

# PCICAN113 WIN2000/XP 驱动程序使用说明书

请您务必阅读《[使用纲要](#)》，他会使您事半功倍！

## 目 录

|                                             |    |
|---------------------------------------------|----|
| PCICAN113 WIN2000/XP驱动程序使用说明书.....          | 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 PCICAN设备专用函数接口介绍.....                   | 2  |
| 第一节、设备驱动接口函数列表（每个函数省略了前缀“PCICAN113_”） ..... | 2  |
| 第二节、设备对象管理函数原型说明.....                       | 4  |
| 第三节、寄存器操作函数原型说明.....                        | 10 |
| 第三章 硬件参数结构 .....                            | 12 |
| 第一节、CAN设备初始化结构介绍（CAN_INIT_PARA_） .....      | 12 |
| 第二节、CAN帧结构（CAN_FRAME_） .....                | 13 |

### 提醒用户：

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

有关设备驱动安装和产品二次发行请参考 PCICAN113Inst.doc 文档。

## 第一章 版权信息与命名约定

### 第一节、版权信息

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

### 第二节、命名约定

一、为简化文字内容，突出重点，本文中提到的函数名通常为基本功能名部分，其前缀设备名如 PCICANxxxx\_则被省略。如 PCICAN113\_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 PCICAN 设备专用函数接口介绍

### 第一节、设备驱动接口函数列表（每个函数省略了前缀“PCICAN113\_”）

| 函数名                           | 函数功能          | 备注 |
|-------------------------------|---------------|----|
| <b>① 设备对象操作函数</b>             |               |    |
| <a href="#">CreateDevice</a>  | 创建设备对象        |    |
| <a href="#">ReleaseDevice</a> | 释放设备对象        |    |
| <a href="#">ResetDevice</a>   | 复位 CAN 卡为出厂状态 |    |
| <a href="#">GetDeviceID</a>   | 取得 CAN 卡设备序列号 |    |
| <a href="#">ListDevices</a>   | 列表设备          |    |
| <a href="#">InitCAN</a>       | 初始化 CAN 卡     |    |
| <a href="#">SetTimer</a>      | CAN 卡定时器设置    |    |

|                                   |                 |  |
|-----------------------------------|-----------------|--|
| <a href="#">SendFrameSingle</a>   | 发送单帧操作          |  |
| <a href="#">SendFrameMultiple</a> | 发送多帧操作          |  |
| <a href="#">ReceiveFrame</a>      | 接收帧操作           |  |
| <a href="#">GetErrInfo</a>        | 读取错误信息          |  |
| <a href="#">SetACRID</a>          | ID 接受码寄存器设置     |  |
| <a href="#">SetAMRID</a>          | ID 屏蔽码寄存器设置     |  |
| <b>② 寄存器操作</b>                    |                 |  |
| <a href="#">ReadReg</a>           | 读 SJA1000 寄存器内容 |  |
| <a href="#">WriteReg</a>          | 写 SJA1000 寄存器   |  |

## 使用需知

### Visual C++ & C++Builder:

首先将 PCICAN113.h 和 PCICAN113.lib 两个驱动库文件从相应的演示程序文件夹下复制到您的源程序文件夹中，然后在您的源程序头部添加如下语句，以便将驱动库函数接口的原型定义信息和驱动接口导入库(PCICAN113.lib)加入到您的工程中。

```
#include "PCICAN113.H"
```

在 VC 中，为了使用方便，避免重复定义和包含，您最好将以上语句放在 StdAfx.h 文件。一旦完成了以上工作，那么使用设备的驱动程序接口就跟使用 VC/C++Builder 自身的各种函数，其方法一样简单，毫无二别。

关于 PCICAN113.h 和 PCICAN113.lib 两个文件均可在演示程序文件夹下面找到。

### Visual Basic:

首先将 PCICAN113.Bas 驱动模块头文件从 VB 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 VB 工程中。其方法是选择 VB 编程环境中的工程(Project)菜单，执行其中的"添加模块"(Add Module)命令,在弹出的对话框中选择 PCICAN113.Bas 模块文件即可，一旦完成以上工作后，那么使用设备的驱动程序接口就跟使用 VB 自身的各种函数，其方法一样简单，毫无二别。

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

### Delphi:

首先将 PCICAN113.Pas 驱动模块头文件从 Delphi 的演示程序文件夹下复制到您的源程序文件夹中，然后将此模块文件加入到您的 Delphi 工程中。其方法是选择 Delphi 编程环境中的 View 菜单,执行其中的"Project Manager"命令,在弹出的对话框中选择\*.exe 项目，再单击鼠标右键，最后 Add 指令，即可将 PCICAN113.Pas 单元模块文件加入到工程中。或者在 Delphi 的编程环境中的 Project 菜单中，执行 Add To Project 命令，然后选择\*.Pas 文件类型也能实现单元模块文件的添加。最后请在使用驱动程序接口的源程序文件中的头部的 Uses 关键字后面的项目中加入：“PCICAN113”。如：

#### uses

```
Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms, Dialogs,  
PCICAN113; // 注意： 在此加入驱动程序接口单元 PCICAN113
```

### LabView / CVI:

LabVIEW 是美国国家仪器公司(National Instrument)推出的一种基于图形开发、调试和运行程序的集成化环境，是目前国际上唯一的编译型的图形化编程语言。在以 PC 机为基础的测量和工控软件中，LabVIEW 的市场普及率仅次于 C++/C 语言。LabVIEW 开发环境具有一系列优点，从其流程图式的编程、

不需预先编译就存在的语法检查、调试过程使用的数据探针，到其丰富的函数功能、数值分析、信号处理和设备驱动等功能，都令人称道。关于 LabView/CVI 的驱动程序接口的详细说明请参考其演示源程序。

## 第二节、设备对象管理函数原型说明

### ◆ 创建设备对象函数

函数原型:

**Visual C++ & C++ Builder:**

**HANDLE CreateDevice(UINT ID = 0)**

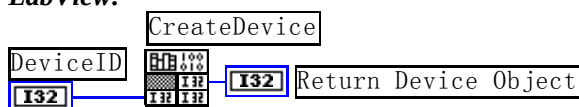
**Visual Basic:**

**Declare Function CreateDevice Lib "PCICAN113" (Optional ByVal ID As Long = 0) As Long**

**Delphi:**

**Function CreateDevice(ID:LongWord = 0):Integer; StdCall; External 'PCICAN113' Name 'CreateDevice';**

**LabView:**



**功能:** 该函数负责创建设备对象，并返回其设备对象句柄。

**参数:**

ID 设备逻辑 ID( Identifier )标识号。当向同一个 Windows 系统中加入若干相同类型的设备时，系统将以该设备的“基本名称”与 ID 标识值为名称后缀的标识符来确认和管理该设备。比如若用户往 Windows 系统中加入第一个 PCICAN113 AD 模板时，系统则以“PCICAN113”作为基本名称，再以 ID 的初值组合成该设备的标识符“PCICAN113-0”来确认和管理这第一个设备，若用户接着再添加第二个 PCICAN113 AD 模板时，则系统将以“PCICAN113-1”来确认和管理第二个设备，若再添加，则以此类推。所以当用户要创建设备句柄管理和操作第一个 CAN 设备时，ID 应置 0，第二应置 1，也以此类推。默认值为 0。

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

**备注:** 创建完成后，如果释放 CAN 需要用 PCICAN113\_CreateDevice 来关闭 CAN 卡。

**相关函数:** [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++ & C++ Builder:**

**BOOL ReleaseDevice(HANDLE hDevice)**

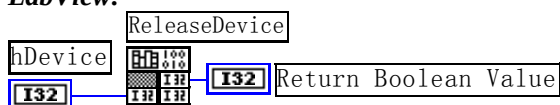
**Visual Basic:**

**Declare Function ReleaseDevice Lib "PCICAN113" (ByVal hDevice As Long ) As Boolean**

**Delphi:**

**Function ReleaseDevice(hDevice : Integer):Boolean; StdCall; External 'PCICAN113' Name 'ReleaseDevice';**

**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 GetErrInfo 捕获错误码。

**相关函数:** [CreateDevice](#)

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

**◆ 复位 PCICAN 卡**

函数原型:

**Visual C++ & C++Builder:**

[BOOL ResetDevice \(HANDLE hDevice\)](#)

**Visual Basic:**

[Declare Function ResetDevice Lib "PCICAN113" \(ByVal hDevice As Long \) As Boolean](#)

**Delphi:**

[Function ResetDevice \(hDevice : Integer\):Boolean; StdCall; External 'PCICAN113' Name 'ResetDevice';](#)

**LabView:**

请参考相关演示程序。

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

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

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

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

**◆ 取得当前设备对象句柄指向的设备所在的设备 ID**

函数原型:

**Visual C++ & C++Builder:**

[BOOL GetDeviceID \(HANDLE hDevice, UCHAR ucID\[\]\)](#)

**Visual Basic:**

[Declare Function GetDeviceIDLib "PCICAN113" \(ByVal hDevice As Long, \\_  
ByRef ucID As Byte\) As Boolean](#)

**Delphi:**

[Function GetDeviceID \(hDevice : Integer;  
ucID: Pointer\): Boolean;  
StdCall; External 'PCICAN113' Name 'GetDeviceID';](#)

**LabView:**

请参考相关演示程序。

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

**参数:**

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

ucID 设备序列号。

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

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

**◆ 列表计算机系统中所有 PCICAN113 设备各种配置信息**

函数原型:

**Visual C++ & C++Builder:**

BOOL ListDevices (HANDLE hDevice,  
PLONG plCount)

**Visual Basic:**

Declare Function ListDevices Lib "PCICAN113" (ByVal hDevice As Long, \_  
ByRef plCount As Long) As Boolean

**Delphi:**

Function ListDevices (hDevice : Integer;  
plCount : Pointer): Boolean;  
StdCall; External 'PCICAN113' Name 'ListDevices';

**LabVIEW:**

请参考相关演示程序。

**功能:** 列表系统中 PCICAN113 的硬件配置信息。

**参数:**

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

plCount 设备数量。

**返回值:** 若成功, 则弹出对话框控件列表所有 PCICAN113 设备的配置情况。

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

◆ **初始化设备对象**

函数原型:

**Visual C++ & C++Builder:**

BOOL InitCAN( HANDLE hDevice,  
PCAN\_INIT\_PARA pPara,  
LONG lCANChannel = 0)

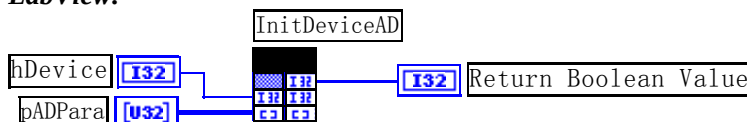
**Visual Basic:**

Declare Function InitCAN Lib "PCICAN113" (ByVal hDevice As Long, \_  
ByRef pPara As PCAN\_INIT\_PARA, \_  
Optional ByVal lCANChannel As Long = 0) As Boolean

**Delphi:**

Function InitCAN( hDevice : Integer;  
pPara: PCAN\_INIT\_PARA;  
lCANChannel : LongInt = 0):Boolean;  
StdCall; External 'PCICAN113' Name 'InitCAN';

**LabView:**



**功能:** 它负责初始化设备对象, 为设备操作就绪有关工作。

**参数:**

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

pPara 设备对象参数结构, 它决定了设备对象的各种状态及工作方式。

lCANChannel CAN 通道号。

**返回值:** 如果初始化设备对象成功, 则返回 TRUE, 且设备被启动。否则返回 FALSE, 用户可用 GetErrInfo 捕获当前错误码, 并加以分析。



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

#### ◆ CAN 卡定时器设置函数

函数原型:

**Visual C++ & C++ Builder:**

```
BOOL SetTimer (HANDLE hDevice,
               WORD wTimer0,
               WORD wTimer1)
```

**Visual Basic:**

```
Declare Function SetTimer Lib "PCICAN113" (ByVal hDevice As Long, _
                                           ByVal wTimer0 As Integer, _
                                           ByVal wTimer1 As Integer) As Boolean
```

**Delphi:**

```
Function SetTimer ( hDevice : Integer;
                   wTimer0 : Word;
                   wTimer1 : Word): Boolean;
StdCall; External 'PCICAN113' Name ' SetTimer ';
```

**LabVIEW:**

请参考相关演示程序。

**功能:** 对设定 CAN 总线速率。

**参数:**

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

wTimer0 定时器 0(寄存器 3)。

wTimer1 定时器 1(寄存器 6)。

**返回值:** 若成功, 返回 TRUE, 否则返回 FALSE。

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

#### ◆ 发送单帧操作

函数原型:

**Visual C++ & C++ Builder:**

```
LONG SendFrameSingle(HANDLE hDevice,
                     PCAN_FRAME psCanFrame,
                     LONG lCANChannel = 0)
```

**Visual Basic:**

```
Declare Function SendFrameSingle Lib "PCICAN113" (ByVal hDevice As Long, _
                                                  ByRef psCanFrame As PCAN_FRAME, _
                                                  Optional ByVal lCANChannel As Long = 0 ) As Long
```

**Delphi:**

```
Function SendFrameSingle ( hDevice : Integer;
                          psCanFrame: Pointer;
                          lCANChannel : LongInt = 0): LongInt;
StdCall; External 'PCICAN113' Name ' SendFrameSingle ';
```

**LabView:**

请参考相关演示程序。

**功能：**发送单帧操作。

**参数：**

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

psCanFrame CAN 帧结构体。

ICANChannel CAN 通道号。

**返回值：**-1 表示出错，0 表示需要重发（表示缓冲区已满）。

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

#### ◆ 发送多帧操作

函数原型：

**Visual C++ & C++Builder:**

```
LONG SendFrameMultiple(HANDLE hDevice,
                       PCAN_FRAME psCanFrame,
                       LONG lFrameCount = 1,
                       LONG lCANChannel = 0)
```

**Visual Basic:**

```
Declare Function SendFrameMultiple Lib "PCICAN113" (ByVal hDevice As Long, _
                                                    ByRef psCanFrame As PCAN_FRAME, _
                                                    Optional ByVal lFrameCount As Long = 1, _
                                                    Optional ByVal lCANChannel As Long = 0) As Long
```

**Delphi:**

```
Function SendFrameMultiple ( hDevice : Integer;
                             psCanFrame: Pointer;
                             lFrameCount : LongInt = 0;
                             lCANChannel : LongInt = 0): LongInt;
StdCall; External 'PCICAN113' Name 'SendFrameMultiple';
```

**LabView:**

请参考相关演示程序。

**功能：**发送多帧操作。

**参数：**

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

psCanFrame CAN 帧结构体。

lFrameCount 帧数量。

ICANChannel CAN 通道号。

**返回值：**-1 表示出错，0 表示需要重发（表示缓冲区已满）。

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

#### ◆ 接收帧操作

函数原型：

**Visual C++ & C++Builder:**

```
LONG ReceiveFrame(HANDLE hDevice,
                  PCAN_FRAME pCanFrame,
                  LONG lCANChannel = 0)
```

**Visual Basic:**

```
Declare Function ReceiveFrame Lib "PCICAN113" (ByVal hDevice As Long, _
```

ByRef pCanFrame As PCAN\_FRAME, \_  
Optional ByVal lCANChannel As Long = 0 ) As Long

**Delphi:**

Function ReceiveFrame (hDevice : Integer;  
pCanFrame: Pointer;  
lCANChannel : LongInt = 0): LongInt;  
StdCall; External 'PCICAN113' Name 'ReceiveFrame ';

**LabView:**

请参考相关演示程序。

**功能:** 接收帧操作。

**参数:**

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

pCanFrame CAN 帧结构体。

lCANChannel CAN 通道号。

**返回值:** 若成功，返回 TRUE，否则返回 FALSE。

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

◆ 读取错误信息

函数原型:

**Visual C++ & C++Builder:**

BOOL GetErrInfo(HANDLE hDevice,  
PBYTE pbyErrCode,  
LONG lCANChannel = 0)

**Visual Basic:**

Declare Function GetErrInfo Lib "PCICAN113" (ByVal hDevice As Long, \_  
ByRef pbyErrCode As Byte, \_  
Optional ByVal lCANChannel As Long = 0 ) As Boolean

**Delphi:**

Function GetErrInfo (hDevice : Integer;  
pbyErrCode: Pointer;  
lCANChannel : LongInt = 0): Boolean;  
StdCall; External 'PCICAN113' Name 'GetErrInfo ';

**LabView:**

请参考相关演示程序。

**功能:** 读取错误信息。

**参数:**

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

pbyErrCode 错误码。

lCANChannel CAN 通道号。

**返回值:** 若成功，返回 TRUE，否则返回 FALSE。

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

◆ ID 接受码寄存器设置

函数原型:

**Visual C++ & C++Builder:**

BOOL SetACRID (HANDLE hDevice, DWORD dwACR)

**Visual Basic:**

Declare Function SetACRIDLib "PCICAN113" (ByVal hDevice As Long, \_  
ByRef dwACR As Long) As Boolean

**Delphi:**

Function SetACRID (hDevice : Integer;  
dwACR: LongWord): Boolean;  
StdCall; External 'PCICAN113' Name 'SetACRID ';

**LabView:**

请参考相关演示程序。

**功能：** ID 接受码寄存器设置。

**参数：**

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

dwACR 接受码。

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

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

**◆ ID 屏蔽码寄存器设置**

函数原型：

**Visual C++ & C++Builder:**

BOOL SetAMRID (HANDLE hDevice, DWORD dwAMR)

**Visual Basic:**

Declare Function SetAMRID Lib "PCICAN113" (ByVal hDevice As Long, \_  
ByRef dwAMR As Long) As Boolean

**Delphi:**

Function SetAMRID (hDevice : Integer;  
dwAMR: LongWord): Boolean;  
StdCall; External 'PCICAN113' Name 'SetAMRID ';

**LabView:**

请参考相关演示程序。

**功能：** ID 屏蔽码寄存器设置。

**参数：**

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

dwAMR 屏蔽码。

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

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

**第三节、寄存器操作函数原型说明****◆读 SJA1000 寄存器内容**

函数原型：

**Visual C++ & C++Builder:**

WORD ReadReg (HANDLE hDevice, WORD waddr)

**Visual Basic:**



Declare Function ReadReg Lib "PCICAN113" (ByVal hDevice As Long, \_  
ByVal waddr As Integer) As Integer

**Delphi:**

Function ReadReg (hDevice : Integer;  
waddr: Word): Boolean;  
StdCall; External 'PCICAN113' Name 'ReadReg';

**LabView:**

请参考相关演示程序。

**功能:** 读 SJA1000 寄存器内容。

**参数:**

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

waddr 寄存器地址。

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

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

◆写 SJA1000 寄存器内容

函数原型:

**Visual C++ & C++Builder:**

WORD WriteReg (HANDLE hDevice, WORD waddr, WORD wVal)

**Visual Basic:**

Declare Function WriteRegLib "PCICAN113" (ByVal hDevice As Long, \_  
ByVal waddr As Integer, \_  
ByVal wVal As Integer) As Integer

**Delphi:**

Function WriteReg (hDevice : Integer;  
waddr: Word;  
wVal : Word): Boolean;  
StdCall; External 'PCICAN113' Name 'WriteReg';

**LabView:**

请参考相关演示程序。

**功能:** 写 SJA1000 寄存器内容。

**参数:**

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

waddr 寄存器地址。

wVal 寄存器内容。

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

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

## 第三章 硬件参数结构

### 第一节、CAN 设备初始化结构介绍 (CAN\_INIT\_PARA\_)

#### Visual C++ & C++Builder:

```
typedef struct _CAN_INIT_PARA_
{
    UINT    WorkMode;        // 工作方式，=0 时正常模式，=1 时为只听模式。
    UINT    SFTType;         // 帧类型发送帧类型，=0 时为正常发送，=1 时为单次发送，
                            // =2 时为自发自收，=3 时为单次自发自收，只在此帧为发送帧时有意义。
    UINT    BaudRate;        // 总线速率
} CAN_INIT_PARA, *PCAN_INIT_PARA;
```

#### Visual Basic:

```
Private Type CAN_INIT_PARA_
    WorkMode As Long        ' 工作方式，=0 时正常模式，=1 时为只听模式
    SFTType As Long         ' 帧类型发送帧类型，=0 时为正常发送，=1 时为单次发送，
                            ' =2 时为自发自收，=3 时为单次自发自收，只在此帧为发送帧时有意义。
    BaudRate As Long        ' 总线速率
End Type
```

#### Delphi:

```
Type // 定义结构体数据类型
PCAN_INIT_PARA_ = ^CAN_INIT_PARA_; // 指针类型结构
CAN_INIT_PARA_ = record            // 标记为记录型
    WorkMode: LongWord;             // 工作方式，=0 时正常模式，=1 时为只听模式
    SFTType: LongWord;              // 帧类型发送帧类型，=0 时为正常发送，=1 时为单次发送，
                                    // =2 时为自发自收，=3 时为单次自发自收，只在此帧为发送帧时有意义。
    BaudRate: LongWord;             // 总线速率
End;
```

#### LabView:

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

[WorkMode](#) 工作方式选择。它的其选项值如下表：

| 常量名                   | 常量值    | 功能定义 |
|-----------------------|--------|------|
| PCICAN113_MODE_NORMAL | 0x0000 | 正常模式 |
| PCICAN113_MODE_LISTEN | 0x0001 | 只听模式 |

[SFTType](#) 发送帧类型选择。它的其选项值如下表：

| 常量名                        | 常量值    | 功能定义   |
|----------------------------|--------|--------|
| PCICAN113_SFTYPE_NORMAL    | 0x0000 | 正常发送   |
| PCICAN113_SFTYPE_SINGLES   | 0x0001 | 单次发送   |
| PCICAN113_SFTYPE_SRSELF    | 0x0002 | 自发自收   |
| PCICAN113_SFTYPE_SIN_RSELF | 0x0003 | 单次自发自收 |

**BaudRate** 波特率选项选择。它的其选项值如下表：

| 常量名                   | 常量值    | 功能定义   |
|-----------------------|--------|--------|
| PCICAN113_BAUD_10KHZ  | 0x0000 | 10KHZ  |
| PCICAN113_BAUD_20KHZ  | 0x0001 | 20KHZ  |
| PCICAN113_BAUD_50KHZ  | 0x0002 | 50KHZ  |
| PCICAN113_BAUD_100KHZ | 0x0003 | 100KHZ |
| PCICAN113_BAUD_125KHZ | 0x0004 | 125KHZ |
| PCICAN113_BAUD_250KHZ | 0x0005 | 250KHZ |
| PCICAN113_BAUD_500KHZ | 0x0006 | 500KHZ |
| PCICAN113_BAUD_800KHZ | 0x0007 | 800KHZ |
| PCICAN113_BAUD_1MHZ   | 0x0008 | 1MHZ   |

## 第二节、CAN 帧结构 (CAN\_FRAME\_)

### Visual C++ & C++Builder:

```
typedef struct _CAN_FRAME_
```

```
{
    BYTE    FrameFormat;    // 帧格式    =0 为数据帧, =1 为远程帧
    BYTE    FrameType;      // 帧类型    =0 位标准帧, =1 为扩展帧
    BYTE    DataLen;        // 数据长度 <=8
    BYTE    Data[16];       // 数据
    DWORD   dwCanID;        // 设备 ID
    DWORD   dwSigntime;     // 接收帧时间标识
} CAN_FRAME, *PCAN_FRAME;
```

### Visual Basic:

```
Type CAN_FRAME_
```

```
    FrameFormat As BYTE    ' 帧格式 =0 为数据帧, =1 为远程帧
    FrameType As BYTE      ' 帧类型 =0 位标准帧, =1 为扩展帧
    DataLen As BYTE        ' 数据长度 <=8
    Data[16] As BYTE       ' 数据
    dwCanID As LongWord    ' 设备 ID
    dwSigntime As LongWord ' 接收帧时间标识
```

```
End Type
```

### Delphi:

```
Type // 定义结构体数据类型
```

```
    PCAN_FRAME_ = ^CAN_FRAME_; // 指针类型结构
    CAN_FRAME_ = record         // 标记为记录型
        FrameFormat : BYTE;     // 帧格式    =0 为数据帧, =1 为远程帧
        FrameType : BYTE;       // 帧类型    =0 位标准帧, =1 为扩展帧
        DataLen : BYTE;         // 数据长度 <=8
        Data : Array[0..15] of BYTE; // 数据
        dwCanID : LongWord;     // 设备 ID
        dwSigntime : LongWord;  // 接收帧时间标识
```

```
End;
```

**LabView:**

请参考其演示程序。

各参数说明:

**FrameFormat** 帧格式选择，它的取值如下表:

| 常量名                      | 常量值  | 功能定义 |
|--------------------------|------|------|
| PCICAN113_FFORMAT_DATA   | 0x00 | 数据帧  |
| PCICAN113_FFORMAT_REMOTE | 0x01 | 远程帧  |

**FrameType** 帧类型选择，它的取值如下表:

| 常量名                      | 常量值  | 功能定义 |
|--------------------------|------|------|
| PCICAN113_FTYPE_STANDARD | 0x00 | 标准帧  |
| PCICAN113_FTYPE_EXTEND   | 0x01 | 扩展帧  |

**DataLen** 数据长度，取值小于等于 8。

**Data[16]** 数据个数由 **DataLen** 决定。

**dwCanID** 设备 ID 号。

**dwSigntime** 接收帧时间标识。