：
Visual C++:

```
BOOL SetDeviceTrigDA(HANDLE hDevice,
                     BOOL bSetSyncTrig,
                     int nDAChannel);
```

Visual Basic:

```
Declare Function SetDeviceTrigDA Lib "PXI8603" (ByVal hDevice As Long,
                                                ByVal bSetSyncTrig As Boolean,
                                                ByVal nDAChannel As Integer) As Boolean
```

功能：当设备使能允许后，产生软件触发事件（只有触发源为软件触发时有效）。
参数：
hDevice 设备对象句柄，它应由 CreateDevice 创建。
bSetSyncTrig 是否置同步触发。
nDAChannel DA 通道号[0, 1]。
返回值：若成功，则返回 TRUE，否则返回 FALSE。
相关函数： CreateDevice SetDevOutputImpedance SetDevFrequencyDA ReadSegmentInfo InitDeviceDA WriteDeviceBulkDA EnableDeviceDA GetDevStatusDA DisableDeviceDA ReleaseDeviceDA

♦ **在 DA 采样过程中取得设备的各种状态,返回值表示函数是否成功**

函数原型：
Visual C++:

```
BOOL GetDevStatusDA(HANDLE hDevice,
                    PPXI8603_STATUS_DA pDAStatus,
                    int nDAChannel);
```

Visual Basic:

```
Declare Function GetDevStatusDA Lib "PXI8603" (ByVal hDevice As Long,
                                                ByRef pDAStatus As PXI8603_STATUS_DA,
                                                ByVal nDAChannel As Integer) As Boolean
```

功能：在 DA 采样过程中取得设备的各种状态,返回值表示函数是否成功。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pDAStatus　DA 的各种信息结构体。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　SetDevOutputImpedance　SetDevFrequencyDA　ReadSegmentInfo　InitDeviceDA　WriteDeviceBulkDA　EnableDeviceDA　SetDeviceTrigDA　DisableDeviceDA　ReleaseDeviceDA

◆ **在使设备之后，禁止设备**

函数原型：

Visual C++:

```
BOOL DisableDeviceDA(HANDLE hDevice,
                     int nDAChannel);
```

Visual Basic:

```
Declare Function DisableDeviceDA Lib "PXI8603" (ByVal hDevice As Long,
                                                ByVal nDAChannel As Integer) As Boolean
```

功能：在使设备之后，禁止设备。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　SetDevOutputImpedance　SetDevFrequencyDA　ReadSegmentInfo　InitDeviceDA　WriteDeviceBulkDA　EnableDeviceDA　SetDeviceTrigDA　GetDevStatusDA　ReleaseDeviceDA

◆ **禁止 DA 设备，且释放资源**

函数原型：

Visual C++:

```
BOOL ReleaseDeviceDA(HANDLE hDevice,
                     int nDAChannel);
```

Visual Basic:

```
Declare Function ReleaseDeviceDA Lib "PXI8603" (ByVal hDevice As Long,
                                                ByVal nDAChannel As Integer) As Boolean
```

功能：禁止 DA 设备，且释放资源。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice　SetDevOutputImpedance　SetDevFrequencyDA　ReadSegmentInfo　InitDeviceDA　WriteDeviceBulkDA　EnableDeviceDA　SetDeviceTrigDA　GetDevStatusDA　DisableDeviceDA

◆ **DA 单点输出**

函数原型：

Visual C++:

```
BOOL WriteDeviceOneDA(HANDLE hDevice,
                      ULONG  ulDataCode,
                      int nDAChannel);
```

Visual Basic:

```
Declare Function WriteDeviceOneDA Lib "PXI8603" (ByVal hDevice As Long,
                                                 ByVal ulDataCode As Long,
                                                 ByVal nDAChannel As Integer) As Boolean
```

功能：DA 单点输出。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

ulDataCode　输入码值（0—4095）

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，则返回 TRUE，否则返回 FALSE。

相关函数：CreateDevice

第六节、DA 硬件参数操作函数原型说明

♦ 往 Windows 系统写入设备硬件参数函数

函数原型：

Viusal C++:

```
BOOL SaveParaDA(HANDLE hDevice,
                PPXI8603_PARA_DA pDAPara,
                int nDAChannel);
```

Visual Basic:

```
Declare Function SaveParaDA Lib "PXI8603" (ByVal hDevice As Long,
                                           ByRef pDAPara As PXI8603_PARA_DA,
                                           ByVal nDAChannel As Integer) As Boolean
```

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。

pDAPara　设备硬件参数，关于 PXI8603_PARA_DA 的详细介绍请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ReleaseDevice

♦ 从 Windows 系统中读入硬件参数函数

函数原型：

Visual C++:

```
BOOL LoadParaDA(HANDLE hDevice,
                PPXI8603_PARA_DA pDAPara,
                int nDAChannel);
```

Visual Basic:

```
Declare Function LoadParaDA Lib "PXI8603" (ByVal hDevice As Long,
                                           ByRef pDAPara As PXI8603_PARA_DA,
                                           ByVal nDAChannel As Integer) As Boolean
```

功能：负责从 Windows 系统中读取设备的硬件参数。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pDAPara 属于 PXI8603_PARA_AD 的结构指针类型，它负责返回 PXI 硬件参数值，关于结构指针类型 PXI8603_PARA_AD 请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ReleaseDevice

♦ **AD 采样参数复位至出厂默认值函数**

函数原型：

Viusal C++:

```
BOOL ResetParaDA(HANDLE hDevice,
                 PPXI8603_PARA_DA pDAPara,
                 int nDAChannel);
```

Visual Basic:

```
Declare Function ResetParaDA Lib "PXI8603" (ByVal hDevice As Long,
                                            ByRef pDAPara As PXI8603_PARA_DA,
                                            ByVal nDAChannel As Integer) As Boolean
```

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 创建。

pDAPara　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 PXI8603_PARA_DA 的详细介绍请参考 PXI8603.h 或 PXI8603.Bas 或 PXI8603.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。

nDAChannel　DA 通道号[0, 1]。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice　LoadParaAD　SaveParaAD　ResetParaAD　ReleaseDevice

第七节、数字 I/O 输入输出函数原型说明

♦ **数字量输入**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceDI Lib "PXI8603" (ByVal hDevice As Long,
                                            ByRef bDISts As Byte) As Boolean
```

功能：负责将 PXI 设备上的输入数字量状态读入到 bDISts[x]数组参数中。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

bDISts　八路数字量输入状态的参数结构，共有 8 个元素，分别对应于 DI0～DI7 路数字量输入状态位。如 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 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　RetDeviceDO　ReleaseDevice

♦ **输出开关量状态**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function PXI8603_SetDeviceDO Lib "PXI8603" (ByVal hDevice As Long,
```

ByRef bDOSts As Byte) As Boolean

功能：负责将 PXI 设备上的输出数字量置成由 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 八路数字量输出状态的参数结构，共有 8 个元素，分别对应于 DI0～DI7 路数字量输出状态位。如 bDOSts[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，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI RetDeviceDO ReleaseDevice

- **回读开关量输出状态**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function RetDeviceDO Lib "PXI8603" (ByVal hDevice As Long,
                                            ByRef bDOSts As Byte) As Boolean
```

功能：负责将 PXI 设备上的输出数字量的相应状态回读到 bDOSts[x]中。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

bDOSts 八路数字量输出状态的参数结构，共有 8 个元素，分别对应于 DO0～DO7 路数字量输出状态位。如 bDOSts[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，否则返回 FALSE。

相关函数： CreateDevice GetDeviceDI SetDeviceDO 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

第一节、AD 硬件参数介绍（PXI8603_PARA_AD）

Visual C++:

```
typedef struct _PXI8603_PARA_AD
{
LONG ADMode;
LONG FirstChannel;
LONG LastChannel;
LONG Frequency;
LONG GroupInterval;
LONG LoopsOfGroup;
LONG Gains;
LONG InputRange;
LONG TriggerMode;
LONG TriggerType;
LONG TriggerDir;
LONG TriggerSource;
LONG TrigLevelVolt;
LONG TrigWindow;
```

```
    LONG ClockSource;
    LONG bClockOutput;
    LONG GroundingMode;
    } PXI8603_PARA_AD, *PPXI8603_PARA_AD;
```

Visual Basic：

```
Type PXI8603_PARA_AD
    ADMode              As Long '
    FirstChannel        As Long     '
    LastChannel         As Long '
    Frequency           As Long     '
    GroupInterval       As Long '
    LoopsOfGroup        As Long     '
    Gains               As Long '
    InputRange          As Long     '
    TriggerMode         As Long '
    TriggerType         As Long '
    TriggerDir          As Long
    TriggerSource       As Long     '
    TrigLevelVolt       As Long     '
    TrigWindow          As Long         '
    ClockSource         As Long     '
    bClockOutput        As Long     '
    GroundingMode       As Long     '

End Type
```

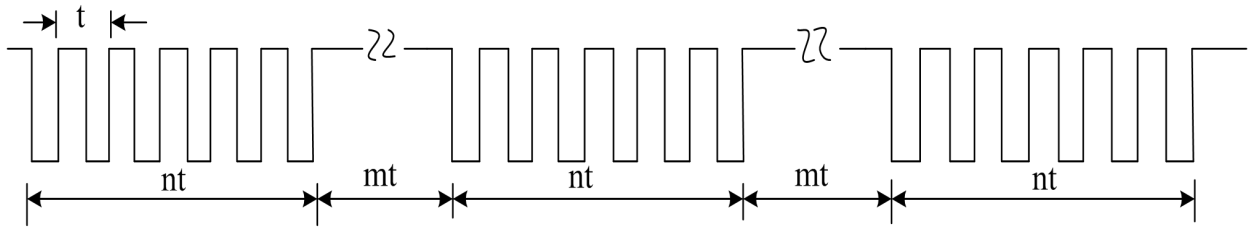
此结构主要用于设定设备 AD 硬件参数值，用这个参数结构对设备进行硬件配置完全由 InitDeviceAD 函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

ADMode AD 采样模式。它的取值如下表：

常量名	常量值	功能定义
PXI8603_ADMODE_SEQUENCE	0x00	连续采集模式
PXI8603_ADMODE_GROUP	0x01	分组采集模式

连续采集模式：表示所有通道在采样过程中均按相等时间间隔转换，即所有被采样的点在时间轴上其间隔完全相等。在下图中的过程，若没有 mt 的组间间隔时间 GroupInterval，就属于连续采样的模式。

分组采集模式：表示所有采样的数据点均以指定的通道数分成若干组，组内各通道数据点按等间隔采样，其频率由 Frequency 参数决定，组与组之间则有相当的间隔时间，其间隔长度由参数 GroupInterval 决定，可以精确到微秒。如下图 即是分组采样的整过情况：



说明：t=1/Frequency
mt = GroupInterval
n = ChannelCount

FirstChannel　AD 采样首通道，其取值范围为[0, 15]，它应等于或小于 LastChannel 参数。
LastChannel　AD 采样末通道，其取值范围为[0, 15]，它应等于或大于 FirstChannel 参数。

Frequency AD 采样频率，本设备的频率取值范围为[1, 500000]。

GroupInterval 组间间隔，单位微秒 uS，其范围[1, 419430]，通常由用户确定。但是一般情况下，此间隔时间应不小于组内相邻两通道的间隔。在内时钟连续采集模式和外时钟模式下，则参数无效。

LoopsOfGroup 在分组采集模式中，控制各组的循环次数。取值范围为[1, 255]。

Gains AD 采样程控增益。

常量名	常量值	功能定义
PXI8603_GAINS_1MULT	0x00	1 倍增益
PXI8603_GAINS_2MULT	0x01	2 倍增益
PXI8603_GAINS_4MULT	0x02	4 倍增益
PXI8603_GAINS_8MULT	0x03	8 倍增益

InputRange 模拟量输入量程选择。

常量名	常量值	功能定义
PXI8603_INPUT_N10000_P10000mV	0x00	±10000mV
PXI8603_INPUT_N5000_P5000mV	0x01	±5000mV
PXI8603_INPUT_N2500_P2500mV	0x02	±2500mV
PXI8603_INPUT_0_P10000mV	0x03	0～10000mV

TriggerMode AD 触发模式。

常量名	常量值	功能定义
PXI8603_TRIGMODE_SOFT	0x00	软件触发（属于内触发）
PXI8603_TRIGMODE_POST	0x01	硬件后触发（属于外触发）

TriggerType AD 触发类型。

常量名	常量值	功能定义
PXI8603_TRIGTYPE_EDGE	0x00	边沿触发
PXI8603_TRIGTYPE_PULSE	0x01	脉冲触发(电平)

TriggerDir AD 触发方向。它的选项值如下表：

常量名	常量值	功能定义
PXI8603_TRIGDIR_NEGATIVE	0x00	负向触发（低脉冲/下降沿触发）
PXI8603_TRIGDIR_POSITIVE	0x01	正向触发（高脉冲/上升沿触发）
PXI8603_TRIGDIR_POSIT_NEGAT	0x02	正负方向均有效

注明：PXI8603_TRIGDIR_POSIT_NEGAT 在边沿类型下，则都表示不管是上边沿还是下边沿均触发。而在电平类型下，无论正电平还是负电平均触发。

TriggerSource 触发源选择。

常量名	常量值	功能定义
PXI8603_TRIGSRC_ATR_AD	0x00	选择 ATR 作为触发源
PXI8603_TRIGSRC_DTR_AD	0x01	选择 DTR 作为触发源

TrigLevelVolt 触发电平（0～10000mV）。

TrigWindow 触发灵敏窗[1, 65535], 单位 25 纳秒

ClockSource AD 时钟源选择。它的选项值如下表：

常量名	常量值	功能定义
PXI8603_CLOCKSRC_IN	0x00	内部时钟定时触发
PXI8603_CLOCKSRC_OUT	0x01	外部时钟定时触发

当选择内时钟时，其 AD 定时触发时钟为板上时钟振荡器经分频得到。它的大小由 Frequency 参数决定。

当选择外时钟时：

当选择连续采集时(即 ADMode = PXI8603_ADMODE_SEQUENCE)，其 AD 定时触发时钟为外界时钟输 CLKIN 得到，而 Frequency 参数则自动失效。

但是当选择分组采集时(即 ADMode = PXI8603_ADMODE_GROUP)，外时钟则是每一组的触发时钟信号，而组内的触发频率则由 Frequency 参数决定，由此可见，此时外时钟触发周期必须大于每组总周期，否则紧跟其后的某一外时钟可能会被失效。

bClockOutput 成员变量所使用内部和外部时钟源。它的选项值如下表：

常量名	常量值	功能定义
XI8603_CLOCKOUT_DISABLE	0x00	禁止本卡上的自带时钟向外输出
PXI8603_CLOCKOUT_ENABLE	0x01	允许本卡上的自带时钟向外输出

GroundingMode AD 接地方式选择。它的选项值如下表：

常量名	常量值	功能定义
PXI8603_GNDMODE_SE	0x00	单端方式（SE:Single end）
PXI8603_GNDMODE_DI	0x01	双端方式（DI:Differential）

相关函数： CreateDevice LoadParaAD SaveParaAD ReleaseDevice

第二节、AD 状态参数结构（PXI8603_STATUS_AD）

Visual C++:

```
typedef struct _PXI8603_STATUS_AD
{
LONG bNotEmpty;
LONG bHalf;
LONG bDynamic_Overflow;
LONG bStatic_Overflow;
LONG bConverting;
LONG bTriggerFlag;
} PXI8603_STATUS_AD, *PPXI8603_STATUS_AD;
```

Visual Basic:

```
Type PXI8603__STATUS_AD
  bNotEmpty     As Long          '
  bHalf       As Long            '
  bDynamic_Overflow As Long '
  bStatic_Overflow As Long '
  bConverting       As Long      '
  bTriggerFlag      As Long      '
  End Type
```

此结构体主要用于查询 AD 的各种状态，GetDevStatusAD 函数使用此结构体来实时取得 AD 状态，以便同步各种数据采集和处理过程。

bNotEmpty AD 板载存储器 FIFO 的非空标志，=TRUE 表示存储器处在非空状态，即有可读数据，否则表示空。

bHalf AD 板载存储器 FIFO 的半满标志，=TRUE 表示存储器处在半满状态，即有至少有半满以上数据可读，否则表示在半满以下，可能有小于半满的数据可读。

bDynamic_Overflow 板载 FIFO 存储器的动态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出

bStatic_Overflow 板载 FIFO 存储器的静态溢出标志，= TRUE 已发生溢出， = FALSE 未发生溢出

bConverting AD 是否正在转换， =TRUE:表示正在转换， =FALS 表示转换完成

bTriggerFlag 触发标志, =TRUE 表示触发事件发生， =FALSE 表示触发事件未发生

相关函数： CreateDevice ReleaseDevice

第三节、DMA 状态参数结构（PXI8603_STATUS_AD）

Visual C++:

```
const int MAX_SEGMENT_COUNT = 64;
typedef struct _PXI8603_STATUS_DMA
{
   LONG iCurSegmentID;
   LONG bSegmentSts[PXI8603_MAX_SEGMENT_COUNT];
   LONG bBufferOverflow;
} PXI8603_STATUS_DMA, *PPXI8603_STATUS_DMA;
```

Visual Basic:

```
Public Const MAX_SEGMENT_COUNT = 64
Type PXI8603_STATUS_DMA
      iCurSegmentID   As Long
      bSegmentSts(0 To PXI8603_MAX_SEGMENT_COUNT - 1) As Long   '
      bBufferOverflow   As Long
End Type
```

iCurSegmentID 当前段缓冲 ID,表示 DMA 正在传输的缓冲区段。

bSegmentSts 各个缓冲区的新旧状态,=1 表示该相应缓冲区数据为新,否则为旧。

bBufferOverflow 返回溢出状态

相关函数： CreateDevice ReleaseDevice

第四节、DA 硬件参数介绍（PXI8603_PARA_DA）

Visual C++:

```
typedef struct _PXI8603_PARA_DA
  {
        LONG OutputRange;
        LONG Frequency;
        LONG LoopCount;
        LONG TriggerMode;
        LONG TriggerSource;
        LONG TriggerDir;
        LONG bSingleOut;
        LONG ClockSource;
        } PXI8603_PARA_DA, *PPXI8603_PARA_DA;
```

Visual Basic:

```
Type PXI8603_PARA_DA
```

```
        OutputRange As Long
        Frequency As Long
        LoopCount As Long
        TriggerMode As Long
        TriggerSource As Long
        TriggerDir As Long
        bSingleOut As Long
        ClockSource As Long
    End Type
```

此结构主要用于设定设备 DA 硬件参数值，用这个参数结构对设备进行硬件配置完全由 InitDeviceDA 函数自动完成。用户只需要对这个结构体中的各成员简单赋值即可。

OutputRange　模拟量输出量程选择。

常量名	常量值	功能定义
PXI8603_OUTPUT_0_P5000mV	0x00	0～5000mV
PXI8603_OUTPUT_0_P10000mV	0x01	0～10000mV
PXI8603_OUTPUT_N10000_P10000mV	0x02	±10000mV
PXI8603_OUTPUT_N5000_P5000mV	0x03	±5000mV

Frequency　DA 的采样频率，其取值范围为[1, 500000Hz]。

LoopCount　整过 RAM 的大循环次数,=0:无限循环, =n:表示 n 次循环(1<n<32768)。

TriggerMode　成员变量触发模式

常量名	常量值	功能定义
PXI8603_TRIGMODE_SINGLE	0x00	单次触发
PXI8603_TRIGMODE_CONTINUOUS	0x01	连续触发
PXI8603_TRIGMODE_STEPED	0x02	单步触发
PXI8603_TRIGMODE_BURST	0x03	紧急触发

TriggerSource　成员变量触发源

常量名	常量值	功能定义
PXI8603_TRIGSRC_SOFT_DA	0x00	软件触发
PXI8603_TRIGSRC_ATR_DA	0x01	ATR 硬件模拟触发
PXI8603_TRIGSRC_DTR_DA	0x02	DTR 硬件数字触发

TriggerDir　成员触发方向

常量名	常量值	功能定义
PXI8603_TRIGDIR_NEGATIVE	0x00	负向触发(低脉冲/下降沿触发)
PXI8603_TRIGDIR_POSITIVE	0x01	正向触发(高脉冲/上升沿触发)
PXI8603_TRIGDIR_POSIT_NEGAT	0x02	正负向触发(高/低脉冲或上升/下降沿触发)

bSingleOut　是否单点输出。

ClockSource　成员变量时钟源

常量名	常量值	功能定义
PXI8603_CLOCKSRC_IN	0x00	内部时钟
PXI8603_CLOCKSRC_OUT	0x01	外部时钟

第五节、DA 状态参数结构（PXI8603_STATUS_DA）

Visual C++：

```
typedef struct _PXI8603_STATUS_DA
    {
    LONG bEnable;
    LONG bTrigFlag;
    LONG bConverting;
    LONG nCurSegNum;
    LONG nCurSegAddr;
    LONG nCurLoopCount;
    LONG nCurSegLoopCount;
    } PXI8603_STATUS_DA, *PPXI8603_STATUS_DA;
```

Visual Basic：

```
Type PXI8603__STATUS_DA
      bEnable As Long                '
      bTrigFlag As Long              '
      bConverting As Long            '
      nCurSegNum As Long              '
      nCurSegAddr As Long            '
      nCurLoopCount As Long          '
      nCurSegLoopCount As Long       '
End Type
```

此结构体主要用于查询 DA 的各种状态，GetDevStatusDA 函数使用此结构体来实时取得 DA 状态，以便同步各种数据采集和处理过程。

bEnable　DA 使能启动标志，＝TRUE 表示 DA 已被使能, = FALSE 表示 DA 被禁止。

bTrigFlag　触发标志是否有效，=TRUE 表示触点标有效，　= FALSE 表示无效（即触发点未到）。

bConverting　DA 是否正在转换，　=TRUE:表示正在转换，　= FALS 表示转换完成。

nCurSegNum　可读取的 RAM 段号，取值为[0, SegmentCount-1], (注 SegmentCount 为 InitDeviceDA 函数的参数)。

nCurSegAddr　可读取的 RAM 段地址。

nCurLoopCount　当前总循环次数。

nCurSegLoopCount　当前段循环次数。

相关函数：　CreateDevice　GetDevStatusDA　ReleaseDevice

第五章 数据格式转换与排列规则

第一节、AD 原码 LSB 数据转换成电压值的换算方法

首先应根据设备实际位数屏蔽掉不用的高位，然后依其所选量程，按照下表公式进行换算即可。这里只以缓冲区 ADBuffer[]中的第 1 个点 ADBuffer[0]为例。

量程(mV)	计算机语言换算公式(ANSI C 语法)	Volt 取值范围（mV）
±10000mV	Volt = (20000.00/4096) * (ADBuffer[0]&0x0FFF) – 10000.00	[-10000, +9995.11]
±5000mV	Volt = (10000.00/4096)* (ADBuffer[0]&0x0FFF) – 5000.00	[-5000, +4997.55]
±2500mV	Volt = (5000.00/4096)* (ADBuffer[0]&0x0FFF) – 2500.00	[-2500, +2498.77]
0～10000mV	Volt = (10000.00/4096)* (ADBuffer[0]&0x0FFF)	[0, +9997.55]

下面举例说明各种语言的换算过程（以±10000mV 量程为例）

Visual C++：

```
Lsb = ADBuffer[0]&0x0FFF;
Volt = (20000.00/40964) * Lsb -10000.00;
```

Visual Basic：

```
Lsb = ADBuffer [0] And &0FFF
    Volt = (20000.00/4096) * Lsb – 10000.00
```

第二节、AD 采集函数的 ADBuffer 缓冲区中的数据排放规则

单通道采集，当通道总数首末通道相等时，假如此时首末通道=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（假如 FirstChannel=0，LastChannel=1）：

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（假如 FirstChannel=0，LastChannel=3）：

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的问题，尤其是在任意通道数采集时。否则，用户无将规则排放在缓冲区中的各通道数据正确分离出来。那怎样正确处理呢？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，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断的多通道数据链。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 ReadDeviceProAD_X 函数读回，即便不考虑是否能一次读完的问题，仅对于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，而且还高效呢？ 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。

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

第三节、AD 测试应用程序创建并形成的数据文件格式

首先该数据文件从始端 0 字节位置开始往后至第 HeadSizeBytes 字节位置宽度属于文件头信息，而从 HeadSizeBytes 开始才是真正的 AD 数据。HeadSizeBytes 的取值通常等于本头信息的字节数大小。文件头信息包含的内容如下结构体所示。对于更详细的内容请参考 Visual C++高级演示工程中的 UserDef.h 文件。

```
typedef struct _FILE_HEADER
{
    LONG HeadSizeBytes;                 // 文件头信息长度
    LONG FileType;                      // 该设备数据文件共有的成员
    LONG BusType;                       // 设备总线类型(DEFAULT_BUS_TYPE)
    LONG DeviceNum;                     // 该设备的编号(DEFAULT_DEVICE_NUM)
    LONG HeadVersion;                   // 头信息版本(D15-D8=Major,D7-D0=Minijor) = 1.0
    LONG VoltBottomRange;               // 量程下限(mV)
    LONG VoltTopRange;                  // 量程上限(mV)
    LONG StaticOverFlow;                // 同批文件识别码
    PXI8603_PARA_AD ADPara;             // 保存硬件参数
    LONG HeadEndFlag;                   // 头信息结束位
} FILE_HEADER，*PFILE_HEADER;
```

AD 数据的格式为 12 位二进制格式，它的排放规则与在 ADBuffer 缓冲区排放的规则一样，即每 12 位二进制(字)数据对应一个 12 位 AD 数据。您只需要先开辟一个 12 位整型数组或缓冲区，然后将磁盘数据从指定位置(即双字节对齐的某个位置)读入数组或缓冲区，然后访问数组中的每个元素，即是对相应 AD 数据的访问。

第四节、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	Lsb 取值范围
0～5000mV	Lsb=Volt/(5000.00/4096)	[0，4095]
0～10000mV	Lsb=Volt/(10000.00/4096)	[0，4095]
±10000mV	Lsb=Volt/(20000.00/4096)+2048	[0，4095]
±5000mV	Lsb=Volt/(10000.00/4096)+2048	[0，4095]

第六章 上层用户函数接口应用实例

第一节、简易程序演示说明

一、怎样使用 ReadDeviceProAD Npt 函数直接取得 AD 数据

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[非空方式]

二、怎样使用 ReadDeviceProAD Half 函数直接取得 AD 数据

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[半满方式]

三、怎样使用 DMA 方式取得 AD 数据

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[DMA 方式]

四、怎样使用 WriteDeviceDA 函数取得 AD 数据

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[DA 方式]

五、怎样使用 GetDeviceDI 函数进行更便捷式数字量输入操作

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[DIO...]

六、怎样使用 SetDeviceDO 函数进行更便捷式数字量输出操作

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]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[简易代码演示]丨[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 PXI8603.h 和 ADDoc.cpp)。

[程序]丨[阿尔泰测控演示系统]丨[PXI8603 AD、DA 、DIO(V6.01.02)]丨[Microsoft Visual C++]丨[高级代码演示]

其默认存放路径为：系统盘\ART\PXI8603\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 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“PXI8603_”）

函数名	函数功能	备注
① PXI 总线内存映射寄存器操作函数		
GetDeviceAddr	取得指定 PXI 设备寄存器操作基地址	底层用户
GetDeviceBar	取得指定的指定设备寄存器组 BAR 地址	底层用户
GetDeviceVersion	获取设备固件及程序版本	底层用户
WriteRegisterByte	以字节(8Bit)方式写寄存器端口	底层用户
WriteRegisterWord	以字(16Bit)方式写寄存器端口	底层用户
WriteRegisterULong	以双字(32Bit)方式写寄存器端口	底层用户
ReadRegisterByte	以字节(8Bit)方式读寄存器端口	底层用户
ReadRegisterWord	以字(16Bit)方式读寄存器端口	底层用户
ReadRegisterULong	以双字(32Bit)方式读寄存器端口	底层用户
② ISA 总线 I/O 端口操作函数		
WritePortByte	以字节(8Bit)方式写 I/O 端口	用户程序操作端口
WritePortWord	以字(16Bit)方式写 I/O 端口	用户程序操作端口
WritePortULong	以无符号双字(32Bit)方式写 I/O 端口	用户程序操作端口
ReadPortByte	以字节(8Bit)方式读 I/O 端口	用户程序操作端口
ReadPortWord	以字(16Bit)方式读 I/O 端口	用户程序操作端口
ReadPortULong	以无符号双字(32Bit)方式读 I/O 端口	用户程序操作端口
③ 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	用于线程同步或中断
④ 文件对象操作函数		
CreateFileObject	初始设备文件对象	底层用户
WriteFile	保存用户空间中数据	底层用户
ReadFile	读数据	底层用户
SetFileOffset	设置文件偏移指针	底层用户
GetFileLength	取得指定文件长度（字节）	底层用户
ReleaseFile	释放设备文件对象	底层用户
GetDiskFreeBytes	获得指定盘符的磁盘空间(注意使用 64 位变量)	底层用户
⑤ 辅助函数		
kbhit	探测用户是否有击键动作(在控制台应用程序 Console 中有效)	底层用户
getch	等待并获取用户击键值(在控制台应用程序 Console 中有效)	底层用户
DelayTimeUs	探测时间是否有延迟	底层用户
SaveParaInt	将整型数据从注册表中回读出来(Device-x\Others)	底层用户
SaveParaString	将字符串数据保存到注册表中(Device-x\Others)	底层用户
LoadParaString	将字符串数据从注册表中回读出来(Device-x\Others)	底层用户
GetLastErrorEx	从错误信息库中获得指定函数的最后一次错误信息	底层用户
RemoveLastErrorEx	从错误信息库中移除指定函数的最后一次错误信息	底层用户

第二节、PXI 内存映射寄存器操作函数原型说明

♦ 取得指定内存映射寄存器的线性地址

函数原型：

Visual C++:

```
BOOL GetDeviceAddr(HANDLE hDevice,
                   PULONG MemCPLDBase,
                   PULONG MemADBuffer,
                   PULONG MemDA0Buffer,
                   PULONG MemDA1Buffer,
                   int RegisterID =0 );
```

Visual Basic:

```
Declare Function GetDeviceAddr Lib "PXI8603" (ByVal hDevice As Long, _
                                              ByRef MemCPLDBase As Long,
                                              ByRef MemADBuffer As Long,
                                              ByRef MemDA0Buffer As Long,
                                              ByRef MemDA1Buffer As Long，
                                              ByVal RegisterID As Integer) As Boolean
```

功能：取得 PXI 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。

MemCPLDBase　MemADBuffer　MemDA0Buffer　MemDA1Buffer　返回指定映射寄存器的线性地址

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，我们为用户加以声明。本设备的寄存器组 ID 定义如下：

常量名	常量值	功能定义
PXI8603_REG_MEM_PLXCHIP	0x0000	0 号寄存器对应 PLX 芯片所使用的内存模式基地址（使用 LinearAddr）
PXI8603_REG_IO_PLXCHIP	0x0001	1 号寄存器对应 PLX 芯片所使用的 IO 模式基地址（PhysAddr）

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　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

♦ 取得指定的指定设备寄存器组 BAR 地址

函数原型：

Visual C++:

```
BOOL GetDeviceBar (HANDLE hDevice,
                   ULONG pulPXIBar[6])
```

Visual Basic:

```
Declare Function GetDeviceBar Lib"PXI8603" (_
                                  ByVal hDevice As Long, _
                                  ByRef pulPCIBar As Long) As Boolean
```

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。

pulPXIBar　返回 PXI BAR 所有地址。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数： CreateDevice ReleaseDevice

◆ **获取设备固件及程序版本**

函数原型：

Visual C++:

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

Visual Basic:

```
Declare Function GetDeviceVersion Lib"PXI8603" (ByVal hDevice As Long,
                                                 ByRef pulFmwVersion As Long,
                                                 ByRef pulDriverVersion As Long) As Boolean
```

功能：获取设备固件及程序版本。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

pulFmwVersion 指针参数，用于取得固件版本。

pulDriverVersion 指针参数，用于取得驱动版本。

返回值：如果执行成功，则返回 TRUE，否则会返回 FALSE。

相关函数: CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

◆ **以单字节（即 8 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
BOOL WriteRegisterByte (HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        BYTE Value)
```

Visual Basic:

```
Declare Function WriteRegisterByte Lib "PXI8603" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Byte) As Boolean
```

功能：以单字节（即 8 位）方式写 PXI 内存映射寄存器。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 创建。

LinearAddr PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes 相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterByte 函数所访问的映射寄存器的内存单元。

Value 输出 8 位整数。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数: CreateDevice GetDeviceAddr WriteRegisterByte WriteRegisterWord WriteRegisterULong ReadRegisterByte ReadRegisterWord ReadRegisterULong ReleaseDevice

Visual C++程序举例:

```
:
HANDLE hDevice;
```

```
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox "取得设备地址失败…";
}
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterByte(hDevice, LinearAddr, OffsetBytes, 0x20); // 往指定映射寄存器单元写入 8 位的十六进制数据 20
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterByte( hDevice, LinearAddr, OffsetBytes, &H20)
ReleaseDevice(hDevice)
    :
```

- **以双字节（即 16 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
BOOL WriteRegisterWord (HANDLE hDevice,
                        ULONG LinearAddr,
                        ULONG OffsetBytes,
                        WORD Value)
```

Visual Basic:

```
Declare Function WriteRegisterWord Lib "PXI8603" (ByVal hDevice As Long, _
                                                  ByVal LinearAddr As Long, _
                                                  ByVal OffsetBytes As Long, _
                                                  ByVal Value As Integer) As Boolean
```

功能：以双字节（即 16 位）方式写 PXI 内存映射寄存器。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

LinearAddr　PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes　相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterWord 函数所访问的映射寄存器的内存单元。

Value　输出 16 位整型值。

返回值：无。

相关函数:　CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
```

```
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
WriteRegisterWord(hDevice, LinearAddr, OffsetBytes, 0x2000); //往指定映射寄存器单元写入 16 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes=100
WriteRegisterWord( hDevice, LinearAddr, OffsetBytes, &H2000)
ReleaseDevice(hDevice)
    :
```

♦ **以四字节（即 32 位）方式写 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
BOOL WriteRegisterULong (HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes
                         ULONG Value)
```

Visual Basic:

```
Declare Function WriteRegisterULong Lib"PXI8603" (_
                                    ByVal hDevice As Long, _
                                    ByVal LinearAddr As Long, _
                                    ByVal OffsetBytes As Long, _
                                    ByVal Value As Long) As Boolean
```

功能：以四字节（即 32 位）方式写 PXI 内存映射寄存器。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

LinearAddr　PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes　相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterULong 函数所访问的映射寄存器的内存单元。

Value　输出 32 位整型值。

返回值：若成功，返回 TRUE，否则返回 FALSE。

相关函数:　CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
hDevice = CreateDevice(0)
if (!GetDeviceAddr(hDevice, &LinearAddr,  &PhysAddr, 0) )
{
    AfxMessageBox “取得设备地址失败…”;
}
OffsetBytes=100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
```

```
WriteRegisterULong(hDevice, LinearAddr, OffsetBytes, 0x20000000);  // 往指定映射寄存器单元写入 32 位的十六进制数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
hDevice = CreateDevice(0)
GetDeviceAddr ( hDevice, LinearAddr, PhysAddr, 0)
OffsetBytes = 100
WriteRegisterULong ( hDevice, LinearAddr, OffsetBytes, &H20000000)
ReleaseDevice (hDevice)
    :
```

◆ **以单字节（即 8 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
BYTE ReadRegisterByte (HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterByte Lib"PXI8603" (ByVal hDevice As Long,
                                                ByVal LinearAddr As Long,
                                                ByVal OffsetBytes As Long) As Byte
```

功能：以单字节（即 8 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

LinearAddr　PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes　相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 ReadRegisterByte 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 8 位数据。

相关函数:　CreateDevice　GetDeviceAddr　WriteRegisterByte
WriteRegisterWord　WriteRegisterULong　ReadRegisterByte
ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
BYTE   Value;
hDevice = CreateDevice(0);  // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterByte(hDevice, LinearAddr, OffsetBytes);  // 从指定映射寄存器单元读入 8 位数据
ReleaseDevice( hDevice );  // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Byte
```

```
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterByte( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

◆ **以双字节（即 16 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
WORD ReadRegisterWord (HANDLE hDevice,
                       ULONG LinearAddr,
                       ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterWord Lib"PXI8603" (_
                                    ByVal hDevice As Long, _
                                    ByVal LinearAddr As Long,_
                                     ByVal OffsetBytes As Long) As Integer
```

功能：以双字节（即 16 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

pbLinearAddr　PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes　相对于 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 ReadRegisterWord 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 16 位数据。

相关函数: CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr, PhysAddr, OffsetBytes;
WORD Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100; // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterWord(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 16 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Word
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterWord( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

♦ **以四字节（即 32 位）方式读 PXI 内存映射寄存器的某个单元**

函数原型:

Visual C++:

```
ULONG ReadRegisterULong (HANDLE hDevice,
                         ULONG LinearAddr,
                         ULONG OffsetBytes)
```

Visual Basic:

```
Declare Function ReadRegisterULong Lib "PXI8603" (_
                                   ByVal hDevice As Long, _
                                   ByVal LinearAddr As Long, _
                                   ByVal OffsetBytes As Long) As Long
```

功能：以四字节（即 32 位）方式读 PXI 内存映射寄存器的指定单元。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 创建。

LinearAddr　PXI 设备内存映射寄存器的线性基地址，它的值应由 GetDeviceAddr 确定。

OffsetBytes　相对与 LinearAddr 线性基地址的偏移字节数，它与 LinearAddr 两个参数共同确定 WriteRegisterULong 函数所访问的映射寄存器的内存单元。

返回值：返回从指定内存映射寄存器单元所读取的 32 位数据。

相关函数:　CreateDevice　GetDeviceAddr　WriteRegisterByte　WriteRegisterWord　WriteRegisterULong　ReadRegisterByte　ReadRegisterWord　ReadRegisterULong　ReleaseDevice

Visual C++程序举例:

```
    :
HANDLE hDevice;
ULONG LinearAddr,   PhysAddr, OffsetBytes;
ULONG   Value;
hDevice = CreateDevice(0);   // 创建设备对象
GetDeviceAddr(hDevice, &LinearAddr, &PhysAddr, 0);    // 取得 PXI 设备 0 号映射寄存器的线性基地址
OffsetBytes = 100;  // 指定操作相对于线性基地址偏移 100 个字节数位置的单元
Value = ReadRegisterULong(hDevice, LinearAddr, OffsetBytes);   // 从指定映射寄存器单元读入 32 位数据
ReleaseDevice( hDevice );   // 释放设备对象
    :
```

Visual Basic 程序举例:

```
    :
Dim hDevice As Long
Dim LinearAddr, PhysAddr, OffsetBytes As Long
Dim Value As Long
hDevice = CreateDevice(0)
GetDeviceAddr( hDevice, Linearddr, PhysAddr, 0)
OffsetBytes = 100
Value = ReadRegisterULong( hDevice, LinearAddr, OffsetBytes)
ReleaseDevice(hDevice)
    :
```

第三节、I/O 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。

♦ **以单字节(8Bit)方式写 I/O 端口**

Visual C++:

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

Visual Basic:

```
Declare Function WritePortByte Lib "PXI8603" (ByVal hDevice As Long,
                                              ByVal nPort As Integer,
                                              ByVal Value As Byte) As Boolean
```

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　指定寄存器的物理基地址。

OffsetBytes　相对于物理基地址的偏移位置(字节)。

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　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

◆ **以双字(16Bit)方式写 I/O 端口**

Visual C++:

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

Visual Basic:

```
Declare Function WritePortWord Lib"PXI8603" (_
                                ByVal hDevice As Long,_
                                ByVal nPort As Integer,_
                                ByVal Value As Integer) As Boolean
```

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　指定寄存器的物理基地址。

OffsetBytes　相对于物理基地址的偏移位置(字节)。

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　WritePortByte　WritePortWord　WritePortULong　ReadPortByte　ReadPortWord

◆ **以四字节(32Bit)方式写 I/O 端口**

Visual C++:

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

Visual Basic:

```
Declare Function WritePortULong Lib"PXI8603" (_
                                 ByVal hDevice As Long, _
                                 ByVal nPort As Long,_
                                 ByVal Value As Long) As Boolean
```

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　指定寄存器的物理基地址。

OffsetBytes　相对于物理基地址的偏移位置(字节)。
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　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以单字节(8Bit)方式读 I/O 端口**

Visual C++:

```
BYTE ReadPortByte (HANDLE hDevice,
                   UINT nPort）
```

Visual Basic:

```
Declare Function ReadPortByte Lib "PXI8603" (_
                                  ByVal hDevice As Long,_
                                  ByVal nPort As Integer) As Byte
```

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　指定寄存器的物理基地址。
OffsetBytes　相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以双字节(16Bit)方式读 I/O 端口**

Visual C++:

```
WORD ReadPortWord (HANDLE hDevice,
                   UINT nPort）
```

Visual Basic:

```
Declare Function ReadPortWord Lib "PXI8603" (_
                                  ByVal hDevice As Long,_
                                  ByVal nPort As Integer) As Integer
```

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　指定寄存器的物理基地址。
OffsetBytes　相对于物理基地址的偏移位置(字节)。
返回值：返回由 nPort 指定的端口的值。
相关函数： CreateDevice　WritePortByte　WritePortWord
WritePortULong　ReadPortByte　ReadPortWord

- **以四字节(32Bit)方式读 I/O 端口**

Visual C++:

```
ULONG ReadPortULong (HANDLE hDevice,
                     UINT nPort）
```

Visual Basic:

```
Declare Function ReadPortULong Lib"PXI8603" (_
                                  ByVal hDevice As Long,_
                                  ByVal nPort As Integer) As Long
```

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　指定寄存器的物理基地址。

OffsetBytes 相对于物理基地址的偏移位置(字节)。

返回值：返回由 nPort 指定端口的值。

相关函数： CreateDevice WritePortByte WritePortWord WritePortULong ReadPortByte ReadPortWord

第四节、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）

♦ 创建内核系统事件

函数原型：

Visual C++:

```
HANDLE CreateSystemEvent (void)
```

Visual Basic:

```
Declare Function CreateSystemEvent Lib "PXI8603" () As Long
```

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)。

相关函数： CreateDevice ReleaseSystemEvent ReleaseDevice

♦ 释放内核系统事件

函数原型：

Visual C++:

```
BOOL ReleaseSystemEvent (HANDLE hEvent)
```

Visual Basic:

```
Declare Function ReleaseSystemEvent Lib "PXI8603" (ByVal hEvent As Long) As Boolean
```

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。

相关函数： CreateDevice ReleaseSystemEvent ReleaseDevice

第五节、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 "PXI8603" (ByVal hDevice As Long, _
                                                 ByVal strFileName As String, _
                                                 ByVal Mode As Integer ) As Long
```

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 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:\\PXI8603\\Data.Dat”，在 Basic 中，其语法格式如：“C:\PXI8603\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
PXI8603_modeRead	0x0000	只读文件方式

PXI8603_modeWrite	0x0001	只写文件方式
PXI8603_modeReadWrite	0x0002	既读又写文件方式
PXI8603_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
PXI8603_typeText	0x4000	以文本方式操作文件

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"PXI8603" (ByVal hFileObject As Long, _
                                         ByRef pDataBuffer As Long, _
                                         ByVal nWriteSizeBytes As Long ) As Boolean
```

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 "PXI8603" (ByVal hFileObject As Long, _
                                         ByRef pDataBuffer As Long, _
                                         ByVal nOffsetBytes As Long, _
                                         ByVal nReadSizeBytes As Long) As Boolean
```

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)

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)

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)

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++:

ULONGLONG GetDiskFreeBytes (LPCTSTRstrDiskName)

功能：取得指定磁盘的可用剩余空间(以字为单位)。

参数：DiskName 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。

相关函数： CreateDevice ReleaseDevice

第六节、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)

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\PXI8603\Device-0\Others。

参数：

hDevice 设备对象句柄，它应由 CreateDevice 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)
```

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\PXI8603\Device-0\Others。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 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)
```

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\PXI8603\Device-0\Others。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 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)
```

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\PXI8603\Device-0\Others。

参数：

hDevice　设备对象句柄，它应由 CreateDevice 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

第七节、其他函数原型说明

◆ **怎样获取驱动函数错误信息**

函数原型：

Visual C++:

DWORD GetLastErrorEx (LPCTSTR strFuncName ,
LPTSTR strErrorMsg)

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 PXI8603_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PXI8603_InitDeviceAD”，否则得不到相应信息。

strErrorMsg 取得指定函数的错误信息串，该串为字符数组，其分配空间最好不要小于 256 字节。

返回值：返回错误码。

相关函数：无。

- **移除驱动函数错误信息**

函数原型：

Visual C++:

BOOL RemoveLastErrorEx (LPCTSTR strFuncName）

功能：从错误信息库中移除指定函数的最后一次错误信息。

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 PXI8603_InitDeviceAD 出现错误，此时调用该函数时，此参数必须为“PXI8603_InitDeviceAD”，否则得不到相应信息。

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：无。

- **探测用户是否有击键动作(在控制台应用程序 Console 中有效)**

函数原型：

Visual C++:

BOOL kbhit(void);

功能：探测用户是否有击键动作(在控制台应用程序 Console 中有效)。

相关函数：无。

- **等待并获取用户击键值(在控制台应用程序 Console 中有效)**

函数原型：

Visual C++:

BOOL getch(void);

功能：等待并获取用户击键值(在控制台应用程序 Console 中有效)。

相关函数：无。

- **探测击键时间是否有延迟**

函数原型：

Visual C++:

BOOL DelayTimeUs(HANDLE hDevice, LONG nTimeUs);

功能：探测击键时间是否有延迟。

相关函数：无。