PiUsb API

Version 2.0

December 5, 2020

Chapter 1

Overview

- **Version**: 2.0
- **Date**: December 5, 2020

The PiUsb library is a library for interfacing to Picard Industries UBS devices. It provides a software interface that allows you to write your own programs for use with any Picard Industrides USB product.

The PiUsb library can be used from any programming language or application that can call functions in a DLL (Dynamic Link Library). This includes Visual C++, Visual C#, and most other development environments.

The PiUsb library is supported in Windows 10. It should run under Windows XP and later operating systems. It supports development of 32-bit and 64-bit applications.

If you are developing in a .Net environment, we recommend that you use the **PiUsbNet API**. PiUsbNet is a .Net assembly (DLL), that serves as a wrapper for PiUsb. It is more powerful and easier to use from .net languauges such as C# and VB. The PiUsbNet API is described in a separate manual which is available in the SDK (software developement kit). Contact Picard Industries for additional information.

Chapter 2

Contact us

2.1 Technical Support

For help with hardware or software, or to report software bugs or errors in this documentation, email us at:

- **Support email**: info@picardindustries.com

2.2 Sales Information

Picard Industries, established in 1993, is committed to professional, affordable, and personalized service.

We manufacture a line of innovative miniature smart motor and sensor solutions to seemingly complex problems. We use integrated micro-controllers embedded into the motors and sensors themselves. Our customized software allows these motors and sensors to perform tasks with lower cost, greater ease, and in less space than can be done with other technology.

We also manufacture and sell stages, hexapods, and other motorized devices in a variety of sizes. Many of our products are customized for specific applications. We can design and manufacture motorized devices for your required various range of motion, resolution, and load capacity.

We also offer custom product development services:

- **Electrical**: Latest micro-controller based designs, schematic capture, and PCB layout.
- **Mechanical**: SolidWorks CAD designing and modeling.
- **Software**: Windows C++ and C# development, as well as embedded C development.

Please contact us for more information about any of the products or services we offer.

- **Web site**: http://www.picardindustries.com/

- **Sales email**: info@picardindustries.com
- **Telephone**: +1 585 589 0358

Chapter 3

Using the Library

3.1 Header and Lib Files

Header files

To use the PiUsb API, you must use the correct function "signature". In most programming languages you define function prototypes that describe the function name, parameters, and return value for the compiler.

Header files in C or C++

If you are using C or C++, these function prototypes already exist for you. Copy the file:

```
PiUsb.h
```

to your project folder, and include it in your project.

Add this line to your source code file that uses the PiUsb API.

```
#include "PiUsb.h"
```

Header files in other languages

If you are using a language other than C or C++, you will usually need to produce equivalent "header files" for your language. How this is done depends on the language and is beyond the scope of this document. Some languages can use or import the `PiUsb.h` file directly. Other languages have converters that can convert the C header files to appropriate constructs in the new language. Or you may need to make the conversion yourself and write a prototype in the language.

If you are using C#, VB, or another .net language, we recommend that you use the **PiUsbNet** API. **PiUsbNet** is a .Net assembly (DLL), that serves as a wrapper for PiUsb.dll. It is more powerful and easier to use from .net languauges such as C# and VB. The PiUsbNet API is described in a separate manual which is available in the SDK (software developement kit). Contact Picard Industries for additional information.

Lib (linker) files

In C and C++ you must use the "lib" file to allow the linker to resolve references to the PiUsb functions.

Copy the file:

```
PiUsb.lib
```

to your project folder, and include it in your project.

Then add the lib file to your library search path. In Visual Studio, go to your project properties, select the Linker / Input page, and add `PiUsb.lib` to the Additional Dependencies. You may also need to add the path to `PiUsb.lib` to Additional Library Directories on the Linker / General page.

Lib files are not normally used in languages other than C and C++.

Naming and linkage conventions

The PiUsb library uses "standard" windows DLL conventions. It uses `extern "C"` function (non-mangled) function names and `__stdcall` calling conventions.

3.2 DLL Files

The DLL file must be available at runtime.

During development, copy the file:

```
PiUsb.dll
```

to your target folder (the folder where your executable .exe file is created).

`PiUsb.dll` is available in 32-bit and 64-bit versions. You must copy the appropriate version, depending on whether you are developing a 32-bit or 64-bit application.

When you are ready to deploy your program to other users, you must also make sure to install `PiUsb.dll` along with your executable and any other files needed. See the Deployment and Redistributing section for additional details.

Driver

All Picard Industries USB products are HID (Human Interface Device) devices. These devices are handled automatically by Windows; there is no need for a separate driver.

3.3 Deployment and Redistributing

When you have written and debugged your application and are ready to release it to others, you need to include PiUsb.dll as part of your deployment.

You must install the appropriate version of `PiUsb.dll` - either 32-bit or 64-bit. If your application is built specifically as a 32-bit or 64-bit version, then install the matching version of `PiUsb.dll`.

If you are developing a .Net application built for "AnyCPU", you must use the version of `PiUsb.dll` that matches the Windows version on the target system. Most recent computers are running the 64-bit version of Windows, but your installer should check which version of windows is being used and copy the appropriate dll.

Chapter 4

Programming Concepts

4.1 Device Handles

PiUsb supports communicating with multiple devices. When you connect to a device, the `piConnect_____` function returns a `DeviceHandle`. The `DeviceHandle` is pointer value (handle) used to identify the device. You must pass the `DeviceHandle` to all other PiUsb functions to indicate which device you are referring to.

When you are done using the device you should disconnect from it with the appropriate `piDisconnect_____` function. After calling the `piDisconnect_____` function the device handle is no longer valid and should not be used.

A device handle is a pointer (void$*$). It is 32 bits in size in 32-bit windows, and 64 bits in size for 64-bit windows. Do not store the handle in an Integer variable, since integers are 32 bits in size and this will truncate the 64-bit handle in 64-bit Windows.

4.2 Error Handling

Errors in PiUsb are indicated by an Error Number which is returned as a parameter value or the function return value. A non-zero return value indicates an error.

The error numbers are defined using #defined constants defined in the header file `PiUsb.h`. See the Error Numbers section for a description of each error.

4.3 Thread Safety

PiUsb is thread-safe. You can call PiUsb functions from any thread.

Of course, if you use the same device from multiple threads, the threads can interfere with each other. For example, if thread 1 calls piRunMotorToPosition() and shortly thereafter thread 2 also calls piRunMotorToPosition(), the motor will move to the destination of thread 2, overriding the command of thread 1.

4.4 Device Disconnect

It is possible for a device to become disconnected from the computer and your application. For example, the user might intentionally or unintenionally disconnect the USB cable that connects the device to the computer. Your program should be able to handle these situations cleanly.

When a device is disconnected, the next attempt to communicate with the device will fail. This generally results in a PI_READ_TIMEOUT or PI_WRITE_FAILED error return value from the function called.

Your program should be prepared to handle all error return values, but especially PI_DEVICE_NOT_FOUND, PI_READ_TIMEOUT, and PI_WRITE_FAILED. When an error is detected, you should call the appropriate `piDisconnect_____` function to disconnect the device in your program.

Once the user has restored the physical connection, you should re-open the device with a call to the appropriate `piConnect_____` function to connect to the device again. One technique for doing this is to provide a button or menu item in the user interface where the user can initiate the Connect operation after they physically reconnect the device.

Another technique is to look for a Windows notification when a device is plugged in to the computer, and attempt to reconnect automatically. The Sample Program for each device demonstrates how to do this.

4.5 Sample Programs

There is a demonstration (sample) program for each device available on the Picard Industries website:
`http://www.picardindustries.com/`

The source code for the sample program is included in the download zip file.

Building the Sample Application

The sample program is a C++ program that uses the Microsoft Foundation Classes (MFC) framework. In recent versions of Visual Studio, MFC is not installed by default. To compile the sample program you will need to enable the MFC framework installation option.

You can install the MFC Framework when you initially install Visual Studio, or at a later time.

- Run the Visual Studio Installer.
- Click on the Individual components tab.
- Scroll down to "SDKs, libraries, and frameworks"
- Check the option for "C++ MFC for latest Vxxx build tools (x86 & x64)".

Chapter 5

Module Index

5.1 PiUsb API Reference

Here is a list of all modules:

Chapter 6

Module Documentation

6.1 Handles

Typedefs and macros used to define device handles.

Macros

- #define INVALID_PIHANDLE ((PIHANDLE)0)
  Constant for an invalid device handle.

Typedefs

- typedef void * PIHANDLE
  Typedef for a device handle.

6.1.1 Detailed Description

Typedefs and macros used to define device handles.

6.1.2 Macro Definition Documentation

6.1.2.1 INVALID_PIHANDLE

```
#define INVALID_PIHANDLE ((PIHANDLE)0)
```

Constant for an invalid device handle.

6.1.3 Typedef Documentation

6.1.3.1 PIHANDLE

```
typedef void* PIHANDLE
```

Typedef for a device handle.

See also

Device Handles

6.2 Error Numbers

Success or error return values.

Macros

- #define PI_NO_ERROR 0

  No error.
- #define PI_DEVICE_NOT_FOUND 1

  Device not found.
- #define PI_OBJECT_NOT_FOUND 2

  Device handle does not exist.
- #define PI_CANNOT_CREATE_OBJECT 3

  Cannot create a device handle for the specified device.
- #define PI_INVALID_DEVICE_HANDLE 4

  Invalid device handle.
- #define PI_READ_TIMEOUT 5

  Timeout while attempting to read from the device.
- #define PI_READ_THREAD_ABANDONED 6

  The system abandoned the read operation.
- #define PI_READ_FAILED 7

  An attempt to read from the device failed.
- #define PI_INVALID_PARAMETER 8

  An invalid parameter value was passed to a function.
- #define PI_WRITE_FAILED 9

  An attempt to write to the device failed.

6.2.1 Detailed Description

Success or error return values.

Most functions in the PiUsb API return an integer Error Number which indicates whether the function succeeded of failed, and the cause of any failure.

See also

Error Handling

6.2.2 Macro Definition Documentation

6.2.2.1 PI_CANNOT_CREATE_OBJECT

```
#define PI_CANNOT_CREATE_OBJECT 3
```

Cannot create a device handle for the specified device.

The system was unable to allocate memory for the requested device.

See also

Error Handling

6.2.2.2 PI_DEVICE_NOT_FOUND

```
#define PI_DEVICE_NOT_FOUND 1
```

Device not found.

The specified device could not be found or has been disconnected.

See also

Error Handling

6.2.2.3 PI_INVALID_DEVICE_HANDLE

```
#define PI_INVALID_DEVICE_HANDLE 4
```

Invalid device handle.

The device handle specifies a device that cannot be used with this function.

See also

Error Handling

6.2.2.4 PI_INVALID_PARAMETER

```
#define PI_INVALID_PARAMETER 8
```

An invalid parameter value was passed to a function.

See also

Error Handling

6.2.2.5 PI_NO_ERROR

```
#define PI_NO_ERROR 0
```

No error.

The operation was successful.

See also

Error Handling

6.2.2.6 PI_OBJECT_NOT_FOUND

```
#define PI_OBJECT_NOT_FOUND 2
```

Device handle does not exist.

The device handle does not exist or is null.

See also

Error Handling

6.2.2.7 PI_READ_FAILED

```
#define PI_READ_FAILED 7
```

An attempt to read from the device failed.

The system reported a failure when attempting to read from the device. Disconnect from the device and attempt to reconnect to it.

See also

Error Handling

6.2.2.8 PI_READ_THREAD_ABANDONED

```
#define PI_READ_THREAD_ABANDONED 6
```

The system abandoned the read operation.

The system thread that is reading from the device has exited or been killed. Disconnect from the device and attempt to reconnect to it.

See also

Error Handling

6.2.2.9 PI_READ_TIMEOUT

```
#define PI_READ_TIMEOUT 5
```

Timeout while attempting to read from the device.

The system timed out while attempting to read data from the device. This usually means that the device has been disconnected.

See also

Error Handling

6.2.2.10 PI_WRITE_FAILED

```
#define PI_WRITE_FAILED 9
```

An attempt to write to the device failed.

The system reported a failure when attempting to write to the device. Disconnect from the device and attempt to reconnect to it.

See also

Error Handling

6.3 Filter Wheel Functions

Functions to control USB Filter Wheel devices You can also connect to and control Filter Wheel devices using the Gradient Wheel Functions and Rotator Functions.

Functions

- PIHANDLE __stdcall piConnectFilter (int *ErrorNumber, int SerialNum)

  Connect to a USB Filter Wheel.
- void __stdcall piDisconnectFilter (PIHANDLE handle)

  Disconnect from a USB Filter Wheel.
- int __stdcall piGetFilterPosition (int *Position, PIHANDLE handle)

  Get the filter wheel position.
- int __stdcall piSetFilterPosition (int Destination, PIHANDLE handle)

  Initiate a move to a new filter wheel position.

6.3.1 Detailed Description

Functions to control USB Filter Wheel devices You can also connect to and control Filter Wheel devices using the Gradient Wheel Functions and Rotator Functions.

6.3.2 Function Documentation

6.3.2.1 piConnectFilter()

```
PIHANDLE __stdcall piConnectFilter (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Filter Wheel.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectFilter()

6.3.2.2 piDisconnectFilter()

```
void __stdcall piDisconnectFilter (
            PIHANDLE handle )
```

Disconnect from a USB Filter Wheel.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectFilter()

6.3.2.3 piGetFilterPosition()

```
int __stdcall piGetFilterPosition (
            int * Position,
            PIHANDLE handle )
```

Get the filter wheel position.

Parameters

Position	[out] The current position of the filter wheel.
handle	[in] The device handle.

Returns

An error number.

For a standard filter wheel, the reported position will be between 1 and 6. Custom devices can have up to 16 positions.

The `Position` will be set to 0 if the filter wheel is between positions. While moving the position will be reported as 0 most of the time, but will report a positive value as it passes through each position.

See also

piSetFilterPosition()

6.3.2.4 piSetFilterPosition()

```
int __stdcall piSetFilterPosition (
            int Destination,
            PIHANDLE handle )
```

Initiate a move to a new filter wheel position.

Parameters

Destination	`[in]` The destination position of the filter wheel.
handle	`[in]` The device handle.

Returns

An error number.

For a standard filter wheel, the destination position should be between 1 and 6. Custom devices can have up to 16 positions.

Setting the position to 0 will cause the filter wheel to stop immediately.

See also

piGetFilterPosition()

6.4 Gradient Wheel Functions

Functions to control USB Gradient Wheel devices You can also connect to and control Gradient Wheel devices using the Rotator Functions.

Functions

- PIHANDLE __stdcall piConnectGWheel (int *ErrorNumber, int SerialNum)

  Connect to a USB Gradient Wheel.
- void __stdcall piDisconnectGWheel (PIHANDLE handle)

  Disconnect from a USB Gradient Wheel.
- int __stdcall piGetGWheelPosition (int *Position, PIHANDLE handle)

  Get the gradient wheel position.
- int __stdcall piSetGWheelPosition (int Destination, PIHANDLE handle)

  Initiate a move to a new gradient wheel position.

6.4.1 Detailed Description

Functions to control USB Gradient Wheel devices You can also connect to and control Gradient Wheel devices using the Rotator Functions.

6.4.2 Function Documentation

6.4.2.1 piConnectGWheel()

```
PIHANDLE __stdcall piConnectGWheel (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Gradient Wheel.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectGWheel()

6.4.2.2 piDisconnectGWheel()

```
void __stdcall piDisconnectGWheel (
            PIHANDLE handle )
```

Disconnect from a USB Gradient Wheel.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectGWheel()

6.4.2.3 piGetGWheelPosition()

```
int __stdcall piGetGWheelPosition (
            int * Position,
            PIHANDLE handle )
```

Get the gradient wheel position.

Parameters

Position	[out] The current position of the gradient wheel.
handle	[in] The device handle.

Returns

An error number.

The reported position will be between 1 and 1023.

See also

piSetGWheelPosition()

6.4.2.4 piSetGWheelPosition()

```
int __stdcall piSetGWheelPosition (
            int Destination,
            PIHANDLE handle )
```

Initiate a move to a new gradient wheel position.

Parameters

Destination	[in] The destination position of the gradient wheel.
handle	[in] The device handle.

Returns

An error number.

The destination position should be between 1 and 1023.

Setting the position to 0 will cause the gradient wheel to stop immediately.

See also

piGetGWheelPosition()

6.5 Flipper Functions

Functions to control USB Flipper devices.

Modules

- Flipper Constants

  State of the Flipper.

Functions

- PIHANDLE __stdcall piConnectFlipper (int *ErrorNumber, int SerialNum)

  Connect to a USB Flipper.
- void __stdcall piDisconnectFlipper (PIHANDLE handle)

  Disconnect from a USB Flipper.
- int __stdcall piGetFlipperState (int *FlipperState, PIHANDLE handle)

  Get the flipper state.
- int __stdcall piSetFlipperState (int FlipperState, PIHANDLE handle)

  Set the flipper state.

6.5.1 Detailed Description

Functions to control USB Flipper devices.

6.5.2 Function Documentation

6.5.2.1 piConnectFlipper()

```
PIHANDLE __stdcall piConnectFlipper (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Flipper.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectFlipper()

6.5.2.2 piDisconnectFlipper()

```
void __stdcall piDisconnectFlipper (
            PIHANDLE handle )
```

Disconnect from a USB Flipper.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectFlipper()

6.5.2.3 piGetFlipperState()

```
int __stdcall piGetFlipperState (
            int * FlipperState,
            PIHANDLE handle )
```

Get the flipper state.

Parameters

FlipperState	[out] The current state of the flipper.
handle	[in] The device handle.

Returns

An error number.

`FlipperState` will be returned as one of:

- PI_FLIPPER_RETRACTED
- PI_FLIPPER_EXTENDED

Immediately after powering up the flipper, the flipper will be retracted.

The flipper device contains a sensor which reports the actual state of the flipper. If the flipper is physically blocked from moving, or you manually move the flipper, piGetFlipperState will return the actual state and not state commanded with piSetFlipperState(). This is in contrast to a Shutter device which has no sensor and piGetShutterState will return the commanded state.

See also

piSetFlipperState()

6.5.2.4 piSetFlipperState()

```
int __stdcall piSetFlipperState (
            int FlipperState,
            PIHANDLE handle )
```

Set the flipper state.

Parameters

FlipperState	[in] The state of the flipper to set.
handle	[in] The device handle.

Returns

An error number.

The flipper will be set to the specified state. Valid values are:

- PI_FLIPPER_RETRACTED
- PI_FLIPPER_EXTENDED

The flipper device contains a sensor which reports the actual state of the flipper. If the flipper is physically blocked from moving, or you manually move the flipper, piGetFlipperState will return the actual state and not state commanded with

piSetFlipperState(). You may want to keep track of the most recently commanded state in order to toggle to the opposite state even if the flipper was physically blocked.

See also

piGetFlipperState()

6.6 Flipper Constants

State of the Flipper.

Macros

- #define PI_FLIPPER_RETRACTED 0

  Flipper is retracted (closed).
- #define PI_FLIPPER_EXTENDED 1

  Flipper is extended (open).

6.6.1 Detailed Description

State of the Flipper.

6.6.2 Macro Definition Documentation

6.6.2.1 PI_FLIPPER_EXTENDED

```
#define PI_FLIPPER_EXTENDED 1
```

Flipper is extended (open).

6.6.2.2 PI_FLIPPER_RETRACTED

```
#define PI_FLIPPER_RETRACTED 0
```

Flipper is retracted (closed).

6.7 Laser Functions

Functions to control USB Laser devices.

Modules

- Laser Constants

  State of the Laser.

Functions

- PIHANDLE __stdcall piConnectLaser (int *ErrorNumber, int SerialNum)

  Connect to a USB Laser.
- void __stdcall piDisconnectLaser (PIHANDLE handle)

  Disconnect from a USB Laser.
- int __stdcall piGetLaserState (int *LaserState, PIHANDLE handle)

  Get the laser state.
- int __stdcall piSetLaserState (int LaserState, PIHANDLE handle)

  Set the laser state.

6.7.1 Detailed Description

Functions to control USB Laser devices.

6.7.2 Function Documentation

6.7.2.1 piConnectLaser()

```
PIHANDLE __stdcall piConnectLaser (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Laser.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectLaser()

6.7.2.2 piDisconnectLaser()

```
void __stdcall piDisconnectLaser (
            PIHANDLE handle )
```

Disconnect from a USB Laser.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectLaser()

6.7.2.3 piGetLaserState()

```
int __stdcall piGetLaserState (
            int * LaserState,
            PIHANDLE handle )
```

Get the laser state.

Parameters

LaserState	[out] The current state of the laser.
handle	[in] The device handle.

Returns

An error number.

`LaserState` will be returned as one of:

- PI_LASER_OFF
- PI_LASER_ON

Immediately after powering up the laser, the laser will be off.

See also

piSetLaserState()

6.7.2.4 piSetLaserState()

```
int __stdcall piSetLaserState (
              int LaserState,
              PIHANDLE handle )
```

Set the laser state.

Parameters

LaserState	`[in]` The state of the laser to set.
handle	`[in]` The device handle.

Returns

An error number.

The laser will be set to the specified state. Valid values are:

- PI_LASER_OFF
- PI_LASER_ON

See also

piGetLaserState()

6.8 Laser Constants

State of the Laser.

Macros

- #define PI_LASER_OFF 0

  Laser is off.
- #define PI_LASER_ON 1

  Laser is on.

6.8.1 Detailed Description

State of the Laser.

6.8.2 Macro Definition Documentation

6.8.2.1 PI_LASER_OFF

```
#define PI_LASER_OFF 0
```

Laser is off.

6.8.2.2 PI_LASER_ON

```
#define PI_LASER_ON 1
```

Laser is on.

6.9 Motor Functions

Functions to control USB Motor devices.

Functions

- PIHANDLE __stdcall piConnectMotor (int *ErrorNumber, int SerialNum)

  Connect to a USB Motor.
- void __stdcall piDisconnectMotor (PIHANDLE handle)

  Disconnect from a USB Motor.
- int __stdcall piGetMotorHomeStatus (BOOL *AtHome, PIHANDLE handle)

  Get the state of the home switch.
- int __stdcall piGetMotorMovingStatus (BOOL *Moving, PIHANDLE handle)

  Get a value indicating whether the motor is moving.
- int __stdcall piGetMotorPosition (int *Position, PIHANDLE handle)

  Get the position of the motor.
- int __stdcall piGetMotorStatus (int *Position, BOOL *Moving, BOOL *AtHome, PIHANDLE handle)

  Get the position and status of the motor.
- int __stdcall piGetMotorVelocity (int *Velocity, PIHANDLE handle)

  Get the velocity of the motor.
- int __stdcall piHaltMotor (PIHANDLE handle)

  Stop the motor
- int __stdcall piHomeMotor (int Velocity, PIHANDLE handle)

  Initiate homing the motor.
- int __stdcall piRunMotorToPosition (int Destination, int Velocity, PIHANDLE handle)

  Initiate a move to a destination position.
- int __stdcall piSetMotorVelocity (int Velocity, PIHANDLE handle)

  Set the velocity of the motor.

6.9.1 Detailed Description

Functions to control USB Motor devices.

USB Motor Position

The USB Motor uses a stepper motor to rotate a lead screw. You command the motor to move to a new position with the piRunMotorToPosition() function, specifying the destination position in steps (or counts). There are 200 steps for each revolution of the lead screw. How far the device attached to the lead screw advances with each step depends on the pitch of the lead screw. You can specify any position between 1 and 2 31 -1, but the physical upper limit depends on the device. Standard devices and their limits are:

Device	Limit (steps)
USB Motor 1	1,900

Device	Limit (steps)
USB Motor 2	5,600
USB Pusher	50,000
USB LabJack	200,000

You can read the current position at any time (while moving or not) with the :: piGetMotorPosition() and piGetMotorStatus() functions. The position will be set to zero when you home the device with piHomeMotor() function.

USB Motor Velocity

The :piRunMotorToPosition() and piHomeMotor() functions require you to specify the velocity of the motion. The velocity is a number between 1 and 12, where 1 is the slowest speed and 12 is the highest speed.

We recommend that you limit your velocity to be 10 or less. These velocities should work well with most devices you connect to the Motor. If your load is small and light, you may be able to use faster velocities (11 and 12). If you attempt to move too large a load at too high a speed, the motor may stall, or may miss steps and not move the full distance.

The following table shows the velocity settings and the approximate speed they correspond to:

Velocity	Steps/sec
1	133
2	143
3	154
4	167
5	182
6	200
7	222
8	250
9	286
10	333
11	400
12	500

6.9.2 Function Documentation

6.9.2.1 piConnectMotor()

```
PIHANDLE __stdcall piConnectMotor (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Motor.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectMotor()

6.9.2.2 piDisconnectMotor()

```
void __stdcall piDisconnectMotor (
            PIHANDLE handle )
```

Disconnect from a USB Motor.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectMotor()

6.9.2.3 piGetMotorHomeStatus()

```
int __stdcall piGetMotorHomeStatus (
            BOOL * AtHome,
            PIHANDLE handle )
```

Get the state of the home switch.

Parameters

AtHome	[out] The state of the home switch.
handle	[in] The device handle.

Returns

An error number.

AtHome will be TRUE when at the home position, and FALSE otherwise.

See also

piHomeMotor()

6.9.2.4 piGetMotorMovingStatus()

```
int __stdcall piGetMotorMovingStatus (
            BOOL * Moving,
            PIHANDLE handle )
```

Get a value indicating whether the motor is moving.

Parameters

Moving	[out] TRUE if the motor is moving, FALSE otherwise.
handle	[in] The device handle.

Returns

An error number.

See also

piRunMotorToPosition()

6.9.2.5 piGetMotorPosition()

```
int __stdcall piGetMotorPosition (
            int * Position,
```

```
            PIHANDLE handle )
```

Get the position of the motor.

Parameters

Position	[out] The current position of the motor.
handle	[in] The device handle.

Returns

An error number.

The Position is a value between 1 and 2^{31}-1, but the physical upper limit depends on the device. See Motor Position.

See also

Motor Position

piRunMotorToPosition()

6.9.2.6 piGetMotorStatus()

```
int __stdcall piGetMotorStatus (
            int * Position,
            BOOL * Moving,
            BOOL * AtHome,
            PIHANDLE handle )
```

Get the position and status of the motor.

Parameters

Position	[out] The current position of the motor.
AtHome	[out] The state of the home switch.
Moving	[out] TRUE if the motor is moving, FALSE otherwise.
handle	[in] The device handle.

Returns

An error number.

This method gets the current motor position and status in a single function call. It returns the value of the piGetMotorPosition(), piGetMotorMovingStatus(), and piGetMotorHomeStatus() functions.

It is more efficient to call this function rather than calling 3 separate functions. Using this method reduces I/O traffic to the device and can improve the responsiveness of your application.

The `Position` is a value between 1 and 2^{31}-1, but the physical upper limit depends on the device. See Motor Position.

See also

Motor Position

piRunMotorToPosition()

6.9.2.7 piGetMotorVelocity()

```
int __stdcall piGetMotorVelocity (
              int * Velocity,
              PIHANDLE handle )
```

Get the velocity of the motor.

Parameters

Velocity	`[out]` The velocity of the motor.
handle	`[in]` The device handle.

Returns

An error number.

See also

Motor Velocity

piRunMotorToPosition()

piHomeMotor()

6.9.2.8 piHaltMotor()

```
int __stdcall piHaltMotor (
              PIHANDLE handle )
```

Stop the motor

Parameters

handle	[in] The device handle.

Returns

An error number.

Calling piHaltMotor() will stop the motor from moving and will abort any ongoing homing operation.

See also

piRunMotorToPosition()

piHomeMotor()

6.9.2.9 piHomeMotor()

```
int __stdcall piHomeMotor (
            int Velocity,
            PIHANDLE handle )
```

Initiate homing the motor.

Parameters

Velocity	[in] The motor velocity to use during homing (1 to 12).
handle	[in] The device handle.

Returns

An error number.

The system will initiate a search for the home switch. This function does not wait for homing to complete.

The motor will move in the negative direction and look for the home switch to turn on. If the home switch is missing or fails, the home operation will continue until you call piHaltMotor().

The home position is not established until you initiate a move to a positive (non-zero) position after finding the home switch location. After finding the home switch the motor sets the position zero. When you subsequently initiate a move to a positive (non-zero) position, the motor will keep the reported position at zero while it moves until the home switch turns off. It then set that position as the zero position.

See also

Motor Velocity

piRunMotorToPosition()

6.9.2.10 piRunMotorToPosition()

```
int __stdcall piRunMotorToPosition (
              int Destination,
              int Velocity,
              PIHANDLE handle )
```

Initiate a move to a destination position.

Parameters

Destination	[in] The destination motor position.
Velocity	[in] The motor velocity to use during the move (1 to 12).
handle	[in] The device handle.

Returns

An error number.

The system will initiate a move to the specified `Destination` position at the specified `Velocity`. This function does not wait for the move to complete.

The destination position can be any position between 1 and 2^{31}-1, but the physical upper limit depends on the device. See Motor Position.

See also

Motor Position

Motor Velocity

6.9.2.11 piSetMotorVelocity()

```
int __stdcall piSetMotorVelocity (
              int Velocity,
              PIHANDLE handle )
```

Set the velocity of the motor.

Parameters

Velocity	`[in]` The velocity of the motor.
handle	`[in]` The device handle.

Returns

An error number.

Deprecated

This function is deprecated and may be removed from future versions of the library.

In theory, this function changes the velocity of the motor. However, the motor device does not permit changing velocity on the fly. You must first call piHaltMotor() to stop the motor, and then issue a new piRunMotorToPosition() and where you specify the new destination and velocity. Setting the velocity with this function is effectively meaningless.

See also

Motor Velocity

piRunMotorToPosition()

piHomeMotor()

6.10 Shutter Functions

Functions to control USB Shutter devices.

Modules

- Shutter Constants

  State of the Shutter.

Functions

- PIHANDLE __stdcall piConnectShutter (int *ErrorNumber, int SerialNum)

  Connect to a USB Shutter.
- void __stdcall piDisconnectShutter (PIHANDLE handle)

  Disconnect from a USB Shutter.
- int __stdcall piGetShutterState (int *ShutterState, PIHANDLE handle)

  Get the shutter state.
- int __stdcall piSetShutterState (int ShutterState, PIHANDLE handle)

  Set the shutter state.

6.10.1 Detailed Description

Functions to control USB Shutter devices.

6.10.2 Function Documentation

6.10.2.1 piConnectShutter()

```
PIHANDLE __stdcall piConnectShutter (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Shutter.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectShutter()

6.10.2.2 piDisconnectShutter()

```
void __stdcall piDisconnectShutter (
            PIHANDLE handle )
```

Disconnect from a USB Shutter.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectShutter()

6.10.2.3 piGetShutterState()

```
int __stdcall piGetShutterState (
            int * ShutterState,
            PIHANDLE handle )
```

Get the shutter state.

Parameters

ShutterState	[out] The current state of the shutter.
handle	[in] The device handle.

Returns

An error number.

The shutter state will be returned as one of:

- PI_SHUTTER_OPEN
- PI_SHUTTER_CLOSED

Immediately after powering up the shutter, the shutter will be closed.

The shutter device does not contain a sensor and the shutter state reflects the most recent state commanded with piSetShutterState(). If the shutter is physically blocked from moving, or you manually move the shutter, piGetShutterState will return the commanded state and not the actual state.

See also

piSetShutterState()

6.10.2.4 piSetShutterState()

```
int __stdcall piSetShutterState (
            int ShutterState,
            PIHANDLE handle )
```

Set the shutter state.

Parameters

ShutterState	[in] The state of the shutter to set.
handle	[in] The device handle.

Returns

An error number.

The shutter will be set to the specified state. Valid values are:

- PI_SHUTTER_CLOSED
- PI_SHUTTER_OPEN

See also

piGetShutterState()

6.11 Shutter Constants

State of the Shutter.

Macros

- #define PI_SHUTTER_CLOSED 0

  Shutter is closed.
- #define PI_SHUTTER_OPEN 1

  Shutter is open.

6.11.1 Detailed Description

State of the Shutter.

6.11.2 Macro Definition Documentation

6.11.2.1 PI_SHUTTER_CLOSED

```
#define PI_SHUTTER_CLOSED 0
```

Shutter is closed.

6.11.2.2 PI_SHUTTER_OPEN

```
#define PI_SHUTTER_OPEN 1
```

Shutter is open.

6.12 Relay Functions

Functions to control USB Relay devices.

Functions

- PIHANDLE __stdcall piConnectRelay (int *ErrorNumber, int SerialNum)

  Connect to a USB Relay.
- void __stdcall piDisconnectRelay (PIHANDLE handle)

  Disconnect from a USB Relay.
- int __stdcall piGetRelayStates (int *RelayStates, PIHANDLE handle)

  Get the state of the relays.
- int __stdcall piSetRelayStates (int RelayStates, PIHANDLE handle)

  Set the state of the relays.

6.12.1 Detailed Description

Functions to control USB Relay devices.

6.12.2 Function Documentation

6.12.2.1 piConnectRelay()

```
PIHANDLE __stdcall piConnectRelay (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Relay.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectRelay()

6.12.2.2 piDisconnectRelay()

```
void __stdcall piDisconnectRelay (
            PIHANDLE handle )
```

Disconnect from a USB Relay.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectRelay()

6.12.2.3 piGetRelayStates()

```
int __stdcall piGetRelayStates (
            int * RelayStates,
            PIHANDLE handle )
```

Get the state of the relays.

Parameters

RelayStates	[out] The current state of the relays.
handle	[in] The device handle.

Returns

An error number.

RelayStates will be returned with a bit for each relay. Bit 0, the low order bit, corresponds to relay 1. Bit 1

corresponds to relay 2, etc. The bit will be 1 when the relay is ON (energized), and will be 0 when the relay is OFF (de-energized).

Immediately after powering up the Relay device, all relays are OFF (de-energized).

See also

piSetRelayStates()

6.12.2.4 piSetRelayStates()

```
int __stdcall piSetRelayStates (
            int RelayStates,
            PIHANDLE handle )
```

Set the state of the relays.

Parameters

RelayStates	[in] The desired state of the relays.
handle	[in] The device handle.

Returns

An error number.

`RelayStates` contains a bit for each relay. Bit 0, the low order bit, corresponds to relay 1. Bit 1 corresponds to relay 2, etc. The bit should be 1 when the relay is set ON (energized), and should be 0 when the relay is set OFF (de-energized).

Immediately after powering up the Relay device, all relays are OFF (de-energized).

See also

piGetRelayStates()

6.13 Rotator Functions

Functions to control USB Rotator devices You can also connect to and control Rotator devices using the Gradient Wheel Functions.

Functions

- PIHANDLE __stdcall piConnectRotator (int *ErrorNumber, int SerialNum)

  Connect to a USB Rotator.
- void __stdcall piDisconnectRotator (PIHANDLE handle)

  Disconnect from a USB Rotator.
- int __stdcall piGetRotatorPosition (int *Position, PIHANDLE handle)

  Get the rotator position.
- int __stdcall piSetRotatorPosition (int Destination, PIHANDLE handle)

  Initiate a move to a new rotator position.

6.13.1 Detailed Description

Functions to control USB Rotator devices You can also connect to and control Rotator devices using the Gradient Wheel Functions.

6.13.2 Function Documentation

6.13.2.1 piConnectRotator()

```
PIHANDLE __stdcall piConnectRotator (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Rotator.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectRotator()

6.13.2.2 piDisconnectRotator()

```
void __stdcall piDisconnectRotator (
            PIHANDLE handle )
```

Disconnect from a USB Rotator.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectRotator()

6.13.2.3 piGetRotatorPosition()

```
int __stdcall piGetRotatorPosition (
            int * Position,
            PIHANDLE handle )
```

Get the rotator position.

Parameters

Position	[out] The current position of the rotator.
handle	[in] The device handle.

Returns

An error number.

The reported position will be between 1 and 1023.

See also

piSetRotatorPosition()

6.13.2.4 piSetRotatorPosition()

```
int __stdcall piSetRotatorPosition (
            int Destination,
            PIHANDLE handle )
```

Initiate a move to a new rotator position.

Parameters

Destination	[in] The destination position of the rotator.
handle	[in] The device handle.

Returns

An error number.

The destination position should be between 1 and 1023.

Setting the position to 0 will cause the rotator to stop immediately.

See also

piGetRotatorPosition()

6.14 Twister Functions

Functions to control USB Twister devices.

Functions

- PIHANDLE __stdcall piConnectTwister (int *ErrorNumber, int SerialNum)

  Connect to a USB Twister.
- void __stdcall piDisconnectTwister (PIHANDLE handle)

  Disconnect from a USB Twister.
- int __stdcall piGetTwisterMovingStatus (BOOL *Moving, PIHANDLE handle)

  Get a value indicating whether the twister is moving.
- int __stdcall piGetTwisterPosition (int *Position, PIHANDLE handle)

  Get the position of the twister.
- int __stdcall piGetTwisterSensorPosition (int *SensorPosition, PIHANDLE handle)

  Get the position of the twister sensor.
- int __stdcall piGetTwisterStatus (int *Position, BOOL *Moving, PIHANDLE handle)

  Get the position and status of the twister.
- int __stdcall piGetTwisterStatusEx (int *Position, int *SensorPosition, BOOL *Moving, PIHANDLE handle)

  Get the position and status of the twister.
- int __stdcall piGetTwisterVelocity (int *Velocity, PIHANDLE handle)

  Get the velocity of the twister.
- int __stdcall piHaltTwister (PIHANDLE handle)

  Stop the twister
- int __stdcall piRunTwisterContinuous (int Direction, int Velocity, PIHANDLE handle)

  Start the Twister moving continuously.
- int __stdcall piRunTwisterToPosition (int Destination, int Velocity, PIHANDLE handle)

  Initiate a move to a destination position.
- int __stdcall piSetTwisterPositionZero (PIHANDLE handle)

  Set the twister position to zero.

6.14.1 Detailed Description

Functions to control USB Twister devices.

USB Twister Position

The USB Twister uses a stepper motor to rotate a shaft. You command the motor to move to a new position with the piRunTwisterToPosition() function, specifying the destination position in steps (or counts). There are 200 steps for each revolution of the shaft, so each step corresponds to 1.8 degrees. You can specify any position between -32767 and +32767. If you specify a position less than -32767, the motor will move to position -32767. Similarly, if you specify a position greater than +32767, the motor will move to position +32767.

You can read the current position at any time (while moving or not) with the piGetTwisterPosition(), piGetTwisterStatus(), and piGetTwisterStatusEx() functions. You can reset the position to zero with the piSetTwisterPositionZero() function.

You can command the motor to move continuously in either the positive or negative direction with the piRunTwisterContinuous() function. When you issue this command, the position is set to zero and remains there during the continuous move. You can leave the motor running for as long as you wish.

The positive direction (increasing counts) is counter-clockwise rotation if you are looking at the USB Twister from the shaft end.

USB Twister Velocity

The piRunTwisterToPosition() and piRunTwisterContinuous() functions let you specify the move velocity. The velocity is a number between 1 and 13, where 1 is the slowest speed and 13 is the highest speed.

We recommend that you limit your velocity to be 10 or less. These velocities should work well with most devices you connect to the Twister. If your load is small and light, you may be able to use some or all of the faster velocities between 11 and 13. If you attempt to move too large a load at too high a speed, the motor may stall, or may miss steps and not move the full distance.

The following table shows the velocity settings and the approximate speed they correspond to.

Velocity	Steps/sec	Degrees/Sec	RPM
1	133	240	40
2	143	257	43
3	154	277	46
4	167	300	50
5	182	328	55
6	200	360	60
7	222	400	67
8	250	450	75
9	286	514	86
10	333	600	100
11	400	720	120
12	500	900	150
13	667	1200	200

USB Twister Sensor Position

Some versions of the Twister include an analog sensor which can be used to read a position. This is an option that can be ordered with the device. It is automatically included on versions of the twister used in the USB ZTable product.

The position is a 10 bit value (0 to 1023) returned from the A/D converter on the board. It reads the value of a potentiometer which is attached to the rotary stage. The sensor position does not have as much resolution as the motor steps, but it is absolute rather than relative to the power-on or user set zero position.

The piGetTwisterSensorPosition() and piGetTwisterStatusEx() functions return the value of this sensor.

6.14.2 Function Documentation

6.14.2.1 piConnectTwister()

```
PIHANDLE __stdcall piConnectTwister (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Twister.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectTwister()

6.14.2.2 piDisconnectTwister()

```
void __stdcall piDisconnectTwister (
            PIHANDLE handle )
```

Disconnect from a USB Twister.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectTwister()

6.14.2.3 piGetTwisterMovingStatus()

```
int __stdcall piGetTwisterMovingStatus (
            BOOL * Moving,
            PIHANDLE handle )
```

Get a value indicating whether the twister is moving.

Parameters

Moving	[out] TRUE if the twister is moving, FALSE otherwise.
handle	[in] The device handle.

Returns

An error number.

See also

piRunTwisterToPosition()

6.14.2.4 piGetTwisterPosition()

```
int __stdcall piGetTwisterPosition (
            int * Position,
            PIHANDLE handle )
```

Get the position of the twister.

Parameters

Position	[out] The current position of the twister.
handle	[in] The device handle.

Returns

An error number.

The `Position` is a value between -32767 and +32767. See Twister Position.

See also

Twister Position

piRunTwisterToPosition()

piRunTwisterContinuous()

6.14.2.5 piGetTwisterSensorPosition()

```
int __stdcall piGetTwisterSensorPosition (
            int * SensorPosition,
            PIHANDLE handle )
```

Get the position of the twister sensor.

Parameters

SensorPosition	`[out]` The current position of the twister sensor.
handle	`[in]` The device handle.

Returns

An error number.

The `SensorPosition` is a value between 0 and 1023. See Twister SensorPosition.

See also

Twister Sensor Position

piGetTwisterStatusEx()

piRunTwisterToPosition()

piRunTwisterContinuous()

6.14.2.6 piGetTwisterStatus()

```
int __stdcall piGetTwisterStatus (
```

```
            int * Position,
            BOOL * Moving,
            PIHANDLE handle )
```

Get the position and status of the twister.

Parameters

Position	[out] The current position of the twister.
Moving	[out] TRUE if the twister is moving, FALSE otherwise.
handle	[in] The device handle.

Returns

An error number.

This method gets the current twister position and status in a single function call. It returns the value of the piGetTwisterPosition() and piGetTwisterMovingStatus() functions.

It is more efficient to call this function rather than calling 2 separate functions. Using this method reduces I/O traffic to the device and can improve the responsiveness of your application.

The Position is a value between -32767 and +32767. See Twister Position.

See also

Twister Position

piGetTwisterStatusEx()

piRunTwisterToPosition()

piRunTwisterContinuous()

6.14.2.7 piGetTwisterStatusEx()

```
int __stdcall piGetTwisterStatusEx (
            int * Position,
            int * SensorPosition,
            BOOL * Moving,
            PIHANDLE handle )
```

Get the position and status of the twister.

Parameters

Position	[out] The current position of the twister.
SensorPosition	[out] The current position of the twister sensor.
Moving	[out] TRUE if the twister is moving, FALSE otherwise.
handle	[in] The device handle.

Returns

An error number.

This method gets the current twister position and status in a single function call. It returns the value of the piGetTwisterPosition(), piGetTwisterSensorPosition(), and piGetTwisterMovingStatus() functions.

It is more efficient to call this function rather than calling 3 separate functions. Using this method reduces I/O traffic to the device and can improve the responsiveness of your application.

The `Position` is a value between -32767 and +32767. See Twister Position.

The `SensorPosition` is a value between 0 and 1023. See Twister Sensor Position.

See also

Twister Position

Twister Sensor Position

piGetTwisterStatus()

piRunTwisterToPosition()

piRunTwisterContinuous()

6.14.2.8 piGetTwisterVelocity()

```
int __stdcall piGetTwisterVelocity (
            int * Velocity,
            PIHANDLE handle )
```

Get the velocity of the twister.

Parameters

Velocity	`[out]` The velocity of the twister.
handle	`[in]` The device handle.

Returns

An error number.

See also

Twister Velocity

piRunTwisterToPosition()

piRunTwisterContinuous()

6.14.2.9 piHaltTwister()

```
int __stdcall piHaltTwister (
            PIHANDLE handle )
```

Stop the twister

Parameters

handle	[in] The device handle.

Returns

An error number.

Calling piHaltTwister() will stop the twister from moving.

See also

piRunTwisterToPosition()

6.14.2.10 piRunTwisterContinuous()

```
int __stdcall piRunTwisterContinuous (
            int Direction,
            int Velocity,
            PIHANDLE handle )
```

Start the Twister moving continuously.

Parameters

Direction	[in] The direction to move. Specify +1 (or larger) for motion in the positive direction. Specify 0 or any negative value for motion in the negative direction.
Velocity	[in] The twister velocity to use during the move (1 to 13).
handle	[in] The device handle.

Returns

An error number.

The twister position will be set to zero and will remain at zero during continuous motion.

See also

Twister Velocity

6.14.2.11 piRunTwisterToPosition()

```
int __stdcall piRunTwisterToPosition (
              int Destination,
              int Velocity,
              PIHANDLE handle )
```

Initiate a move to a destination position.

Parameters

Destination	[in] The destination twister position.
Velocity	[in] The twister velocity to use during the move (1 to 13).
handle	[in] The device handle.

Returns

An error number.

The system will initiate a move to the specified `Destination` position at the specified `Velocity`. This function does not wait for the move to complete.

The destination position can be any position between -32767 and +32767. See Twister Position.

See also

Twister Position

Twister Velocity

6.14.2.12 piSetTwisterPositionZero()

```
int __stdcall piSetTwisterPositionZero (
              PIHANDLE handle )
```

Set the twister position to zero.

Parameters

handle	[in] The device handle.

Returns

An error number.

Sets the current position to zero.

If the twister is moving, it will be halted before setting the position to zero.

See also

Twister Position

6.15 Valve Functions

Functions to control USB Valve devices.

Functions

- PIHANDLE __stdcall piConnectValve (int *ErrorNumber, int SerialNum)

  Connect to a USB Valve.
- void __stdcall piDisconnectValve (PIHANDLE handle)

  Disconnect from a USB Valve.
- int __stdcall piGetValveSensor (int *SensorValue, int SensorNumber, PIHANDLE handle)

  Get the state of the sensor.
- int __stdcall piGetValveStates (int *ValveStates, PIHANDLE handle)

  Get the state of the valves.
- int __stdcall piSetValveStates (int ValveStates, PIHANDLE handle)

  Set the state of the valves.

6.15.1 Detailed Description

Functions to control USB Valve devices.

6.15.2 Function Documentation

6.15.2.1 piConnectValve()

```
PIHANDLE __stdcall piConnectValve (
            int * ErrorNumber,
            int SerialNum )
```

Connect to a USB Valve.

Parameters

ErrorNumber	[out] An error number.
SerialNum	[in] The device serial number.

Returns

The device handle for the new device. If the device is not found or an error occurs, NULL is returned.

See also

piDisconnectValve()

6.15.2.2 piDisconnectValve()

```
void __stdcall piDisconnectValve (
            PIHANDLE handle )
```

Disconnect from a USB Valve.

Parameters

handle	[in] The device handle.

Returns

An error number.

See also

piConnectValve()

6.15.2.3 piGetValveSensor()

```
int __stdcall piGetValveSensor (
            int * SensorValue,
            int SensorNumber,
            PIHANDLE handle )
```

Get the state of the sensor.

Parameters

SensorValue	[out] The current value of the sensor, in the range 0 to 1023. The meaning of this value depends on what sensor you have attached to the device.
SensorNumber	[in] The sensor number you want to read. A value of 0 corresponds to sensor 1 and a value of 1 corresponds to sensor 2.
handle	[in] The device handle.

Returns

An error number.

6.15.2.4 piGetValveStates()

```
int __stdcall piGetValveStates (
            int * ValveStates,
            PIHANDLE handle )
```

Get the state of the valves.

Parameters

ValveStates	[out] The current state of the valves.
handle	[in] The device handle.

Returns

An error number.

ValveStates will be returned with a bit for each valve. Bit 0, the low order bit, corresponds to valve 1. Bit 1 corresponds to valve 2, etc. The bit will be 1 when the valve is ON (relay energized), and will be 0 when the valve is OFF (relay de-energized).

Immediately after powering up the Valve device, all valves are OFF (relay de-energized).

See also

piSetValveStates()

6.15.2.5 piSetValveStates()

```
int __stdcall piSetValveStates (
            int ValveStates,
            PIHANDLE handle )
```

Set the state of the valves.

Parameters

ValveStates	[in] The desired state of the valves.
handle	[in] The device handle.

Returns

An error number.

`ValveStates` contains a bit for each valve. Bit 0, the low order bit, corresponds to valve 1. Bit 1 corresponds to valve 2, etc. The bit should be 1 when the valve is set ON (relay energized), and should be 0 when the valve is set OFF (relay de-energized).

Immediately after powering up the Valve device, all valves are OFF (relay de-energized).

See also

piGetValveStates()

Index