USB-2200 Series

I/O Module User Manual

V1.0.1 August 2025

Written by Tommy Hong

Edited by Anna Huang

Warranty

All products manufactured by ICP DAS are under warranty regarding defective materials for a period of one year, beginning from the date of delivery to the original purchaser.

Warning

ICP DAS assumes no liability for any damage resulting from the use of this product. ICP DAS reserves the right to change this manual at any time without notice. The information furnished by ICP DAS is believed to be accurate and reliable. However, no responsibility is assumed by ICP DAS for its use, nor for any infringements of patents or other rights of third parties resulting from its use.

Copyright

Trademarks

Names are used for identification purposes only and may be registered trademarks of their respective companies.

Contact Us

If you have any problems, please feel free to contact us.
You can count on us for a quick response.
Email: service@icpdas.com

Contents

1. Introduction

This chapter provides an overview of the USB-2255-32/USB-2255-64 modules, including core concepts, key features, application fields, and hardware specifications. Users will gain a clear understanding of the product scope and the differences between the USB-2255-32 and USB-2255-64 models.

The USB-2255-32/ USB-2255-64 is a USB 2.0 digital I/O module equipped with 16/32 digital input and 16/32 digital output channels, capable of functioning as a 32-bit event counter. Inputs support selectable sink or source mode to accommodate different field devices. Each channel includes an LED status indicator, and the module supports startup/safe values, as well as hardware/software watchdogs to ensure reliable operation under adverse conditions.

With USB 2.0 high-speed interface, the module supports hot-swapping and driver-free installation, making it easy to integrate with embedded or industrial systems. Daisy-chaining of up to 5 nodes is supported via a built-in USB hub, with external power required for stable operation.

A full-featured API library and source code examples (VC, VB, BCB, .NET, LabVIEW) are available for Windows and Linux, enabling fast and flexible application development.

1.1. Features

This section lists the key functions and design highlights of the USB-2255-32/USB-2255-64, allowing users to quickly understand its advantages and value for various applications.

Key Features

- Up to 10 kS/s sampling rate
- Wide operating temperature range (-25°C to +75°C)
- USB 2.0 High-Speed (480 Mbps)
- Daisy-chaining supported (external power required)
- Driver-free installation (Plug-and-Play)
- Lockable USB cable design
- Built-in utility for easy configuration and I/O testing
- Built-in dual watchdog (hardware/software)
- API library provided (VC/VB/BCB/.NET)
- Windows 7/10/11 (32/64-bit) support

1.2. Applications

This section outlines typical application scenarios for the USB-2255-32/USB-2255-64, showcasing its versatility in automation and monitoring environments.

- **Building Automation:** Used for managing environmental control, energy systems, and access control systems.
- **Factory Automation:** Implementing process control, machine interfacing, and data collection on production lines.
- **Machine Automation:** Controlling and monitoring individual machinery.
- **Data Acquisition and Control:** Widely used in general data acquisition and control systems across various industries.
- **Environmental Monitoring:** Collecting sensor data under various conditions.
- **Laboratory Equipment and Research:** Performing precise data logging and experimental control.

1.3. Specifications

This section provides the complete technical specifications for the USB-2255-32/USB-2255-64, including general specifications and detailed I/O parameters for each model (USB-2255-32 and USB-2255-64).

1.3.1. General Specification

This section outlines the general hardware and environmental specifications of the USB-2255-32/USB-2255-64, covering communication interface, watchdog functions, LED indicator configuration, electromagnetic compatibility (EMC), and operating environment conditions.

Model	USB-2255-32	USB-2255-64
USB		
Specification	USB 2.0 High-Speed (480 Mbps)	
Interface	1 x USB IN (CA-USB-AC2-L108 CR); 1 x USB OUT (CA-USB-CC1-L003 CR) or (CA-USB-CC2-L0003 CR)	
Watchdog Timer		
Hardware Watchdog	1 (1.6 second)	
Software Watchdog	1 (Programmable)	
LED Indicators		
System LED Indicator	3 indicators (Power / Run / Error)	
I/O LED Indicator	1 LED per I/O channel	
EMS Protection		
ESD (IEC 61000-4-2)	4 kV contact for each terminal 8 kV air for random point	
Power		
Reverse Polarity Protection	Yes	
Powered from Terminal Block	+10~+30VDC	
Consumption	Typical 1.4 W Max. 1.9W	
Mechanical		
Dimensions (W x L x H)	31 mm x 166 mm x 129 mm	
Environmental		
Operating Temperature	-25 ~ +75 °C	
Storage Temperature	-40 ~ +85 °C	
Relative Humidity	10 ~ 95% RH, non-condensing	

1.3.2. I/O Specification

This section presents the detailed specifications of the USB-2255-32/USB-2255-64 model, including digital input and output channel counts, input types and electrical parameters, output driving capability, and isolation protection.

Model		USB-2255-32	USB-2255-64
Digital Input			
Channels		16	32
Input Type	Dry Contact	Source	
	Wet Contact	Sink/Source	
ON Voltage	Dry Contact	Close to GND	
	Wet Contact	+10 ~ +50 VDC	
OFF Voltage	Dry Contact	Open	
	Wet Contact	+4 VDC	
Max. Input Frequency		5k Hz	
Counter		32-bit	
Digital Filter Width		1-65000* 100 µ (1~6500 ms) Default=0, Disable=0	
Input Impedance		10 kΩ	
Overvoltage Protection		70 VDC	
Intra-Module Isolation		3000 VDC	
Effective Distance		Up to 500 meters	
Digital Output			
Channels		16	32
Output Type		Open Collector (Sink, NPN)	
Load Voltage Range		+3.5 ~ +50 VDC	
Max. Load Current		600 mA/Channel	500 mA/Channel
Overvoltage Protection		60 VDC	
Overload Protection		1.4 A with short-circuit protection	
Power-on Value		Programmable	
Safe Value		Programmable	

1.4. Overview

This section provides an overview of the module's physical components and layout. Understanding the location and function of each part will assist in proper wiring, installation, and maintenance.

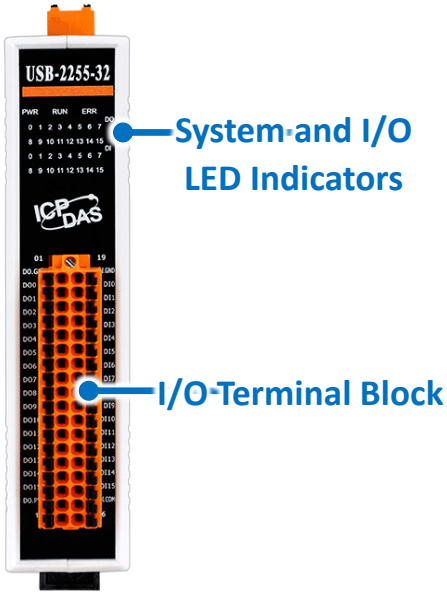
The figure below shows the front, rear and side views of the module, showing the locations of all major components, including the I/O terminal blocks, USB ports, power input, and LED indicators. Refer to the labeled components for wiring and installation.

System and I/O LED Indicators

LED indicators provide real-time status information for power, communication, and errors, as well as the I/O channel states.
DIO Signal LEDs – LED ON when channel is active (ON); OFF when inactive (OFF).

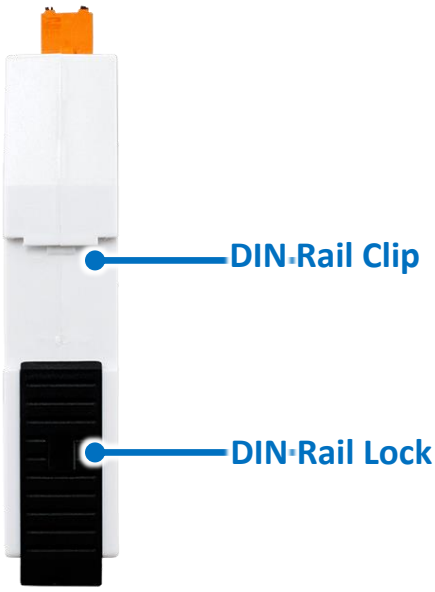
I/O Terminal Block

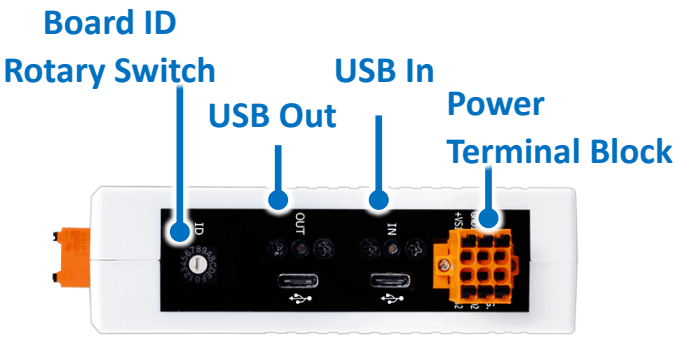
The I/O terminal block provides the physical connection points for digital input and output channels, as well as power supply lines for output control.



Din Rail Clip & DIN Rail Lock

The DIN rail clip and lock mechanism allows secure and tool-free installation of the module on a standard DIN rail, ensuring stability in industrial environments.
The DIN Rail Clip allows the module to be easily mounted on a standard DIN rail.
The DIN Rail Lock secures the module in place to prevent accidental removal.





Board ID Rotary Switch

The Board ID rotary switch allows each module to be assigned a unique identification number, preventing communication conflicts when multiple identical modules are connected.

The **Board ID** is used to distinguish between multiple modules of the same product number connected to the same computer.

When two or more identical modules are connected, each must have a unique Board ID to avoid conflicts.

The Board ID can be set via the rotary switch:

- **Hardware setting: Values 1–15**
- **Software setting: Values 16–127 (when rotary switch is set to 0)**

USB In & USB Out

These ports allow the module to connect to a host device or other modules in a daisy-chain configuration, supporting flexible system expansion.

Port Functions

- **USB IN: Connects to a PC, PAC, or the USB OUT port of another USB-2200 series module.**
- **USB OUT: Connects to the USB IN port of another module to form a daisy-chain.**

Power Terminal Block

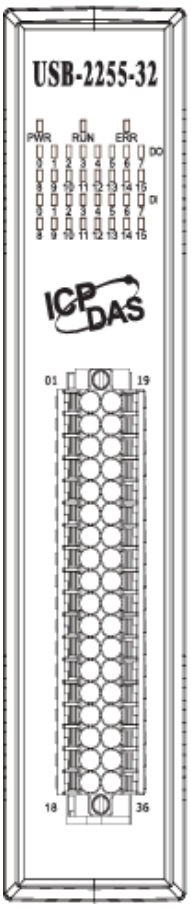
The power terminal block provides the external power connection required for module operation, supporting redundancy for improved reliability.

1.5. Pin Assignments

This section lists the pin assignments and functional descriptions for each connector of the USB-2255-32 and USB-2255-64, including digital input, digital output, and power terminals, to assist users during wiring and system design.

USB-2255-32

This figure below describes the pin definitions and signal functions of the I/O terminal block.

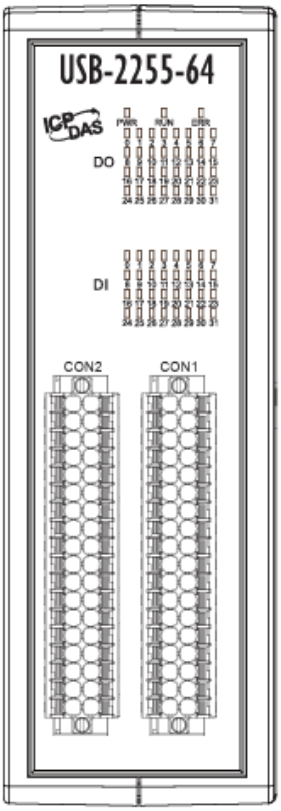


Pin Assignment	Terminal No.		Pin Assignment
DO.GND	01	19	DI.GND
DO0	02	20	DI0
DO1	03	21	DI1
DO2	04	22	DI2
DO3	05	23	DI3
DO4	06	24	DI4
DO5	07	25	DI5
DO6	08	26	DI6
DO7	09	27	DI7
DO8	10	28	DI8
DO9	11	29	DI9
DO10	12	30	DI10
DO11	13	31	DI11
DO12	14	32	DI12
DO13	15	33	DI13
DO14	16	34	DI14
DO15	17	35	DI15
DO.PWR	18	36	DI.COM

Pin	Description
DI(N)	The Digital Input for Channel N
DI.GND	The Ground of Current Path of Dry Contact
DI.COM	The Common Pin for Wet Contact
DO(N)	The Digital Output for Channel N
DO.PWR	The Power Source Input Pin
DO.GND	The Ground of Power Source Input Pin

USB-2255-64

This figure below describes the pin definitions of the CON1 terminal.



Pin Assignment	Terminal No.		Pin Assignment
DO.GND1	01	19	DI.GND1
DO0	02	20	DI0
DO1	03	21	DI1
DO2	04	22	DI2
DO3	05	23	DI3
DO4	06	24	DI4
DO5	07	25	DI5
DO6	08	26	DI6
DO7	09	27	DI7
DO8	10	28	DI8
DO9	11	29	DI9
DO10	12	30	DI10
DO11	13	31	DI11
DO12	14	32	DI12
DO13	15	33	DI13
DO14	16	34	DI14
DO15	17	35	DI15
DO.PWR1	18	36	DI.COM1

This figure below describes the pin definitions of the CON2 terminal.

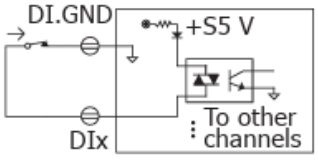
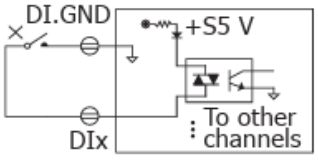
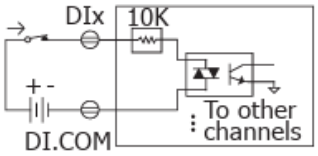
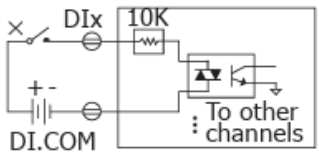
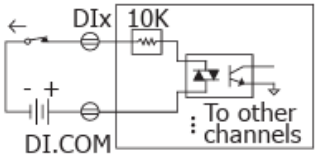
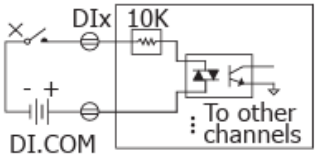
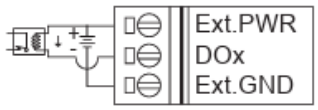
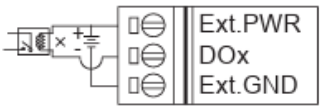
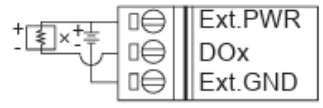
Pin Assignment	Terminal No.		Pin Assignment
DO.GND2	01	19	DI.GND2
DO16	02	20	DI16
DO17	03	21	DI17
DO18	04	22	DI8
DO19	05	23	DI9
DO20	06	24	DI20
DO21	07	25	DI21
DO22	08	26	DI22
DO23	09	27	DI23
DO24	10	28	DI24
DO25	11	29	DI25
DO26	12	30	DI26
DO27	13	31	DI27
DO28	14	32	DI28
DO29	15	33	DI29
DO30	16	34	DI30
DO31	17	35	DI31
DO.PWR2	18	36	DI.COM2

This figure below describes the signal functions of the CON1/CON2 terminal.

Pin	Description
DI(N)	The Digital Input for Channel N
DI.GND(N)	The Ground of Current Path of Dry Contact for CON N
DI.COM(N)	The Common Pin for Wet Contact for CON N
DO(N)	The Digital Output for Channel N
DO.PWR	The Power Source Input Pin for CON N
DO.GND	The Ground of Power Source Input Pin for CON N

1.6. Wiring Connections

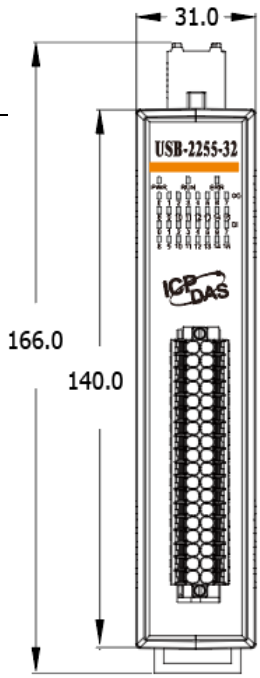
This section provides wiring methods for the digital inputs and outputs of the USB-2255-32 and USB-2255-64, explaining the application scenarios for dry contact and wet contact configurations.

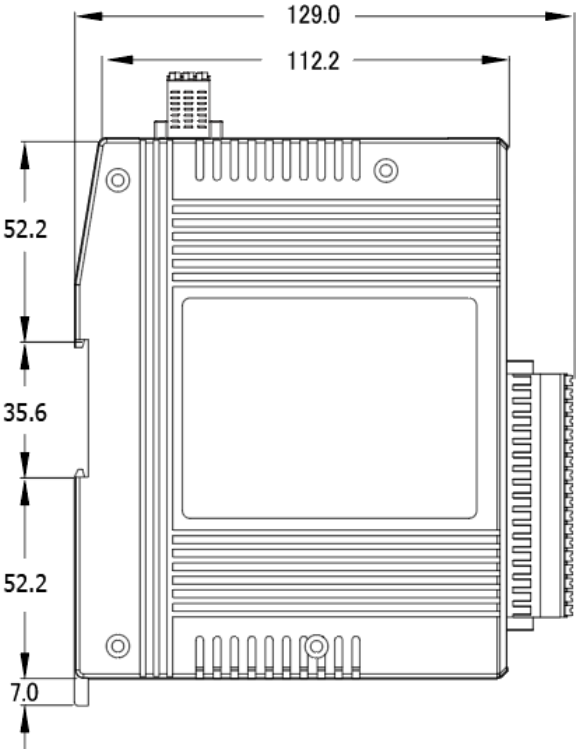
Digital Input /Counter	Readback as 1	Readback as 0
Dry Contact		
Wet Contact (Sink)		
Wet Contact (Source)		
Output Type	**ON State Readback as 1**	**OFF State Readback as 0**
Inductive Load		
Resistance Load		

1.7. Dimension

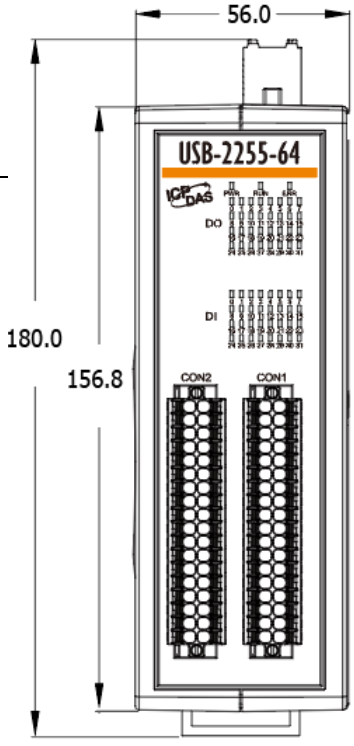
This section provides the dimension diagrams of the USB-2255-32 and USB-2255-64 modules, helping users plan mechanical design and installation.

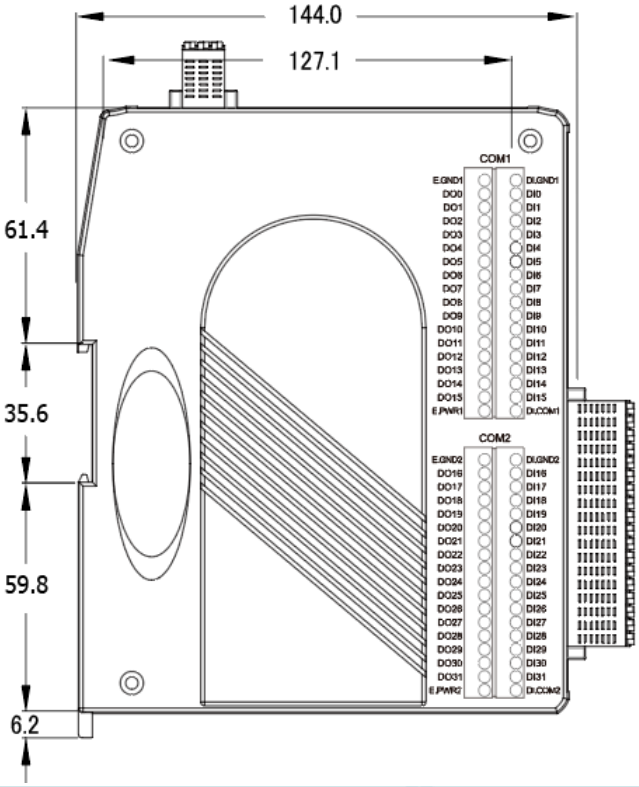
USB-2255-32





USB-2255-64





2. Getting Started

This chapter guides users through hardware connection, driver installation, utility testing, and SDK/API deployment for the USB-2255-32/USB-2255-64, ensuring users can quickly start and develop applications.

After receiving the USB-2255-32/USB-2255-64, please verify that all components are included and undamaged. If any item is missing or defective, contact your supplier or distributor immediately for assistance.

USB I/O Module

USB Cable

Screw Driver

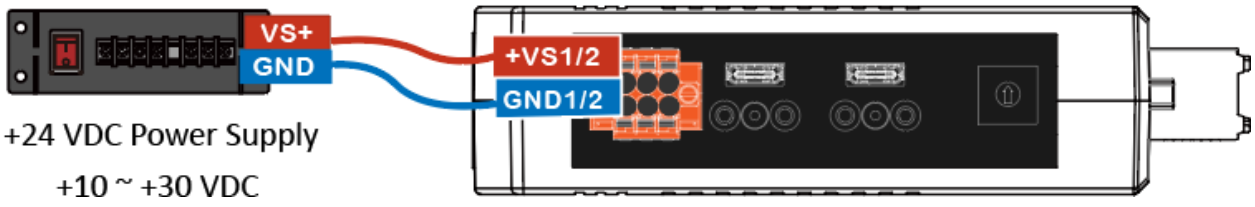
Quick Start Guide

2.1. Hardware Installation

This section explains how to connect the USB-2255-32/USB-2255-64 to your PC for the first time. It includes the detection process of the USB device as a removable drive, and introduces how the Board ID is reflected in the device name.

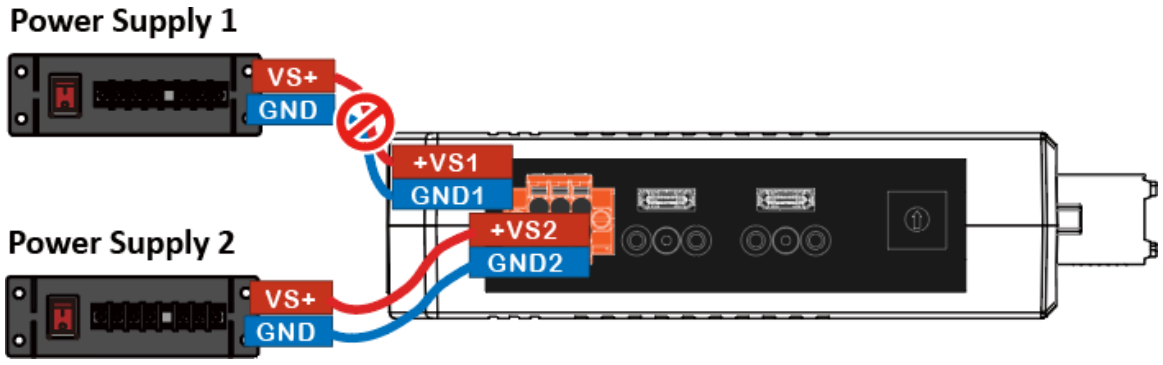
Step 1. Connect the power supply

- Connect the positive terminal (+) of the power supply to the terminal +VS1/2
- Connect the negative terminal (-) of the power supply to the GND1/2



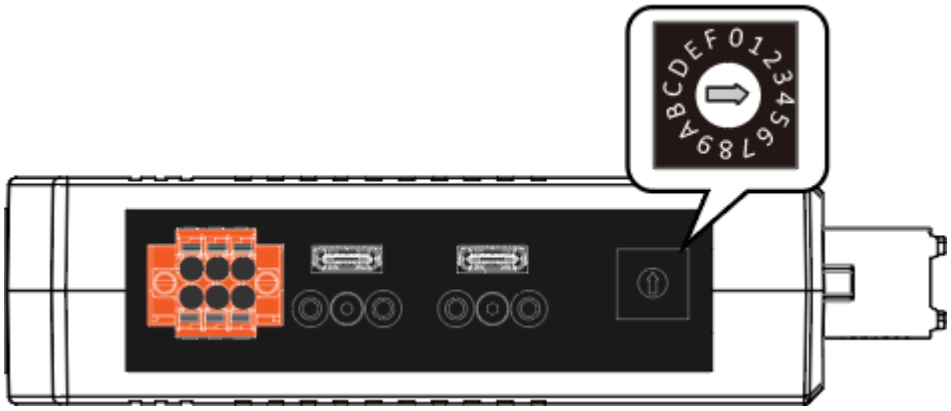
Tips & Warnings

1. The USB-2255-32/USB-2255-64 provides two independent power inputs that can be connected simultaneously.
2. If one power source fails, the other will automatically take over, ensuring redundant power backup and non-stop operation.



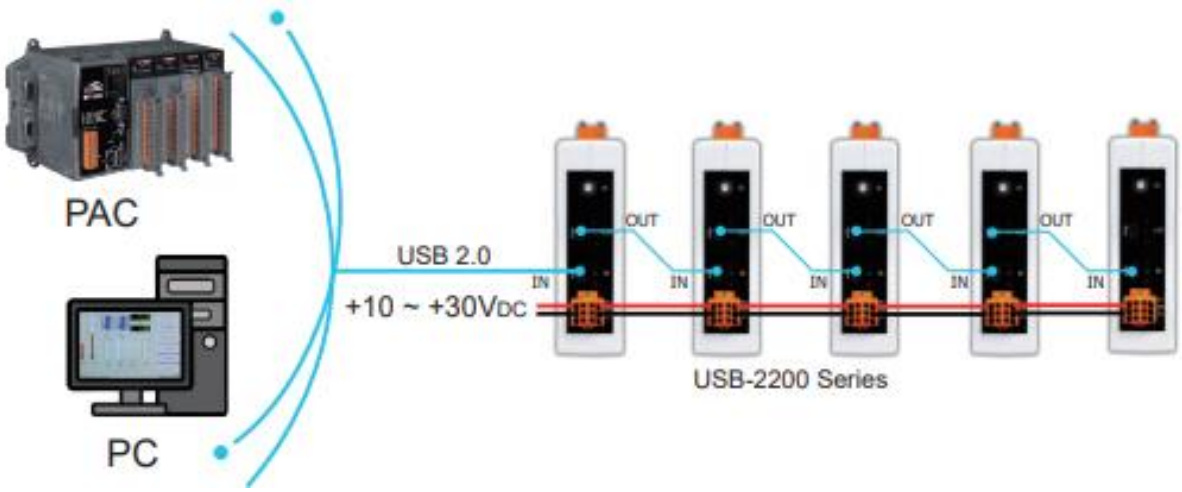
Step 2. Set the Board ID

- Configure the Board ID using the DIP switch or rotary switch (depending on the model).
- Each USB-2255-32/USB-2255-64 in the system must have a unique Board ID when multiple modules are connected.



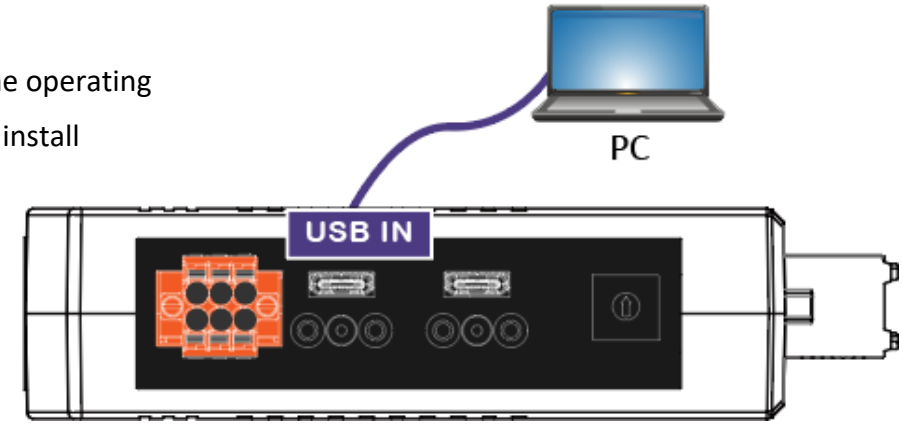
Tips & Warnings

- When two or more identical modules are connected, each must have a unique Board ID to avoid conflicts.
- The USB-2255-32/USB-2255-64 supports Daisy-Chain connection, allowing multiple modules to be connected in sequence. When using Daisy-Chain, make sure that each module has a unique Board ID for correct identification and communication.



Step 3. Connect the USB Cable

- Use the provided USB Type-B to Type-A cable to connect the USB-2255-32/USB-2255-64 to the host PC.
- Upon the first connection, the operating system will automatically install the required driver.

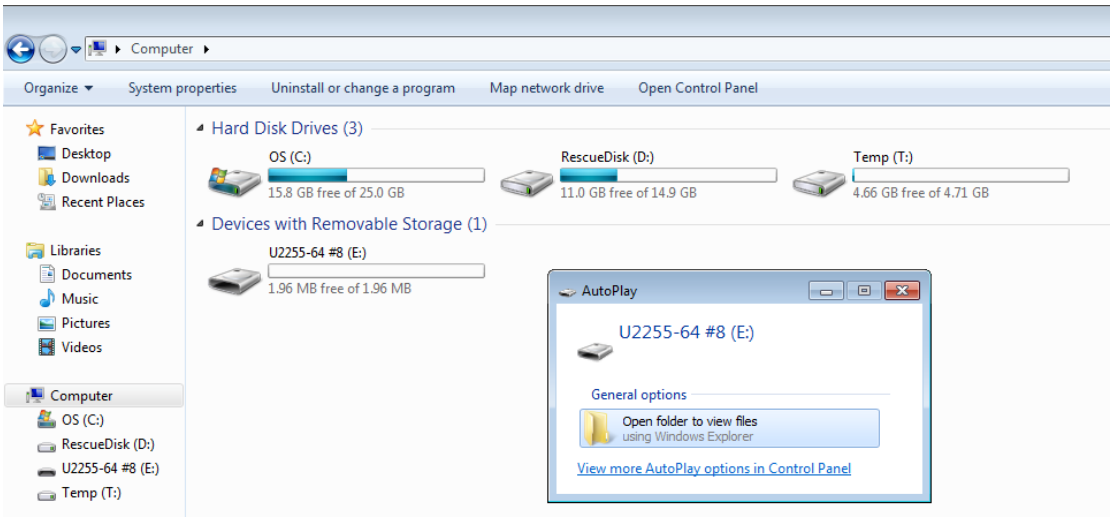


Tips & Warnings

After successful installation, the system will recognize the module as a removable disk. The device name will appear as:

"U22XX #N" or "U22XX-XX #N"

- **22XX / 22XX-XX indicates the module model (e.g., USB-2255-32/USB-2255-64).**
- **#N represents the assigned Board ID.**



Step 4. Verify Power Status

- Confirm that the Power LED indicator on the USB-2255-32/USB-2255-64 is lit.
- If the LED is not illuminated, recheck the power connections.

2.2. Software Installation

This section explains how to download, install, and run the USB I/O Utility and related SDK for functional testing and software development. Users will learn how to obtain the required software package, install it properly, and begin using the USB I/O Utility for device testing and configuration.

Step 1. AccessDownload Center

Go to the ICP DAS official **Download Center**:
https://www.icpdas.com/en/download/index.php?model=USB-2255%20series

Step 2. Select the SDK and utility package

In the Download Center, select the SDK and utility package according to your **operating system**. Choosing the correct version ensures compatibility and proper functionality.

File name

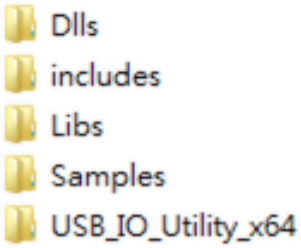
- **64-bit Windows: USB I/O Utility and Demo (x64)**
- **32-bit Windows: USB I/O Utility and Demo (x86)**

Demo Sample

	FILE NAME	DESCRIPTION	MODEL	LAST UPDATE
	USB I/O Utility and Demo (x64)	USB I/O Utility and VC、DotNet (C#)、LabVIEW、Python demo for 64-bit Windows 10 and later	USB-2255 series	2025-08-14
	USB I/O Utility and Demo (x86)	USB I/O Utility and VC、DotNet (C#)、LabVIEW、Python、Delphi、VB6、BCB demo for Winodws x86 platform	USB-2255 series	2025-08-06
	For Linux USB I/O demo	USB I/O demo for Linux	USB-2255 series	2025-08-06

Step 3. Download and Extract the File

Download the selected package and extract the contents.



The software package includes:

- DLLs **– Dynamic link libraries required for device communication.**
- Includes **– Header files for software development integration.**
- Libs **– API libraries for compiling user applications.**
- Samples **– Example source codes demonstrating program usage.**
- USB_IO_Utility_ x86/x64 **– Executable utility for configuring, testing and diagnosing the USB-2255-32/USB-2255-64 on 32/64-bit Windows systems.**

Step 4. Launch the USB I/O Utility

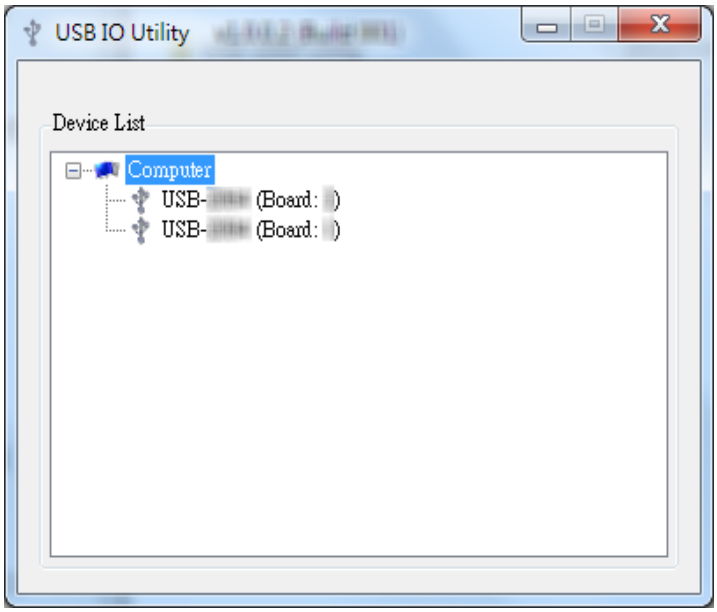
- **The USB I/O Utility allows users to test and acquire data from all ICP DAS USB series I/O modules without programming.**
- **The executable file is located in:**

  extracted folder\USB_IO_Utility_x64 (for 64-bit Windows)

  extracted folder\USB_IO_Utility_x32 (for 32-bit Windows)

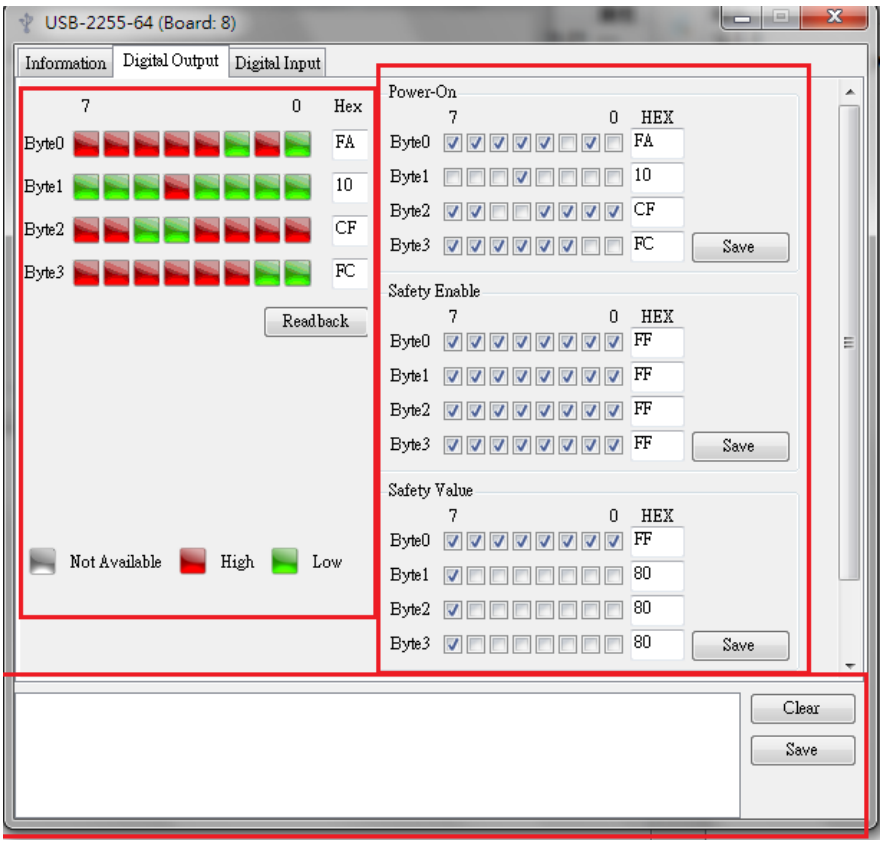
Step 5. Select the Device

- When the USB I/O Utility is opened, all ICP DAS USB series I/O modules connected to the PC are automatically scanned and listed in the Device List.
- Modules will appear in the list when plugged in and will be removed when disconnected.



Step 6. Configure the I/O Settings

- After selecting the device, configure the I/O settings according to application requirements.
- For detailed instructions on configuration and usage of the USB I/O Utility, refer to Chapter 3. USB I/O Utility.



3. USB I/O Utility

This chapter describes the installation and operation of the USB I/O Utility included in the software package. The utility enables users to perform diagnostic tests, configure I/O channels, and monitor device information without programming.

When accessing a module, the USB I/O Utility provides several function tabs, including:

- **Information Tab**
  Displays device name, firmware version, Board ID, watchdog timer settings, description, and provides the option to restore factory defaults.

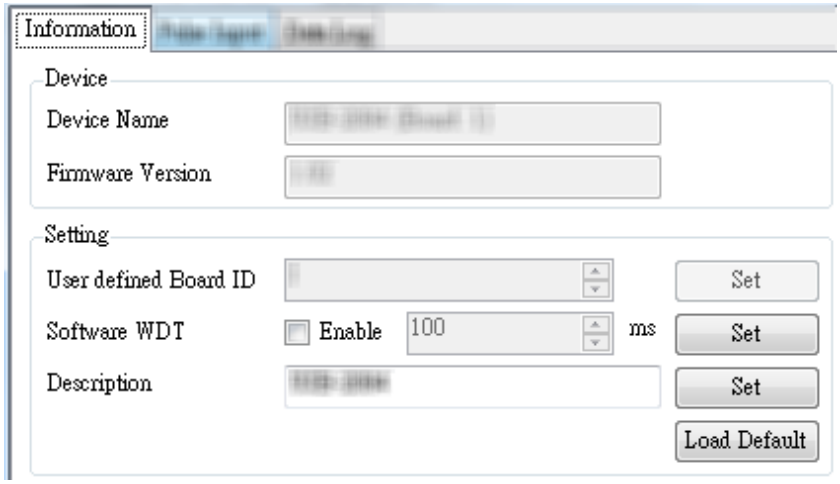
- **Digital Output Tab**
  Allows users to write output values, configure power-on settings, safety values, and invert output logic.

- **Digital Input Tab**
  Enables monitoring of input values with adjustable polling intervals, filter width, counter triggers, and input inversion.

To open and launch the USB I/O Utility, please refer to **Section 2.2 Software Installation** for detailed installation and startup instructions.

3.1. Information

The **Information Tab** provides device-level system details and configuration options. It allows users to view essential information, manage the watchdog timer, and restore factory defaults.



Functions

- **Device Name**

  Displays the name of the opened device.

- **Firmware Version**

  Shows the current firmware version.

- **User Defined Board ID**

  Allows modification of the Board ID (valid range: 16 – 127).

- **Software WDT**

  Enables the watchdog timer to monitor module status (100 ms ~ 30 min). If enabled, the PC and module exchange SYNC packets; upon failure, the system can output a safety value.

- **Description**

  User-defined description of the module (up to 32 characters).

- **Load Default**

  Restores the device to factory default settings.

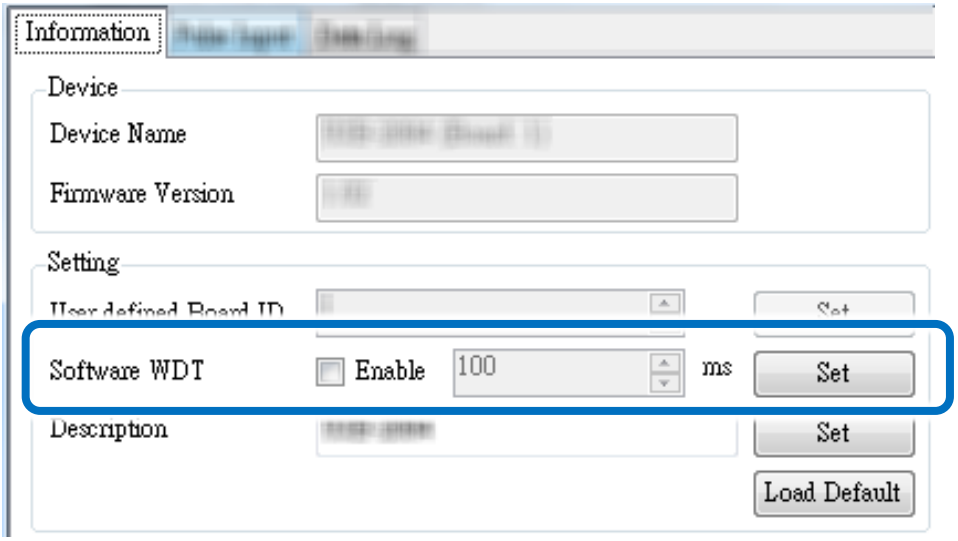
3.1.1. Host Watchdog

Function:

Prevents system errors by monitoring host status. On timeout, output returns to safe values.

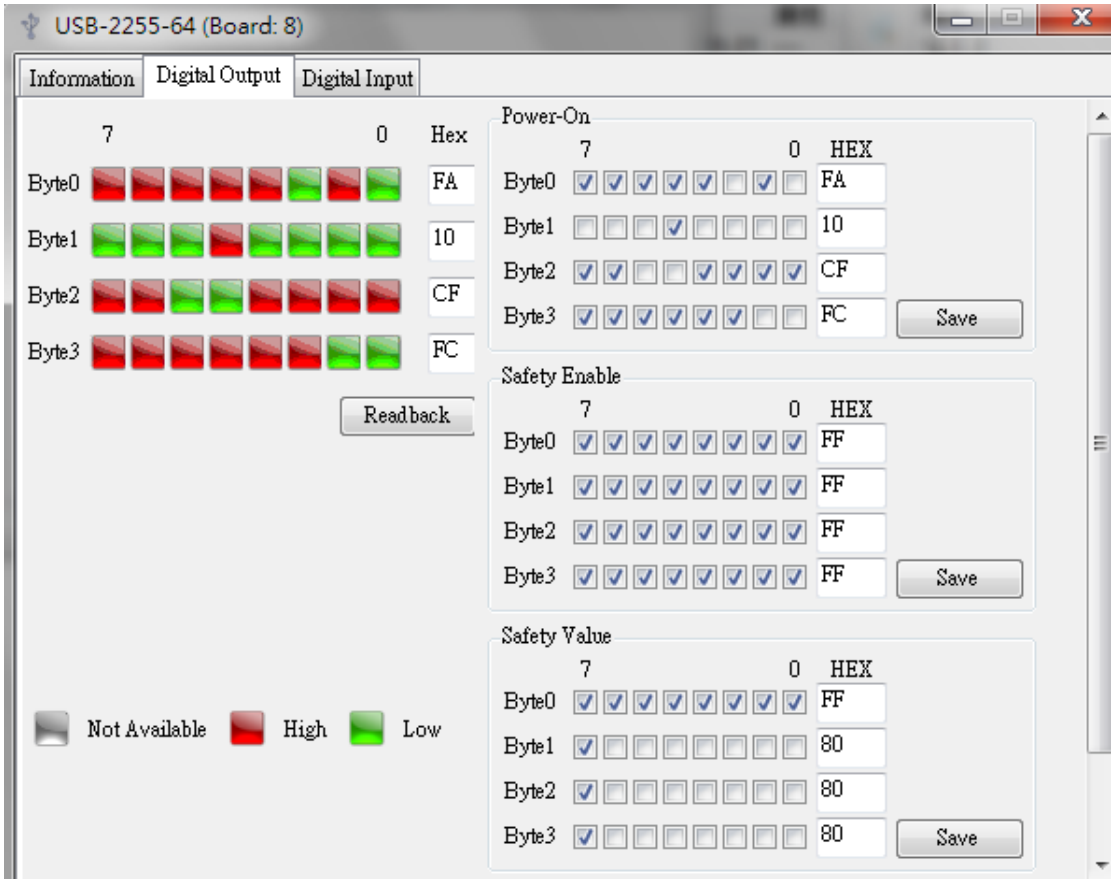
Steps:

1. Enable WDT on Information Page.
2. Select reset timeout interval.
3. Click Set to save.



3.2. Digital Output

The Digital Output tab enables users to write output values, configure startup and safety conditions, and adjust output logic.



Functions

- **Write Output**

  Set output values in hexadecimal format.

- **Readback**

  Displays current output values.

- **Power-On Value**

  Defines output states during startup or reset.

- **Safe Value**

  Defines outputs when host communication fails.

- **DO Output Inverse**

  Inverts DO logic signals.

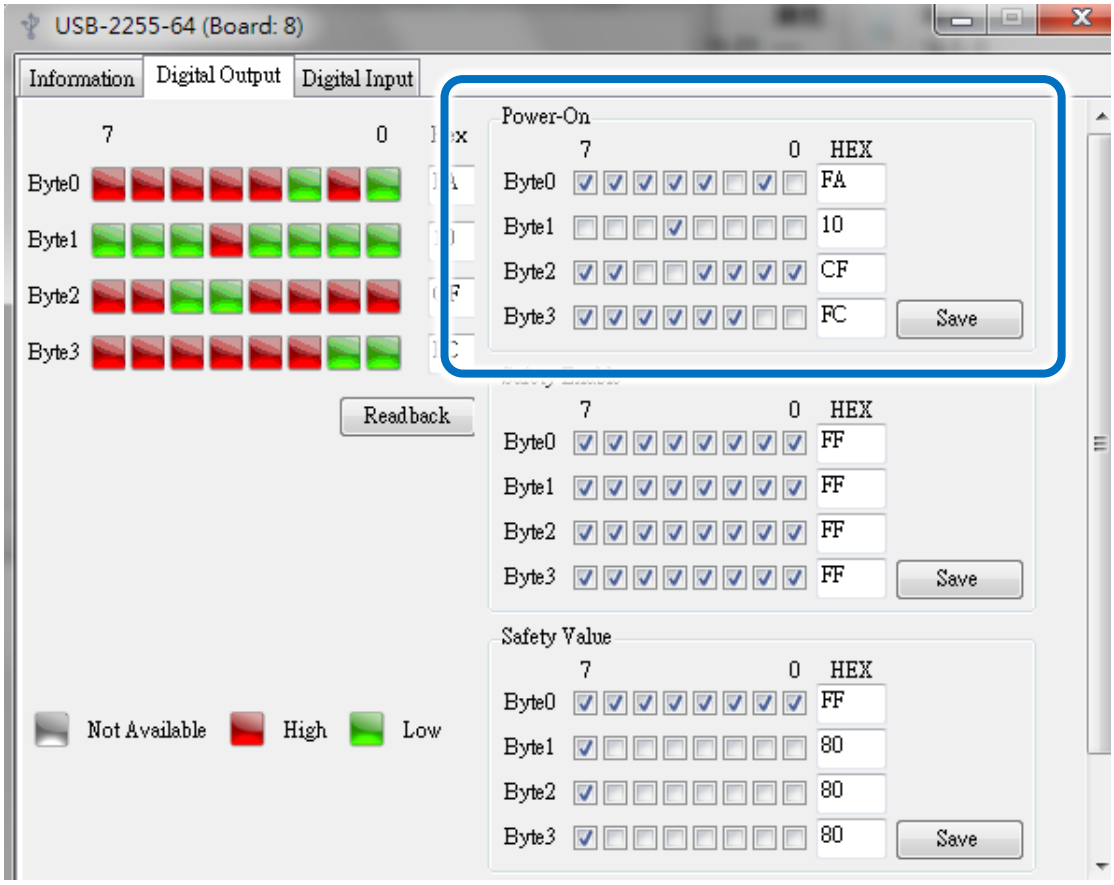
3.2.1. Power-on Value

Function:

Defines the default DO states on startup, reset, or watchdog reset. Stored in EEPROM.

Step:

1. Select checkboxes or enter Hex value.
2. Click Save to store.



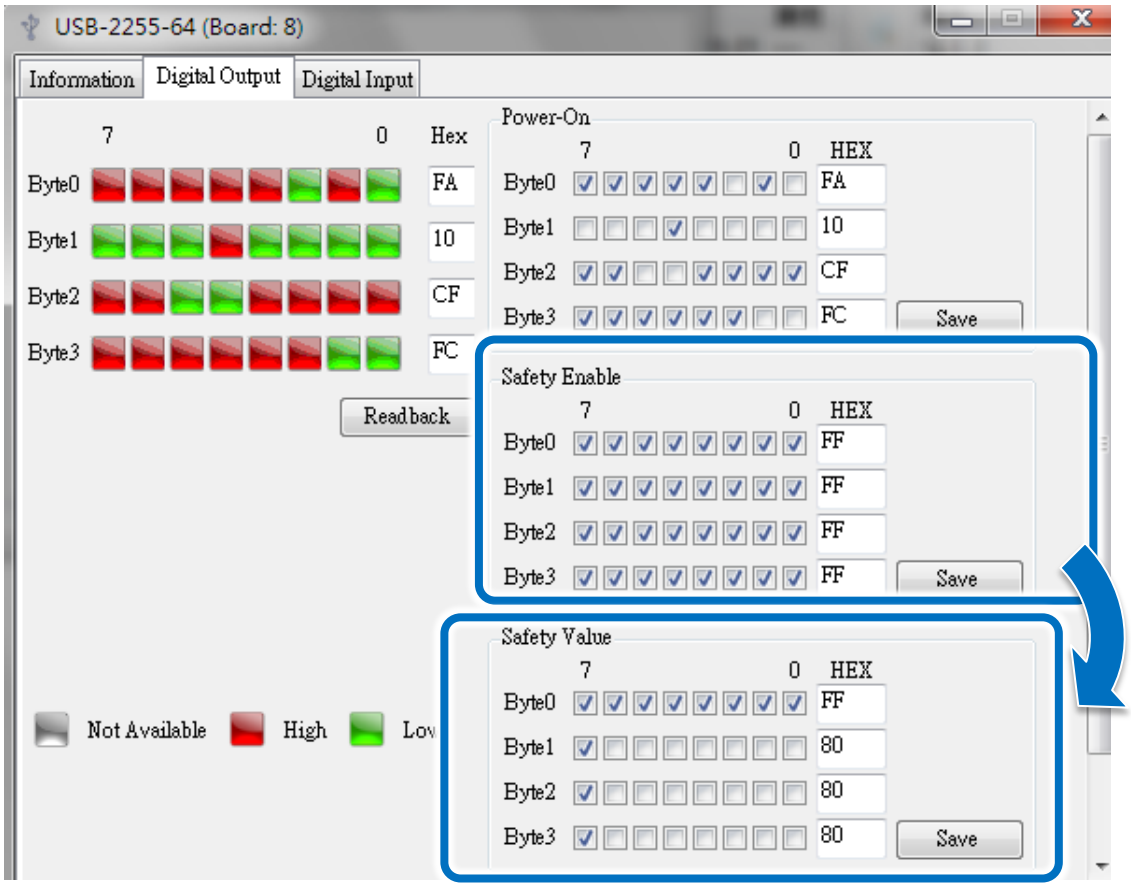
3.2.2. Safe Value

Function:

Ensures safe operation by outputting predefined values when host connection is lost.

Step:

1. Enter Hex value or select check box to enable the safe value function and then click save to store.
2. Enter Hex value and select check box to set the safe value and then click save to store.



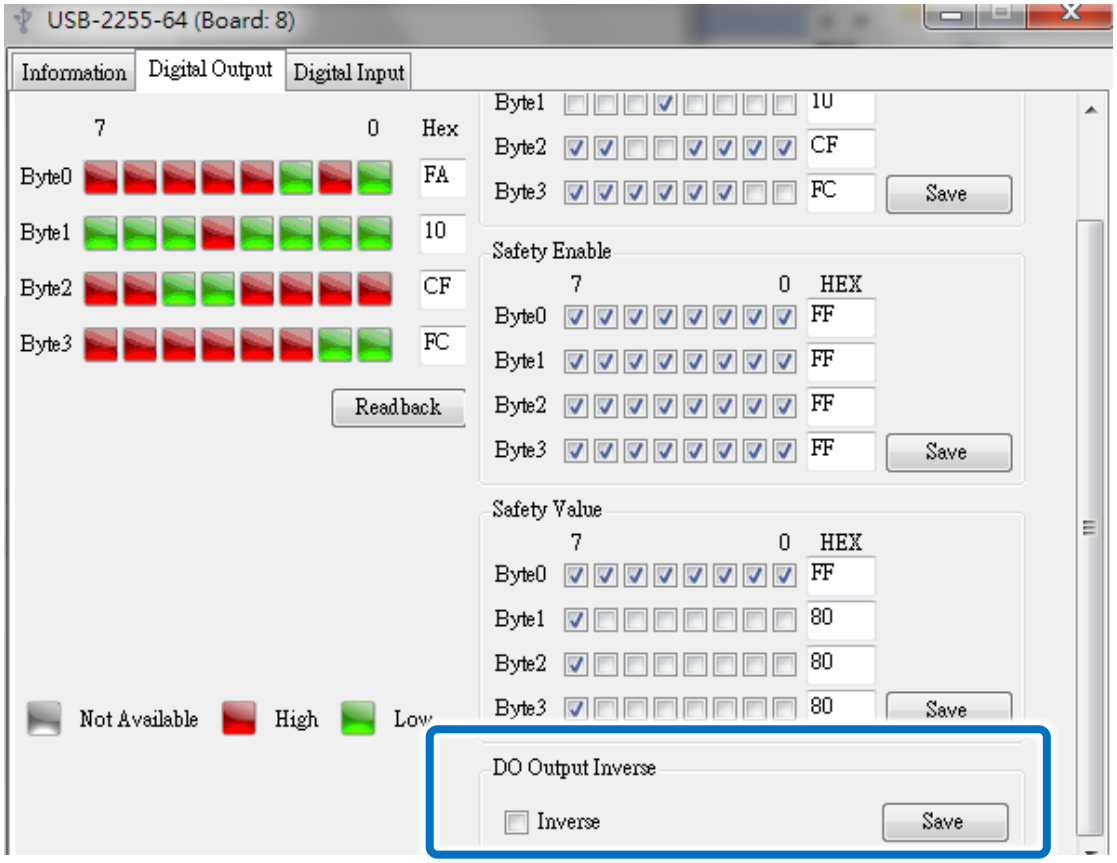
3.2.3. DO Inverse

Function:

Inverts the DO signal logic for compatibility with active-low systems.

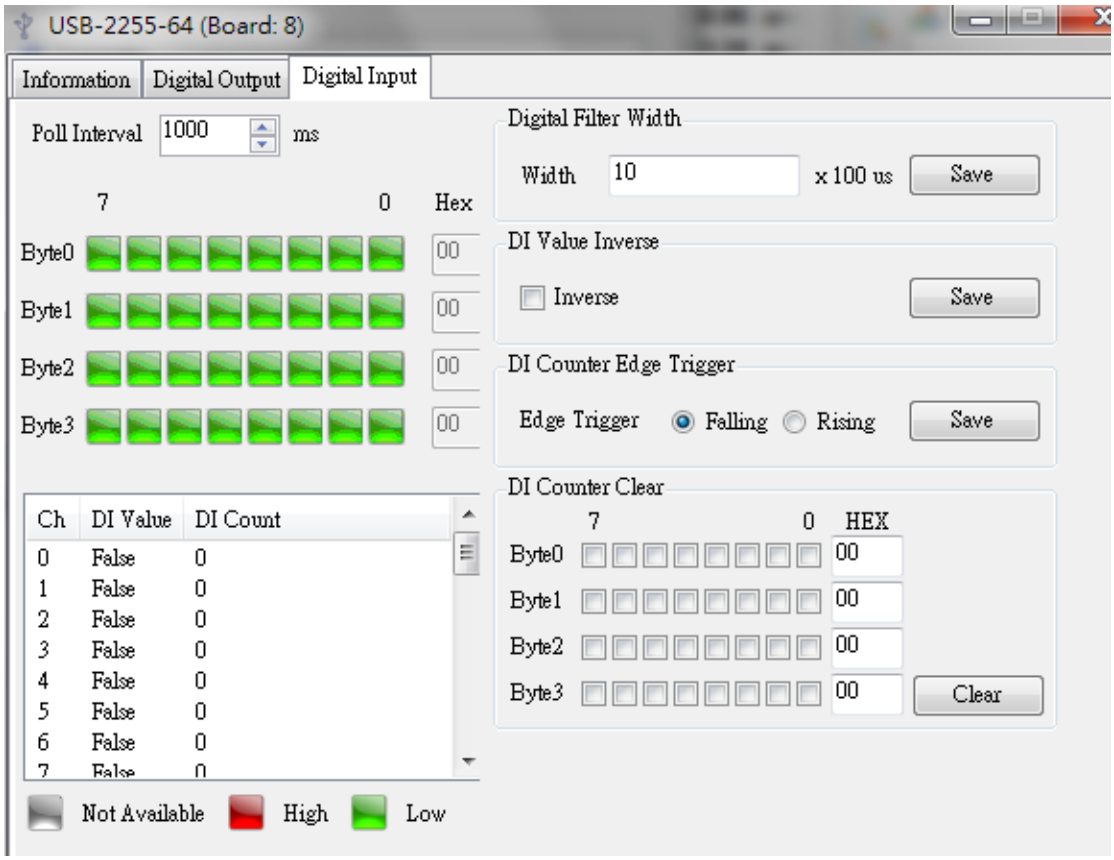
Step:

1. Select Inverse option.
2. Click Save to apply.



3.3. Digital Input

The Digital Input tab enables users to monitor DI signals, configure filters, and apply logic inversion or counters.



Functions

- **Polling Interval**

  Defines polling rate (100–5000 ms).

- **DI Filter**

  Eliminates switch bounce and noise.

- **DI Inverse**

  Inverts input logic.

- **DI Counter**

  Supports edge trigger and reset.

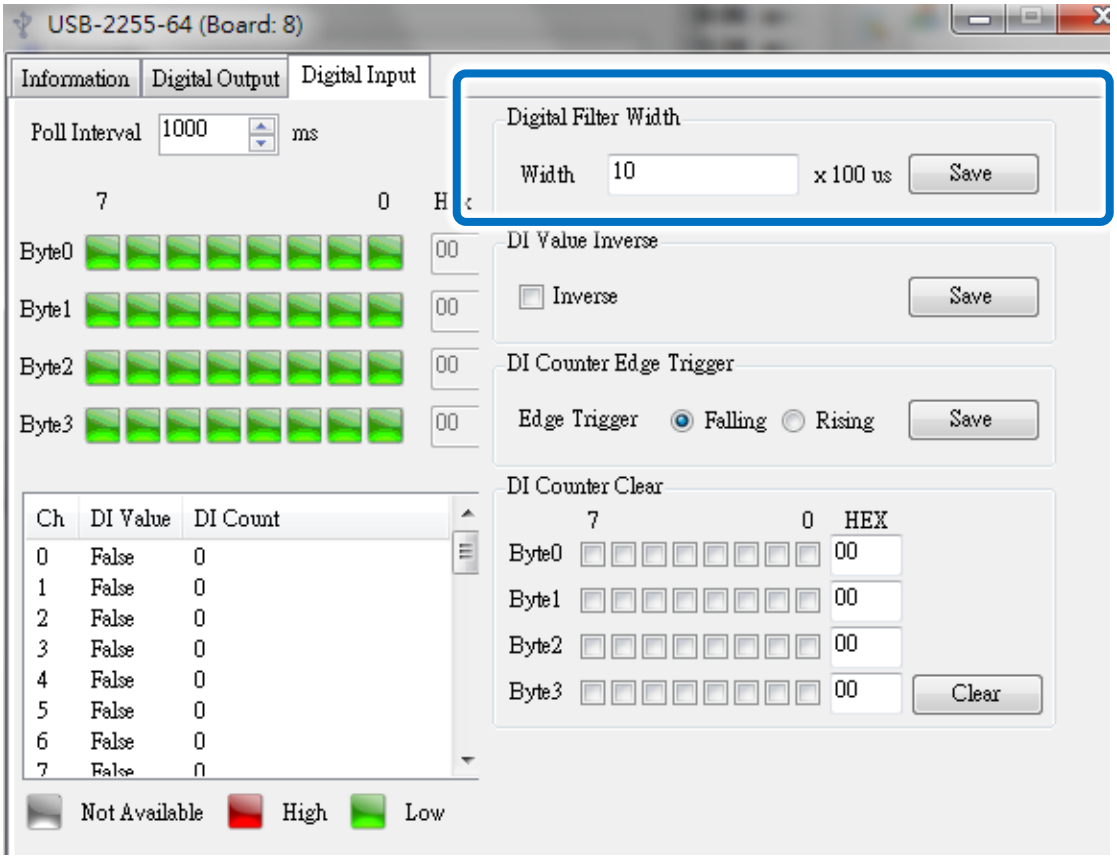
3.3.1. DI Filter

Function:

A low-pass filter removes short pulses and noise. Adjustable from 1–65535 × 100 µs (0 = Disabled).

Step:

1. Enter filter value.
2. Click Save to apply.



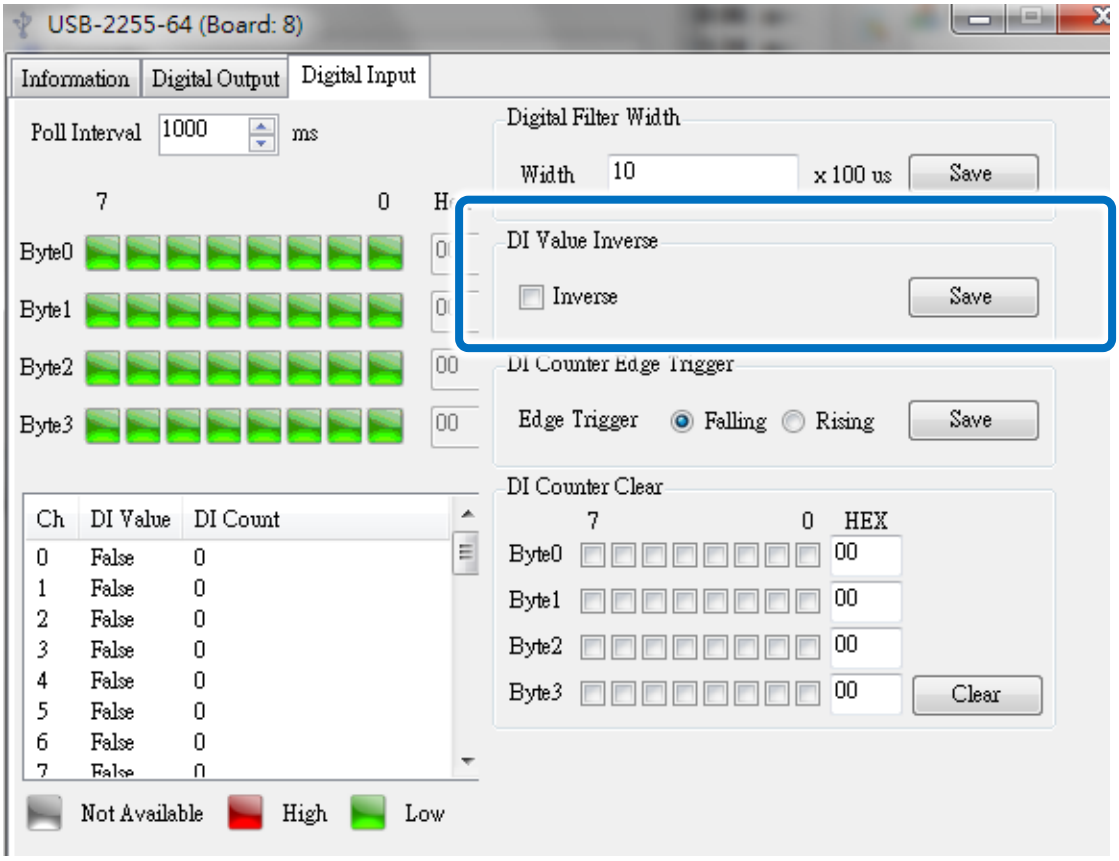
3.3.2. DI Inverse

Function:

Inverts DI logic level for active-low signals.

Step:

1. Select DI Inverse option.
2. Click Save to apply.



4. Operation & Deployment

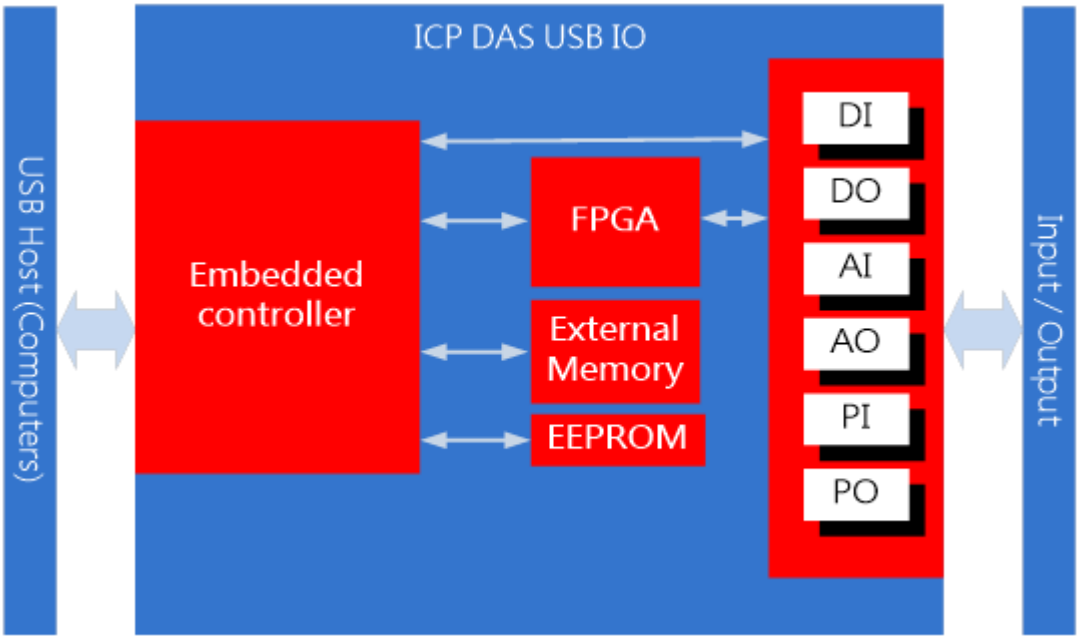
This chapter explains the operation and deployment process of the USB-2255-32/USB-2255-64. It covers the hardware and software architecture, SDK deployment, software structure, and class library operations, ensuring developers can quickly integrate the module into applications and use the APIs correctly.

4.1. Hardware Architecture

This section introduces the internal hardware design of the USB-2255, including its embedded architecture and I/O management components.

The USB-2255-32/USB-2255-64 uses an embedded control architecture and connects to the host computer via USB. It consists of an embedded controller, FPGA, memory, and EEPROM, which manage I/O interfaces such as DI, DO, AI, AO, PI, and PO.

The following diagram illustrates the overall hardware block structure and signal flow of USB-2255-32/USB-2255-64.



4.2. SDK Deployment

This section introduces the SDK resources provided for the USB-2255-32/USB-2255-64, including software packages, development environment integration, sample programs, and Linux support. Each subsection explains the available files, their purposes, and the correct method to integrate them into user applications.

4.2.1. Windows Development Environment

This section describes how to integrate the USB-2255-32/USB-2255-64 SDK into different development environments, including .NET, Visual C++, Visual Basic, and Python. Each environment is provided with corresponding DLLs, header files, libraries, and sample codes. By following the integration guidelines, developers can ensure smooth operation of the USB-2255-32/USB-2255-64 APIs in their projects.

Windows Software Package

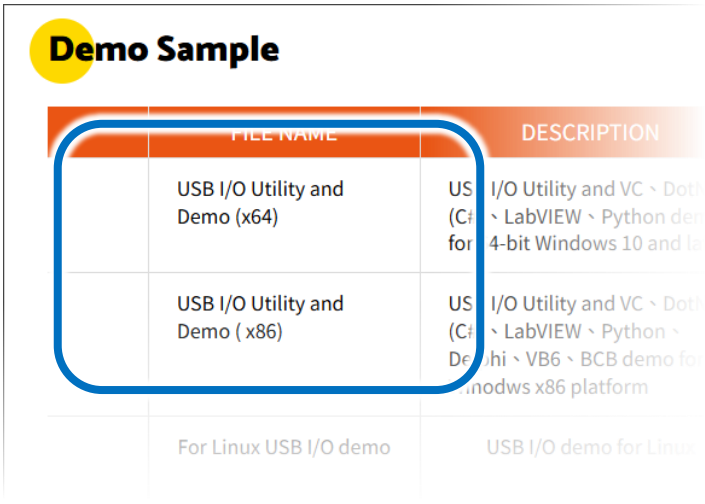
The Windows software package contains drivers, DLLs, libraries, header files, and demo programs for Windows platforms (x64 and x86). It allows developers to quickly build projects using environments such as Visual Studio, .NET, Python, LabVIEW, and Qt.

Download Location

The latest Windows software package can be downloaded from the ICP DAS official website:
https://www.icpdas.com/en/download/index.php?model=USB-2255%20series

File name

- USB IO Utility and Demo (x64)
- USB IO Utility and Demo (x86)



Package Contents:

- **Dlls**

  Dotnet_DLL_x64/x86 : DLLs for .NET development
  System_DLL : Core system DLLs
  VC_DLL_x64 /x86: DLLs for Visual C++ development

- **includes**

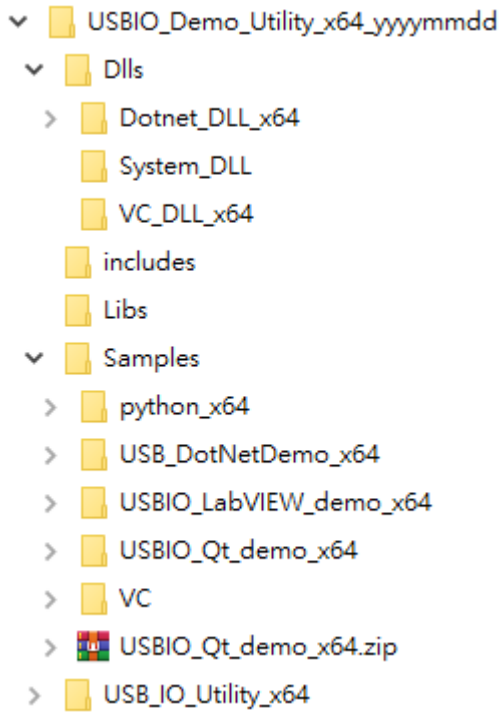
  Header files required for C/C++ development.

- **Libs**

  API libraries used for compilation and linking.

- **Samples**

  Example projects for .NET, Python, LabVIEW, VC++, and Qt.

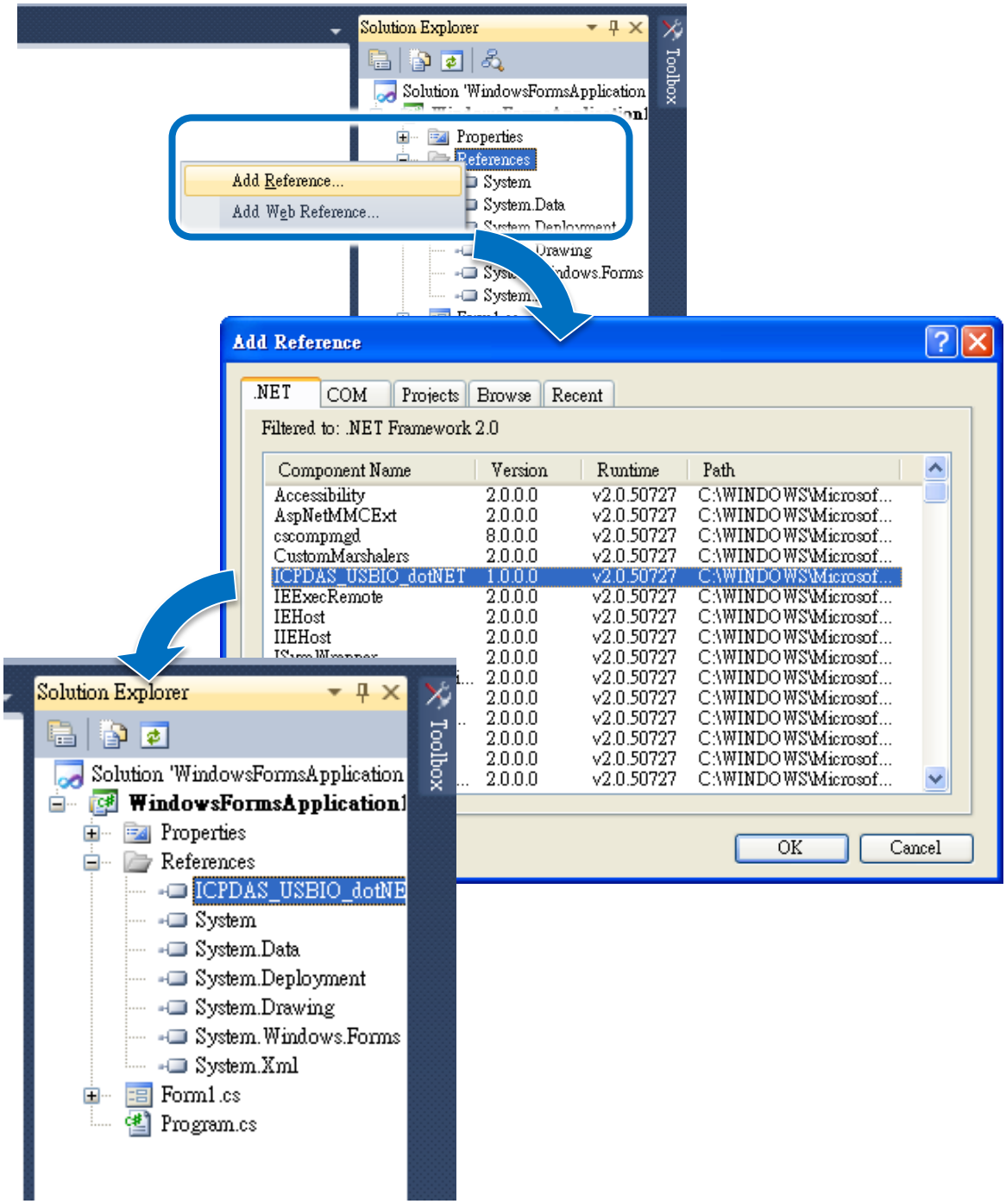


4.2.1.1. .NET (C#, VB.NET)

This section explains how to integrate the SDK into .NET applications such as C# and VB.NET, using provided DLLs as Visual Studio references.

- **Files:** Dlls\Dotnet_DLL_x64 or Dlls\Dotnet_DLL_x86

- **Integration:** Add ICPDAS_USBIO_dotNET.dll as a reference in Visual Studio.

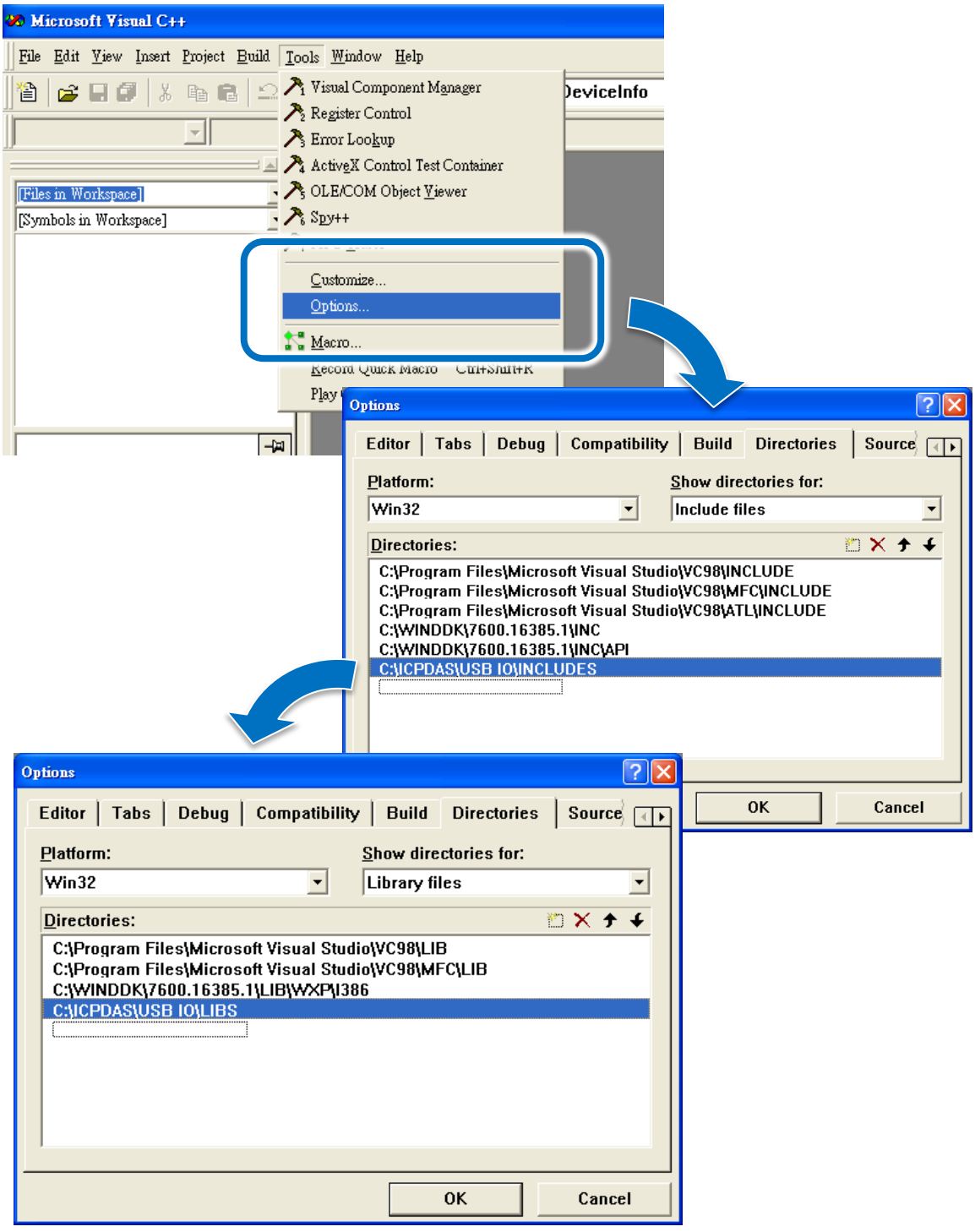


4.2.1.2. Visual C++ (VC)

This section describes how to configure the SDK for Visual C++ development. It requires setting up include paths, library references, and deploying DLLs to the execution directory.

- **Files:** Dlls\VC_DLL_x64 or Dlls\VC_DLL_x86, includes, Libs

- **Integration:** Configure Include and Library paths; deploy .dll in execution directory.

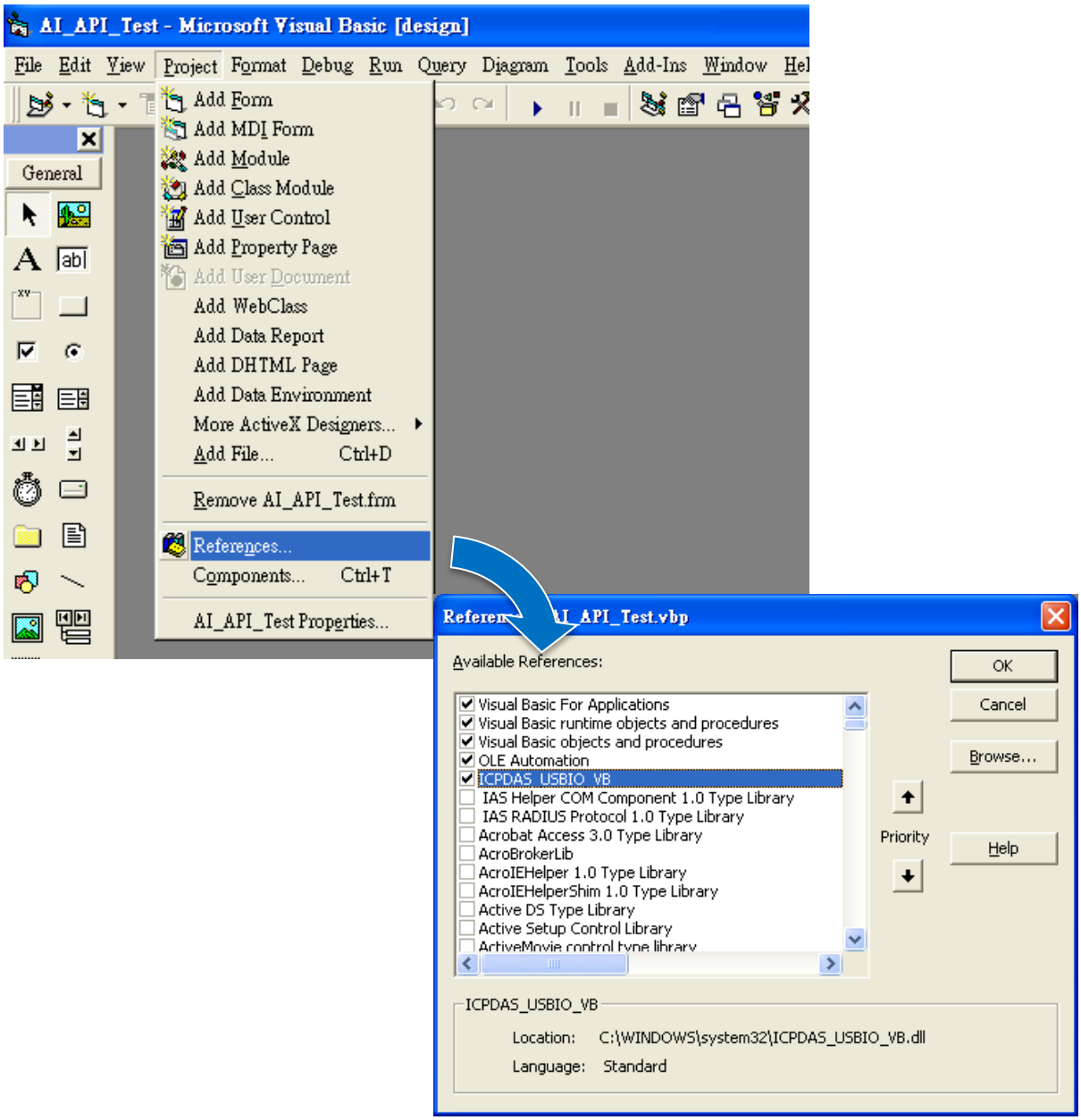


4.2.1.3. Visual Basic (VB6)

This section explains how to use the SDK in Visual Basic projects. The integration method is similar to .NET, where DLLs can be referenced directly, or APIs can be declared manually.

- **Files:** Dlls\Dotnet_DLL_x64 or Dlls\Dotnet_DLL_x86

- **Integration:** Add DLL references to the project, or declare API functions via `Declare` statements.

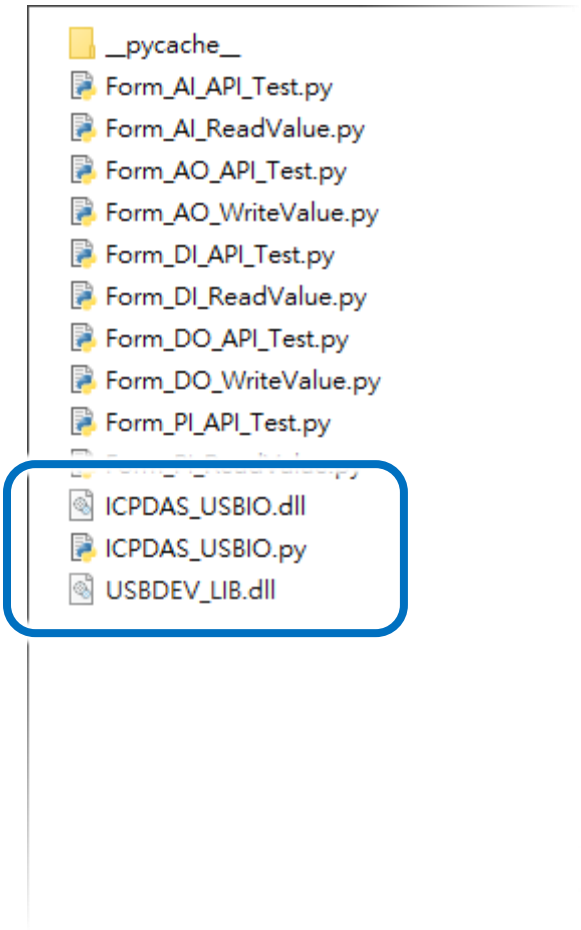


4.2.1.4. Python

This section introduces how to develop Python applications with the SDK. The package provides a Python wrapper and required DLLs, along with demo codes.

- **Files:** Samples\python_x64(or x86)\ ICPDAS_USBIO.dll, Samples\python_x64(or x86)\ ICPDAS_USBIO.py, Samples\python_x64(or x86) USBDEV_LIB.dll

- **Integration:** Import ICPDAS_USBIO.py, ensure ICPDAS_USBIO.dll and USBDEV_LIB.dll are accessible.



4.2.2. Linux Development Environment

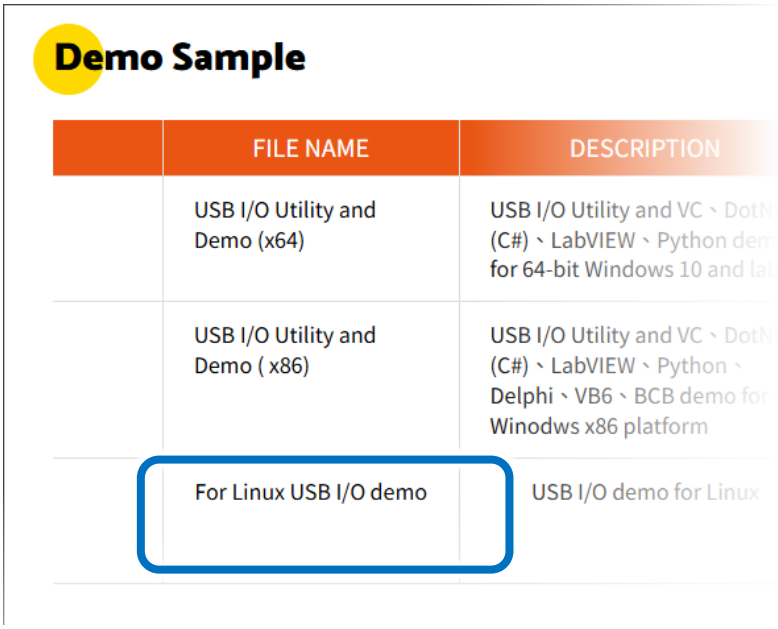
A dedicated software package is provided for Linux platforms, containing all required documentation, libraries, header files, and sample programs. It supports GCC-based compilation with Makefiles.

Download Location:

The latest Windows software package can be downloaded from the ICP DAS official website:
https://www.icpdas.com/en/download/index.php?model=USB-2255%20series

File name

- **For Linux USBIO demo**



Package Contents

usbio/

```
├── doc          → Documentation
├── examples     → Sample codes
├── include      → Header files
├── lib          → Libraries
├── ChangeLog    → Version history
├── Makefile     → Compilation configuration
└── README       → Quick start guide
```

Integration Notes

- Linux kernel 2.6 or later version
- GNU Make version 3.77 or later version
- GNU gcc version egcs-2.91.66 or later version
- NCURSES 5.0 (for i7kon only) or later version

4.2.3. Sample Programs

This section introduces the sample programs included in the software package. These examples demonstrate common operations such as digital input/output, analog data handling, and pulse signal management. Developers can use them as references to accelerate learning and reduce development time.

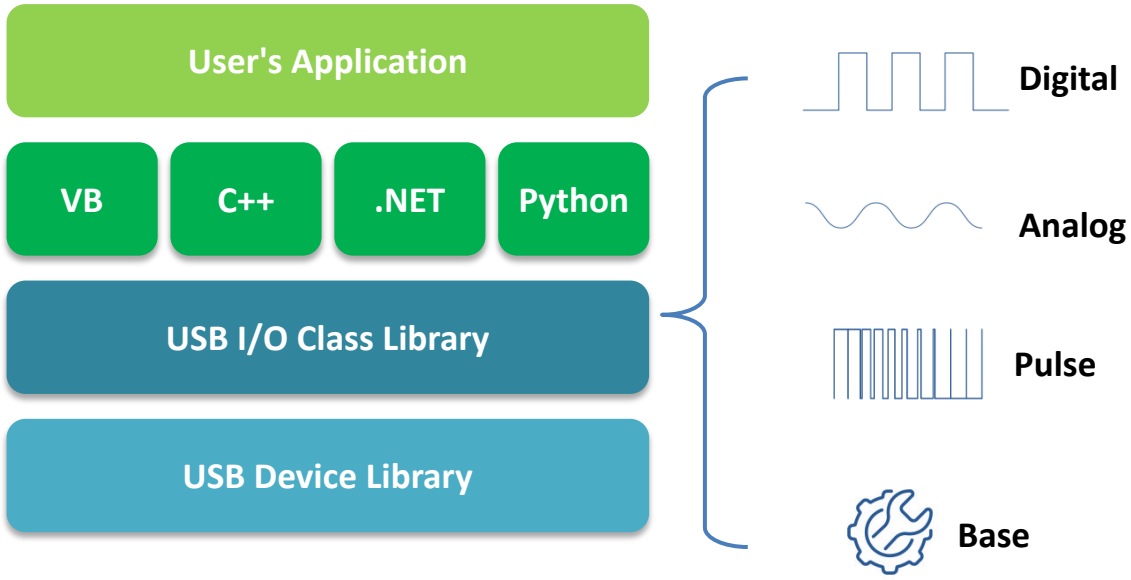
Sample Locations

- **Samples\USB_DotNetDemo_x64:** .NET sample projects
- **Samples\VC:** Visual C++ sample codes
- **Samples\python_x64:** Python sample codes
- **Samples\USBIO_LabVIEW_demo_x64:** LabVIEW demo project
- **USBIO_Qt_demo_x64.zip:** Qt demo project

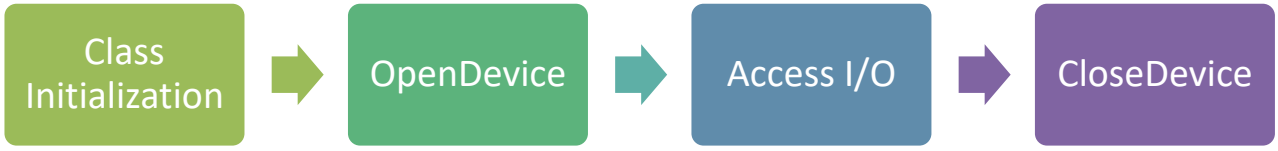
4.3. Software Structure

This section explains the layered software architecture of the USB-2255, which separates device drivers, I/O class libraries, and user applications. By understanding this structure, developers can simplify integration and improve code maintainability.

The USB-2255-32/USB-2255-64 adopts layered software architecture, comprising the "USB Device Driver Layer", "USB I/O Class Library Layer", and "User Application Layer". This modular design allows developers to easily integrate and operate I/O functions (DI/DO, AI/AO, PI/PO) using programming languages such as Visual Basic, C++, and .NET. The architecture enhances code readability and maintainability while reducing integration barriers across platforms.



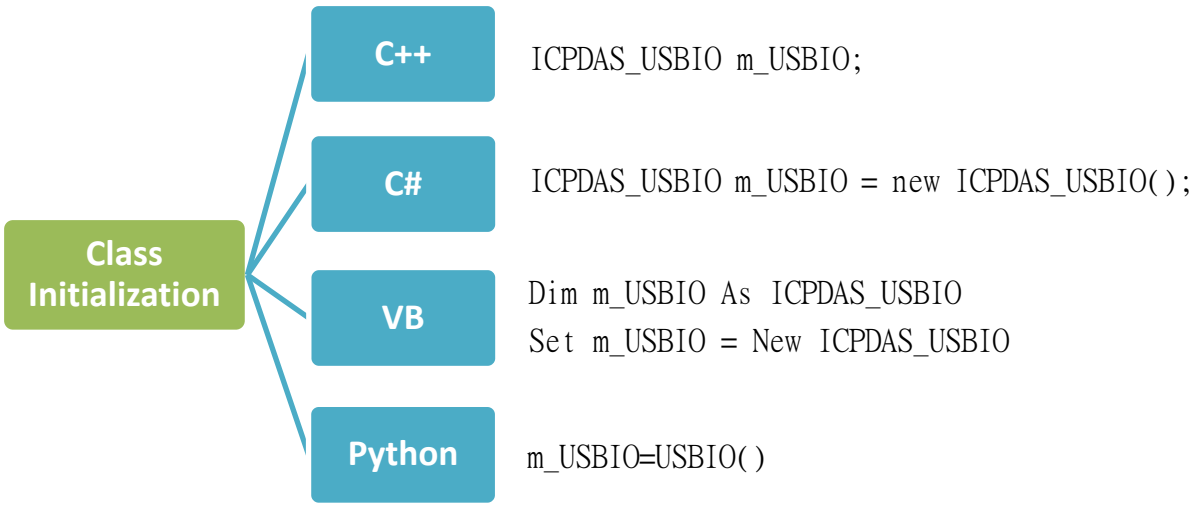
To help developers understand how to use the USB I/O class library, the following diagram presents the complete software flow—from object creation, device connection, I/O access, to device closure and resource release.



The software operation flow is structured into four key stages: (1) Class Initialization, (2) Open Device, (3) I/O Access, and (4) Close Device. Each stage corresponds to specific library methods and syntax examples. Detailed descriptions and implementation guidance are provided in the following subsections.

4.3.1. Class Initialization

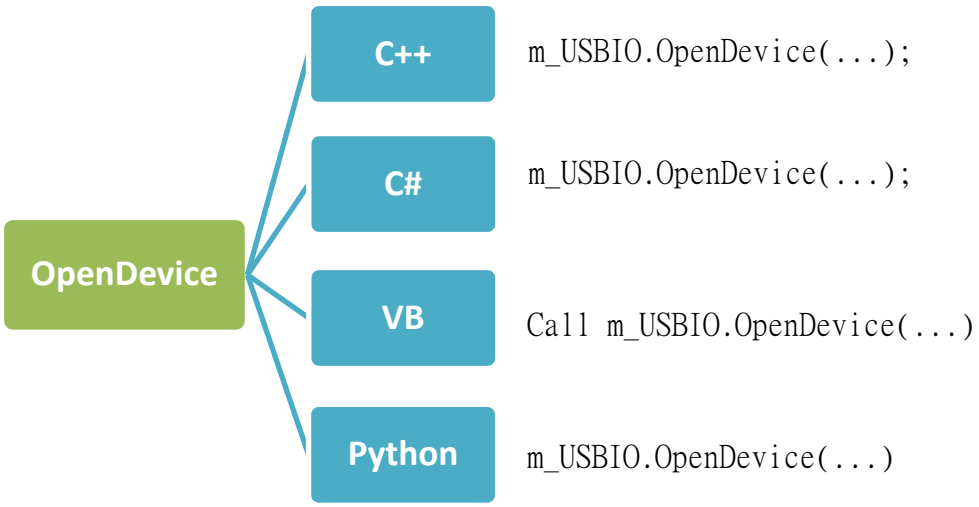
Before performing I/O operations, users must instantiate the USB I/O class and complete the initialization to establish the control interface.



Programming language	Syntax
C++	ICPDAS_USBIO m_USBIO;
C#	ICPDAS_USBIO m_USBIO = new ICPDAS_USBIO();
VB	Dim m_USBIO As ICPDAS_USBIO Set m_USBIO = New ICPDAS_USBIO
Pyhton	m_USBIO=USBIO()

4.3.2. Open Device

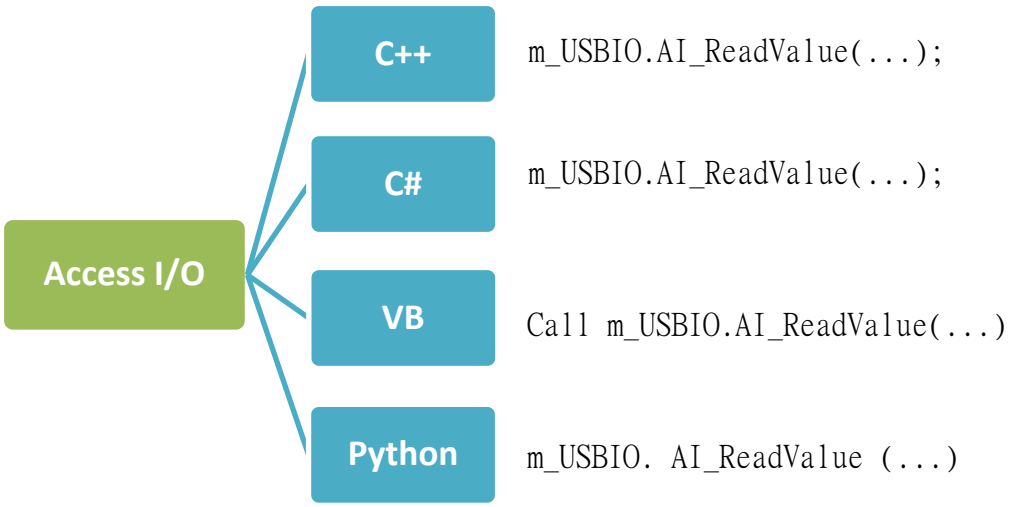
After initialization, use `OpenDevice()` to connect to the device.



Programming language	Syntax
C++	m_USBIO.OpenDevice(...);
C#	m_USBIO.OpenDevice(...);
VB	Call m_USBIO.OpenDevice(...)
Pyhton	m_USBIO.OpenDevice(...)

4.3.3. Access I/O

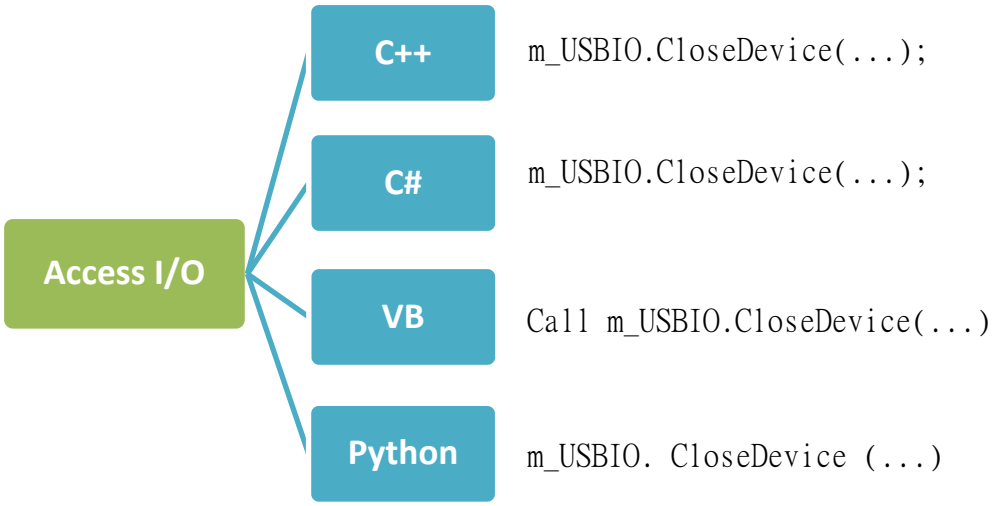
Once the connection is established, I/O access methods can be used to read or write various I/O channels such as DI, DO, AI and AO.



Programming language	Syntax
C++	m_USBIO.AI_ReadValue(...);
C#	m_USBIO.AI_ReadValue(...);
VB	Call m_USBIO.AI_ReadValue(...)
Pyhton	m_USBIO. AI_ReadValue (...)

4.3.4. Close Device

After the application is finished, use `CloseDevice()` to release system resources.

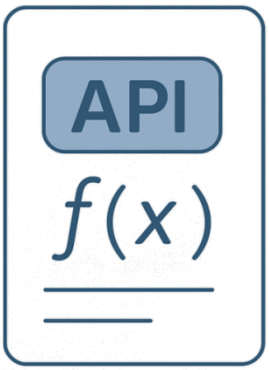


Programming language	Syntax
C++	m_USBIO.CloseDevice(...);
C#	m_USBIO.CloseDevice(...);
VB	Call m_USBIO.CloseDevice(...)
Pyhton	m_USBIO. CloseDevice (...)

5. API Reference

This chapter introduces the class members of ICPDAS_USBIO, the main interface for controlling USB-2200 series modules. It includes constructors, static methods, and public methods for device initialization, query, control, and data access.

The members of the ICPDAS_USBIO class are divided into constructors, static and public methods. The constructors initialize and create the instance of the ICPDAS_USBIO. Static methods are the ways to identify or scan what USB modules are connected. Public methods are used to access USB modules.



Constructors Methods

Name	Description
ICPDAS_USBIO	Initializes a new instance of the ICPDAS_USBIO class.

Static Methods

Name	Description
ListDevice	List all devices connected with local PC.
ScanDevice	Scan devices connected with local PC.

Public Methods

The public methods are categorized by function, covering system control, device management, digital/analog I/O operations, and pulse input handling. The detail of the public methods will be described in the following section.

5.1. System

This section describes public methods related to system operations. These functions provide device open/close, communication synchronization, timeout settings, and watchdog control for basic module management.

Name	Description
OpenDevice	List all devices connected with local PC.
CloseDevice	Scan devices connected with local PC.
SYNCDevice	Send a synchronization packet to clear software WDT.
SetCommTimeout	Set communication timeout.
GetCommTimeout	Get communication timeout.
SetAutoResetWDT	Enable / disable automatic reset of the WDT.

5.1.1. OpenDevice

Open USBIO with device ID and board ID.
The device ID is defined by the header ICPDAS_USBIO.h or the enumeration in ICPDAS_USBIO.

Syntax

```
public int OpenDevice (
    WORDi_wUSBIO_DID,
    BYTEi_byUSBIO_BID
)
```

Parameter

i_wUSBIO_DID

[IN] Device ID for the specific device to open (Defined in ICPDAS_USBIO.h).

i_byUSBIO_BID

[IN] Board ID for the specific device to open.

Return Value

Error code.

Example

```
Int iErrCode;
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
iErrCode = m_usbIO.OpenDevice(USB22XX, 1);
iErrCode = m_usbIO.CloseDevice();
```

5.1.2. CloseDevice

Close device and release resource.

Syntax

```
public int CloseDevice (
  void
)
```

Parameter

None.

Return Value

Error code.

Example

```
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  // Some code accessing USB I/O
  iErrCode = m_usbIO.CloseDevice();
}
```

5.1.3. SYNCDevice

Refreshe the module's host watchdog.
After enabling the WDT, you must use "SYNCDevice" to activate it; otherwise, the WDT will not start counting.

Syntax

```
public int SYNCDevice (
  void
)
```

Parameter

None.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
if(ERR_NO_ERR != (iErrCode = m_usbIO.SetSoftWDTTimeout(1000)))
    printf("%d", iErrCode);
If(ERR_NO_ERR != (iErrCode = m_usbIO.SetAutoResetWDT(TRUE)))
    printf("%d", iErrCode);
If(ERR_NO_ERR != (iErrCode = m_usbIO.SYNCDevice()))
    printf("%d", iErrCode);
iErrCode = m_usbIO.CloseDevice();
```

```
}
```

5.1.4. SetCommTimeout

Set the communication timeout between packet send and receive.
Note 1: The timeout value will affect communication. If the timeout is small, it means the communication is timeout after the value passed.
Note 2: The default value when first initial an ICP DAS USB I/O is 100ms.

Syntax

```
public int SetCommTimeout (
  DWORDi_dwCommTimeout
)
```

Parameter

i_dCommTimeout

[IN] The communication timeout in millisecond (ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR != (iErrCode = m_usbIO.SetCommTimeout(1000)))
printf(“%d”, iErrCode)
```

5.1.5. GetCommTimeout

Get the communication timeout between packet send and receive.
Note 1: The timeout value will affect communication. If the timeout is small, it means the communication is timeout after the value passed.
Note 2: The default value when first initial an ICP DAS USB I/O is 100ms.

Syntax

```
public int GetCommTimeout (
  DWORD*o_dwCommTimeout
)
```

Parameter

o_dCommTimeout

[OUT] The communication timeout in millisecond (ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwCommTimeout;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.SetCommTimeout(1000)))
if(ERR_NO_ERR != (iErrCode = m_usbIO.GetCommTimeout(&o_dwCommTimeout)))
printf("%d", iErrCode);
else
printf("%d\n", o_dwCommTimeout);
```

5.1.6. SetAutoResetWDT

Enable / disable the handle of watchdog by library.

Syntax

```
public int SetAutoResetWDT (
  BOOLi_bEnable
)
```

Parameter

i_bEnable

[IN] To enable / disable the library to automatically handle watchdog.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  If(ERR_NO_ERR != (iErrCode = m_usbIO.SetAutoResetWDT(FALSE)))
  printf("%d", iErrCode);
}
```

5.2. Device

This section covers methods related to device information access. These functions enable retrieval of device ID, firmware version, serial number, nickname, I/O capability, and channel counts, as well as certain configuration options.

Name	Description
RefreshDeviceInfo	Refresh device information.
GetSoftWDTTimeout	Get software WDT timeout.
GetDeviceID	Get ID of the device.
GetFwVer	Get firmware version of the device.
GetFwDate	Get firmware Date of the device.
GetDeviceNickName	Get nick name of the device.
GetDeviceSN	Get serial number of the device.
GetSupportIOMask	Get the mask of this device IO distribution.
GetDITotal	Get DI total channel of the device.
GetDOTotal	Get DO total channel of the device.
GetAITotal	Get AI total channel of the device.
GetAOTotal	Get AO total channel of the device.
GetPITotal	Get PI total channel of the device.
GetPOTotal	Get PO total channel of the device.
ReadSector_BulkValue	Read sector for bulk value
GetBulkFrameIndex	Get bulk frame index
SetUserDefinedBoardID	Set board ID of this device.
SetDeviceNickName	Set nick name of this device.
SetSoftWDTTimeout	Set software WDT timeout.
LoadDefault	Load default setting.
StopBulk	Stop current bulk process.
RegisterEmergencyPktEventHandle	Register the callback function for emergency event sent from USBIO.
SetOpcode	Set Device Opcode

5.2.1. RefreshDeviceInfo

Refresh all information of this device.
Note 1: The RefreshDeviceInfo() will be called automatically when open device.
Note 2: This function will take time to refresh information.

Syntax

```
public int RefreshDeviceInfo (
  void
)
```

Parameter

None.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != iErrCode = m_usbIO.RefreshDeviceInfo()))
  printf("%d", iErrCode)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.2. GetSoftWDTTimeout

Get the software WDT timeout of I/O module.

Syntax

```
public int GetSoftWDTTimeout (
  DWORD *o_dwSoftWDTTimeout
)
```

Parameter

o_dwSoftWDTTimeout

[OUT] The software WDT timeout in millisecond (ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwSoftWDTTimeout;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != iErrCode = m_usbIO.GetSoftWDTTimeout(&o_dwSoftWDTTimeout)))
  printf("%d",iErrCode);
else
  printf("%d\n", o_dwCommTimeout);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.3. GetDeviceID

Get ID of the device.

Syntax

```
public int GetDeviceID (
  DWORD *o_dwDeviceID
)
```

Parameter

`o_dwDeviceID`

[OUT] The device ID.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwDeviceID;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetDeviceID(&o_dwDeviceID)))
  printf("%d", iErrCode);
else
  printf("%d", o_dwDeviceID);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.4. GetFwVer

Get ID of the device.

Syntax

```
public int GetDeviceID (
  WORD *o_ wFwVer
)
```

Parameter

o_ wFwVer

[OUT] The firmware version.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD o_wFwVer;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetFwVer(&o_wFwVer)))
  printf("%d", iErrCode);
else
  printf("%d",o_wDwVer);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.5. GetFwDate

Get firmware Date of the device.

Syntax

```
public int GetFwDate (
  BYTE* o_byFwDate
)
```

Parameter

`o_byFwDate`

[OUT] The byte array of firmware Date.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE o_byFwDate;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. GetFwDate (&o_byFwDate)))
  printf("%d", iErrCode);
else
  for(int i = 0; i<10;i++)
  printf("%02X", o_byFwDate));
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.6. GetDeviceNickName

Get nick name of the device.

Syntax

```
public int GetDeviceNickName (
  BYTE *o_byDeviceNickName
)
```

Parameter

`o_byDeviceNickName`

[OUT] The byte array of the nick name of the device.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byDeviceNickName[USBIO_NICKNAME_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetDeviceNickName(o_byDeviceNickName)))
  printf("%d", iErrCode);
else
  printf("%s", o_byDeviceNickName);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.7. GetDeviceSN

Get serial number of the device.

Syntax

```
public int GetDeviceSN (
  BYTE *o_byDeviceSN
)
```

Parameter

`o_byDeviceSN`

[OUT] The byte array of the serial number of the device.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byDeviceSN[USBIO_SN_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetDeviceSN(o_ byDeviceSN)))
  printf("%d", iErrCode);
else
  printf("%s", o_ byDeviceSN);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.8. GetSupportIOMask

Get the mask of this device IO distribution.
Each bit of the mask indicates each supported IO type as shown in the following table.

Bit7	Bit6	Bit5	Bit4	Bit3	Bit2	Bit1	Bit0
N/A	N/A	PI	PO	AI	AO	DI	DO

Syntax

```
public int GetSupportIOMask (
  BYTE *o_bySupportIOMask
)
```

Parameter

`o_bySupportIOMask`

[OUT] The support IO mask of the device.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_bySupportIOMask;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetSupportIOMask(&o_bySupportIOMask)))
  printf("%d", iErrCode);
else
  printf("0x%02x",o_bySupportIOMask);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.9. GetDITotal

Get DI total number of channels of the device.

Syntax

```
public int GetDITotal (
  BYTE *o_byDITotal
)
```

Parameter

`o_byDITotal`

[OUT] The DI total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byDITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetDITotal(&o_byDITotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byDITotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.10. GetDOTotal

Get DO total number of channels of the device.

Syntax

```
public int GetDOTotal (
  BYTE *o_byDOTotal
)
```

Parameter

`o_byDOTotal`

[OUT] The DO total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byDOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetDOTotal(&o_byDOTotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byDOTotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.11. GetAITotal

Get AI total number of channels of the device.

Syntax

```
public int GetAITotal (
  BYTE *o_byAITotal
)
```

Parameter

`o_byAITotal`

[OUT] The AI total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byAITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetAITotal(&o_byAITotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byAITotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.12. GetAOTotal

Get AO total number of channels of the device.

Syntax

```
public int GetAOTotal (
  BYTE *o_byAOTotal
)
```

Parameter

`o_byAOTotal`

[OUT] The AO total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byAOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetAOTotal(&o_byAOTotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byAOTotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.13. GetPITotal

Get PI total number of channels of the device.

Syntax

```
public int GetPITotal (
  BYTE *o_byPITotal
)
```

Parameter

`o_byPITotal`

[OUT] The PI total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byPITotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetPITotal(&o_byPITotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byPITotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.14. GetPOTotal

Get PO total number of channels of the device.

Syntax

```
public int GetPOTotal (
  BYTE *o_byPOTotal
)
```

Parameter

`o_byPOTotal`

[OUT] The PO total number of channels.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byPOTotal;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.GetPOTotal(&o_byPOTotal)))
  printf("%d", iErrCode);
else
  printf("%d",o_byPOTotal);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.15. ReadSector_BulkValue

Read sector for bulk value.

Syntax

```
public DWORD ReadSector_BulkValue (
  BYTE b_Opcode,
  DWORD i_dwSectorCnt,
  BYTE* o_SectorBuf,
  OnReadSectorBulkEvent i_XBCallBack
)
```

Parameter

b_Opcode

[IN] set value for device OPCode

i_dwSectorCnt

[IN] The index to sector

o_SectorBuf

[OUT] The bulk value of AI or DI

i_XBCallBack

[IN] The bulk function for read sector. The address of pointer to the user-defined callback function to handle the above event type.

Return Value

Error code.

Example

```
void i_XBCallBack(WORD i_wUSBIO_DID, BYTE i_byUSBIO_BID, BYTE i_OpCode, DWORD
dwCallBackIndex ,DWORD dwBuffLength, BYTE *BulkBuf, double spendTime)
{
  //read sector value back
}
void main()
{
  int iErrCode
  ICPDAS_USBIO m_usbIO;
  BYTE b_Opcode;
  DWORD i_dwSectorCnt;
  BYTE* o_SectorBuf;

  m_usbIO = new ICPDAS_USBIO();
  if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
  {
    if(ERR_NO_ERR != (iErrCode = m_usbIO. ReadSector_BulkValue (b_Opcode,
    i_dwSectorCnt, o_SectorBuf, i_XBCallBack)))
    {
      printf("%d", iErrCode);
    }
    iErrCode = m_usbIO.CloseDevice();
  }
}
```

5.2.16. GetBulkFrameIndex

Get bulk frame index.

Syntax

```
public DWORD GetBulkFrameIndex (
  void
)
```

Parameter

None.

Return Value

Number of Frame index.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD FrameIndex;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  FrameIndex = m_usbIO. GetBulkFrameIndex();
  printf("%d", FrameIndex);
}
```

5.2.17. SetUserDefinedBoardID

Set board ID of this device. The valid value of the ID is from 16 to 127.

Syntax

```
public int SetUserDefinedBoardID (
  BYTEi_byBID
)
```

Parameter

`i_byBID`

[IN] The board ID to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.SetUserDefinedBoardID(123)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.18. SetDeviceNickName

Set nick name of this device. The maximum number of the character of this device is 32.

Syntax

```
public int SetDeviceNickName (
  BYTE *i_byDeviceNickName
)
```

Parameter

`*i_byDeviceNickName`

[OUT] The byte array of the nick name to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte byNickName[USBIO_NICKNAME_LENGTH];

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  sprintf(byNickName, “Station 1-1-3”);
  if(ERR_NO_ERR != (iErrCode = m_usbIO.SetDeviceNickName(byNickName)))
  printf(“%d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.19. SetSoftWDTTimeout

Set the software WDT timeout.
The minimum value of timeout is 100ms, and maximum is 30 minutes.

Syntax

```
public int SetSoftWDTTimeout (
  DWORD i_dwSoftWDTTimeout
)
```

Parameter

`i_dwSoftWDTTimeout`

[IN] The software WDT timeout in millisecond(ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.SetSoftWDTTimeout(1000)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.20. LoadDefault

Load default setting.

Syntax

```
public int LoadDefault (
  void
)
```

Parameter

None.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  If(ERR_NO_ERR != (iErrCode = m_usbIO.LoadDefault()))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.21. StopBulk

Stop current bulk process.

Syntax

```
public int LoadDefault (
  Void
)
```

Parameter

None.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  If(ERR_NO_ERR != (iErrCode = m_usbIO.StopBulk()))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.22. RegisterEmergencyPktEventHandle

Register the callback function for emergency event sent from USBIO.
When in callback operation, it will cause the performance in your callback function. Please reduce execute time in this callback function.

Syntax

```
public int RegisterEmergencyPktEventHandle (
  OnEmergencyPktArriveEvent i_evtHandle
)
```

Parameter

`i_evtHandle`

[IN] The callback function for emergency event.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Bool m_bEmcyPktArrive;
Byte byEmcyPkt[USBIO_MAX_PACKET_LENGTH];

Void emcypkeEvt(Byte* byData, Byte byLen)
{
  m_ bEmcyPktArrive = true;
  memcpy(byEmcyPkt, byData, byLen);
}

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22xx, 1)))
{
  If(ERR_NO_ERR != (iErrCode =
  m_usbIO.RegisterEmergencyPktEventHandle(emcypkeEvt)))
  printf("%d", iErrCode);
  while(1)
  {
    // User's application loop
    If(m_ bEmcyPktArrive)
    {
      // Handle emcy packet
    }
  }
  iErrCode = m_usbIO.CloseDevice();
}
```

5.2.23. SetOpcode

Set Device Opcode.
This API can be set to multiple modes for using mass storage with the USB22xx and XB series.
OP0: Uses the polling method.
OP1 or OP2: Uses the callback operation mode for AI or DI.

Syntax

```
public int SetOpcode (
  BYTE code
)
```

Parameter

`code`

[IN] Set poling mode or bulk mode.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE code;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. SetOpcode (code))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.3. Digital Input

This section introduces methods for Digital Input functionality. It includes digital value reading, filter configuration, input inversion, edge triggering, and counter operations for typical DI data acquisition.

Name	Description
DI_GetDigitalFilterWidth	Get DI Digital Filter Width
DI_GetDigitalValueInverse	Get DI Value Inverse
DI_GetCntEdgeTrigger	Get DI Counter Edge Trigger
DI_ReadValue	Read DI Value
DI_ReadCounterValue	Read DI Counter Value
DI_SetDigitalFilterWidth	Set DI Digital Filter Width
DI_SetDigitalValueInverse	Set DI Value Inverse
DI_SetCntEdgeTrigger	Set DI Counter Edge Trigger
DI_WriteClearCounter	Clear Specified Channel of DI Counter Value
DI_WriteClearCounters	Clear DI Counter Value with Clear Mask

5.3.1. DI_GetDigitalFilterWidth

Digital input function - Get DI Digital Filter Width. The digital filter width is used for filtering the noise or the glitch. The unit of the filter is 0.1 million-second.

Syntax

```
public int DI_GetDigitalFilterWidth (
  WORD *o_wFilterWidth
)
```

Parameter

*o_wFilterWidth

[OUT] The digital filter width (The unit is 0.1 ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORDo_wFilterWidth;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.DI_GetDigitalFilterWidth(&o_wFilterWidth)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_wFilterWidth);
  }
}
```

5.3.2. DI_GetDigitalValueInverse

Get DI Value Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DI_GetDigitalValueInverse (
  DWORD * o_dwInverse
)
```

Parameter

* o_dwInverse

[OUT] The inverse setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORDo_dwInverse;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_GetDigitalValueInverse(&o_dwInverse)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_dwInverse);
  }
}
```

5.3.3. DI_GetCntEdgeTrigger

Get DI Counter Edge Trigger. This edge trigger is used for the counting operation of the counter.

Edge Trigger	Value
Falling	0
Rising	1

Syntax

```
public int DI_GetCntEdgeTrigger (
  DWORD * o_dwEdgeTrig
)
```

Parameter

* o_dwEdgeTrig

[OUT] The counter edge trigger setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORDo_dwEdgeTrig;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_GetCntEdgeTrigger(&o_dwEdgeTrig)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_dwEdgeTrig);
  }
}
```

5.3.4. DI_ReadValue

Read DI Value. The values of digital input channels in byte array format.

Syntax

```
public int DI_ReadValue (
  BYTE *o_byDIValue
)
```

Parameter

`*o_byDIValue`

[OUT] The byte arrays of the DI value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTEo_byDIValue[USBIO_DI_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_ReadValue (o_byDIValue)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx < (USBIO_DI_MAX_CHANNEL>> 3); iIdx++)
    printf("%d\n", o_byDIValue[iIdx]);
  }
}
```

5.3.5. DI_ReadCounterValue

Read DI Counter Value. The counting value of the digital input counter in word array format.

Syntax

```
public int DI_ReadCounterValue (
  DWORD *o_ dwDICntValue
)
```

Parameter

o_ dwDICntValue

[OUT] The double-word arrays of the DI counter value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORDo_dwDICntValue[USBIO_DI_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_ReadCounterValue(o_dwDICntValue)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx < (USBIO_DI_MAX_CHANNEL>> 3); iIdx++)
    printf("%d\n",o_dwDICntValue[iIdx]);
  }
}
```

5.3.6. DI_SetDigitalFilterWidth

Set DI Digital Filter Width. Used for setting the filter width for the digital input. The unit of the filter width is 0.1 million-second..

Syntax

```
public int SetDigitalFilterWidth (
  WORD *i_ wFilterWidth
)
```

Parameter

*i_ wFilterWidth

[IN] The digital filter width (The unit is 0.1 ms).

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetDigitalFilterWidth (10))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.3.7. DI_SetDigitalValueInverse

Set DI Value Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DI_SetDigitalValueInverse(
  DWORD i_dwInverse
)
```

Parameter

i_dwInverse

[IN] The inverse setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetDigitalValueInverse (1))
  printf(“%d”, iErrCode);
  m_usbIO.CloseDevice();
}
```

5.3.8. DI_SetCntEdgeTrigger

Set DI Counter Edge Trigger.

Edge Trigger	Value
Falling	0
Rising	1

Syntax

```
public int DI_SetCntEdgeTrigger (
  DWORD *i_dwEdgeTrig
)
```

Parameter

*i_dwEdgeTrig

[IN] The counter edge trigger setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_SetCntEdgeTrigger (1))
  printf(“%d”, iErrCode);
  m_usbIO.CloseDevice();
}
```

5.3.9. DI_WriteClearCounter

Clear specified channel of DI Counter Value.

Syntax

```
public int DI_WriteClearCounter(
  BYTE *i_byChToClr
)
```

Parameter

*i_byChToClr

[IN] The counter channel for clearing.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_WriteClearCounter (1))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.3.10. DI_WriteClearCounters

Clear DI Counter Value with Clear Mask.

Syntax

```
public int DI_WriteClearCounters(
  DWORD *i_dwCntClrMask
)
```

Parameter

*i_dwCntClrMask

[IN]The counter clear mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DI_WriteClearCounters(0x0000FFFF))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.4. Digital Output

This section describes methods for Digital Output functionality. It includes digital output read/write, power-on settings, safety value configuration, and inversion control, enabling digital control and protection logic.

Name	Description
DO_GetPowerOnEnable	Get Power-On Enable
DO_GetSafetyEnable	Get Safety Enable
DO_GetSafetyValue	Get Safety Value
DO_GetDigitalOutputInverse	Get DO Output Inverse
DO_ReadValue	Read DO Value
DO_SetPowerOnEnable	Set Power-On Enable
DO_SetSafetyEnable	Set Safety Enable
DO_SetSafetyValue	Set Safety Value
DO_SetDigitalOutputInverse	Set DO Output Inverse
DO_WriteValue	Write DO Value

5.4.1. DO_GetPowerOnEnable

Get Power-On Enable.

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_GetPowerOnEnable(
  BYTE *o_byPowerOnEnable
)
```

Parameter

*o_byPowerOnEnable

OUT] The power-on enable mask. Each byte represents the power-on enable/disable configuration of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byPowerOnEnable[USBIO_DO_MAX_CHANNEL];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetPowerOnEnable(&o_byPowerOnEnable)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx <USBIO_DO_MAX_CHANNEL; iIdx++)
    printf("%02x\n", o_byPowerOnEnable[iIdx]);
  }
}
```

5.4.2. DO_GetSafetyEnable

Get Safety Enable. Each channel takes one bit.

Safety Enable	Value
Disable	0
Enable	1

Syntax

```
public int DO_GetSafetyEnable (
  DWORD *o_bySafetyEnables
)
```

Parameter

*o_ bySafetyEnables

[OUT] The safety enable mask. Each bit of the mask represents the safety enable/disable configuration of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_bySafetyEnable[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetSafetyEnable(&o_bySafetyEnables))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
    printf("%02x\n", o_bySafetyEnables[iIdx]);
  }
}
```

5.4.3. DO_GetSafetyValue

Get Safety Value. Each channel takes one bit.

Safety Value	Value
Off	0
On	1

Syntax

```
public int DO_GetSafetyValue (
  BYTE* o_bySafetyValue
)
```

Parameter

*o_bySafetyValue

[OUT] The safety value. Each bit represents the safety value of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_bySafetyValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetSafetyValue(&o_bySafetyValue))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
    printf("%02x\n", o_bySafetyValue[iIdx]);
  }
}
```

5.4.4. DO_GetDigitalOutputInverse

Get DO Output Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DO_ GetDigitalOutputInverse (
  DWORD *o_dwInverse
)
```

Parameter

*o_dwInverse

[OUT] The inverse setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORDo_dwInverse;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_GetDigitalOutputInverse(&o_dwInverse)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_dwInverse);
  }
}
```

5.4.5. DO_ReadValue

Read DO Value.

Syntax

```
public int DO_ReadValue(
  BYTE *o_byDOValue
)
```

Parameter

*o_byDOValue

[OUT] The DO value. Each bit represents the DO value of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byDOValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_ReadValue(&o_byDOValue))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    for(iIdx = 0; iIdx <((USBIO_DO_MAX_CHANNEL + 7] / 8); iIdx++)
    printf("%02x\n", o_byDOValue[iIdx]);
  }
}
```

5.4.6. DO_SetPowerOnEnable

The class has two overload methods for configuring power on functionality. One provides specifying channel to set, another for all channel. These two overload methods are listed as following table and described in following section.

Name of Methods
DO_SetPowerOnEnable(BYTE i_byChToSet, BYTE i_byPowerOnEnable)
DO_SetPowerOnEnable(BYTE* i_byPowerOnEnables)

5.4.6.1. DO_SetPowerOnEnable(BYTE i_byChToSet, BYTE i_byPowerOnEnable)

Set Power-On enable for specific channel. The value of the enable byte is listed below.

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_SetPowerOnEnable (
  BYTE i_byChToSet,
  BYTE i_byPowerOnEnable
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_byTypeCode

[IN] The power-on enable for the specific channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetPowerOnEnable(0, 0x1)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.4.6.2. DO_SetPowerOnEnable(BYTE* i_byPowerOnEnables)

Set Power-On enable for all channel. The value of the enable byte is listed below.

Power-On Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int DO_SetPowerOnEnable (
  BYTE *i_byPowerOnEnables
)
```

Parameter

*i_byPowerOnEnables

[IN] The byte array of the power-on enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte i_byChPwrOn[USBIO_DO_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_DO_MAX_CHANNEL; iIdx)
  i_byChPwrOn[iIdx] = 0x1;
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetPowerOnEnable(i_byChPwrOn)))
  printf(“%d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.4.7. DO_SetSafetyEnable

Set Safety Enable. Each bit represents channel safety enable. The value of the safety enable is shown below..

Safety Enable	Value
Disable	0
Enable	1

Syntax

```
public int DO_SetSafetyEnable(
  BYTE *i_bySafetyEnable
)
```

Parameter

*i_bySafetyEnable

[IN] The safety enable mask. Each bit represents the safety configuration of each channel..

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Bytei_bySafetyEnable[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  memset(i_bySafetyEnable, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetSafetyEnable(i_bySafetyEnable))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.4.8. DO_SetSafetyValue

Set Safety Value. Each bit represents safety value. The value of the safety is shown below.

Safety Value	Value
Off	0
On	1

Syntax

```
public int DO_ SetSafetyValue (
  BYTE *i_bySafetyValue
)
```

Parameter

*i_bySafetyValue

[IN] The safety value. Each bit represents the safety value of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Bytei_bySafetyValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  memset(i_bySafetyValue, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetSafetyValue(i_bySafetyValue))
  printf(“%d”, iErrCode);
  m_usbIO.CloseDevice();
}
```

5.4.9. DO_SetDigitalOutputInverse

Set DO Output Inverse.

Inverse	Value
No	0
Yes	1

Syntax

```
public int DO_SetDigitalOutputInverse (
  DWORD i_dwInverse
)
```

Parameter

i_dwInverse

[OUT] The inverse setting.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_SetDigitalOutputInverse (1))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.4.10. DO_WriteValue

The class has two overload methods for writing DO value. One provides specifying channel to write, another for all channel. These two overload methods are listed as following table and described in following section.

Name of Methods
DO_WriteValue (BYTEi_byChannel, BYTEi_byValue)
DO_WriteValue (BYTE *i_byDOValue)

5.4.10.1. DO_WriteValue (BYTEi_byChannel, BYTEi_byValue)

Write DO Value to specific channel. The value of the digital output is shown below..

Enable Bit	Value
Off	0
On	1

Syntax

```
public int DO_WriteValue(
  BYTE i_byChannel
  BYTE i_byValue
)
```

Parameter

i_byChannel

[IN] The specific DO channel to be set.

i_byValue

[IN] The DO on / off bit.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_WriteValue(5, 1))
  printf(“%d”, iErrCode);
  m_usbIO.CloseDevice();
}
```

5.4.10.2. DO_WriteValue (BYTE *i_byDOValue)

Write DO Value. Each bit represents channel value. The value of the digital output is shown below.

Enable Bit	Value
Off	0
On	1

Syntax

```
public int DO_WriteValue(
  BYTE *i_byDOValue
)
```

Parameter

*i_byDOValue

[IN] The DO value. Each bit represents the digital output value of each channel..

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Bytei_byDOValue[(USBIO_DO_MAX_CHANNEL + 7] / 8];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  memset(i_byDOValue, 0xFF, (USBIO_DO_MAX_CHANNEL + 7] / 8);
  if(ERR_NO_ERR != (iErrCode = m_usbIO.DO_WriteValue(i_byDOValue))
  printf("%d", iErrCode);
  m_usbIO.CloseDevice();
}
```

5.5. Analog Input

This section outlines methods for Analog Input functionality. It supports channel type configuration, filtering, resolution settings, as well as real-time/bulk value reading and alarm status, suitable for analog signal monitoring and diagnostics.

Name	Description
AI_GetTotalSupportType	Get total supported amount.
AI_GetSupportTypeCode	Get supported type code.
AI_GetTypeCode	Get type code.
AI_GetChCJCOffset	Get channel CJC offset.
AI_GetChEnable	Get channel enable/disable.
AI_GetFilterRejection	Get filter rejection.
AI_GetCJCOffset	Get CJC offset.
AI_GetCJCEnable	Get CJC enable.
AI_GetWireDetectEnable	Get wire detect enable.
AI_GetResolution	Get resolution.
AI_ReadValue	Read AI value in double word format. (Overload)
AI_ReadBulkValue	Read bulk AI value (Fast acquire functionality)
AI_ReadCJCValue	Get CJC value.
AI_GetChSampleRate	Get Sample rate of channel.
AI_ReadLatchValue	Read Latch value.
AI_GetAlarmEnable	Get Alarm Enable.
AI_GetAlarmMode	Get Alarm Mode.
AI_GetAlarmLimit	Get Alarm Limit.
AI_GetAlarmStatus	Get Alarm Status.
AI_SetTypeCode	Set type code for specific channel. (Overload)
AI_SetChCJCOffset	Set channel CJC offset for specific channel. (Overload)
AI_SetChEnable	Set channel enable/disable.
AI_SetFilterRejection	Set filter rejection.

Name	Description
AI_SetCJCOffset	Set CJC offset.
AI_SetCJCEnable	Set CJC enable.
AI_SetWireDetectEnable	Set wire detect enable.
AI_SetChSampleRate	Set Sample rate of channel.
AI_SetAlarmEnable	Set Alarm Enable.
AI_SetAlarmMode	Set Alarm Mode.
AI_SetAlarmLimit	Set Alarm Limit.
AI_ClearLatchValue	Clear Latch Value.
AI_ClearAlarmLatchStatus	Clear Alarm Latch Status.

5.5.1. AI_GetTotalSupportType

Get supported type code Please refer to Appendix A.1 of user's manual to map AI channels input type.

Syntax

```
public int AI_GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetTotalSupportType(&o_
  byTotalSupportType)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AI_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
```

5.5.2. AI_GetSupportTypeCode

Get supported type code Please refer to Appendix A.1 of user's manual to map AI channels input type.

Syntax

```
public int AI_ GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetTotalSupportType(&o_
  byTotalSupportType)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AI_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
```

5.5.3. AI_GetTypeCode

Get type code Please refer to user's manual to map AI channels input type. The type code can reference to Appendix A.1.

Syntax

```
public int AI_GetTypeCode (
  BYTE *o_byTypeCode
)
```

Parameter

*o_byTypeCode

[OUT] The byte array of type code.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTypeCode[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetTypeCode(o_byTypeCode)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("%02x\n", o_byTypeCode[iIdx]);
  }
}
```

5.5.4. AI_GetChCJCOffset

Get channel CJC offset. The valid range of offset is -40.96 ~ +40.95.

Syntax

```
public int AI_GetChCJCOffset (
  float *o_fChCJCOffset
)
```

Parameter

*o_fChCJCOffset

[OUT] The float array of channel CJC offset.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fChCJCOffset[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetChCJCOffset(o_fChCJCOffset)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("%.5f\n", o_fChCJCOffset[iIdx]);
  }
}
```

5.5.5. AI_GetChEnable

Get channel enable/disable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AI_GetChCJCOffset (
  BYTE *o_byChEnable
)
```

Parameter

`*o_byChEnable`

[OUT] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byChEnable[(USBIO_AI_MAX_CHANNEL + 7] / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetChEnable(o_byChEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx <(USBIO_AI_MAX_CHANNEL + 7) / 8; iIdx++)
    printf("%02x\n", o_byChEnable[iIdx]);
  }
}
```

5.5.6. AI_GetFilterRejection

Get filter rejection.

Rejection Setting	Value
60Hz	0
50Hz	1

Syntax

```
public int AI_GetFilterRejection (
  BYTE *o_byFilterRejection
)
```

Parameter

`*o_byFilterRejection`

[OUT] The filter rejection.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byFilterRejection;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AI_GetFilterRejection(&o_byFilterRejection)))
  printf(“%d”, iErrCode);
  else
  printf(“%d\n”, o_byFilterRejection);
}
```

5.5.7. AI_GetCJCOffset

Get CJC offset The valid range of offset is -40.96 ~ +40.95.

Syntax

```
public int AI_GetCJCOffset (
  float *o_fCJCOffset
)
```

Parameter

`*o_fCJCOffset`

[OUT] The CJC offset.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
floato_fCJCOffset;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetCJCOffset(&o_fCJCOffset)))
  printf(“%d”, iErrCode);
  else
  printf(“%.5f\n”, o_fCJCOffset);
}
```

5.5.8. AI_GetCJCEnable

Get CJC enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_GetCJCEnable (
  BYTE *o_byCJCEnable
)
```

Parameter

`*o_byCJCEnable`

[OUT] The CJC enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byCJCEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetCJCEnable(&o_byCJCEnable)))
  printf("%d", iErrCode);
  else
  printf("%d\n", o_byCJCEnable);
}
```

5.5.9. AI_GetWireDetectEnable

Get wire detect enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_GetWireDetectEnable (
  BYTE *o_byWireDetectEnable
)
```

Parameter

`*o_byWireDetectEnable`

[OUT] The wire detect enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byWireDetectEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AI_GetWireDetectEnable(&o_byWireDetectEnable)))
  printf("%d", iErrCode);
  else
  printf("%d\n", o_byWireDetectEnable);
}
```

5.5.10. AI_GetResolution

Get resolution. Each byte indicates each channel real resolution.

Syntax

```
public int AI_GetResolution (
  BYTE *o_byResolution
)
```

Parameter

`*o_byResolution`

[OUT] The byte array of resolution for each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byResolution[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetResolution(o_byResolution)))
  printf("%d", iErrCode);
  else
  {
    For(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("%d\n", o_byResolution[iIdx]);
  }
}
```

5.5.11. AI_ReadValue

The class library provides 4 overload methods to read AI value. Two methods, the parameter in float format, will convert raw value to true inside the method. Others will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AI_ReadValue (DWORD*o_dwAIValue)
AI_ReadValue (DWORD*o_dwAIValue, BYTE* o_byAIChStatus)
AI_ReadValue (float* o_fAIValue)
AI_ReadValue (float* o_fAIValue, BYTE* o_byAIChStatus)

5.5.11.1. AI_ReadValue (DWORD*o_dwAIValue)

Read AI value in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

☑ Channel Over

The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $Y = \left(1 + \frac{0X13E}{0xFFFF}\right) \times \left(10 - (-10)\right) + (-10)$

☑ Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

☑ Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AI value cannot detect the channel status. It only read the AI value but has the most efficiency.

Syntax

```
public int AI_ReadValue (
  DWORD *o_dwAIValue
)
```

Parameter

`*o_dwAIValue`

[OUT] The raw value of AI value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORDo_dwAIValue[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadValue(o_dwAIValue)))
  printf("%d", iErrCode);
  else
  {
    For(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("0x%08x\n", o_dwAIValue[iIdx]);
  }
}
```

5.5.11.2. AI_ReadValue (DWORD*o_dwAIValue, BYTE* o_byAIChStatus)

Read AI value in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

☑ Channel Over

The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $Y = \left(1 + \frac{0X13E}{0xFFFF}\right) \times \left(10 - (-10)\right) + (-10)$

☑ Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

☑ Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AI_ReadValue (
  DWORD *o_dwAIValue
  BYTE* o_byAIChStatus
)
```

Parameter

*o_dwAIValue

[OUT] The raw value of AI value.

*o_byAIChStatus

[OUT] The bytearray of channel status.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwAIValue[USBIO_AI_MAX_CHANNEL];
Byte o_byAIChStatus[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadValue(o_dwAIValue,
  o_byAIChStatus)))
  printf("%d", iErrCode);
  else
  {
    For(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("0x%08x,  0x%02x\n", o_dwAIValue[iIdx], o_byAIChStatus);
  }
}
```

5.5.11.3. AI_ReadValue (float* o_fAIValue)

Read the real AI value without channel status.
The reading value is calculated, users no need to convert it to real value for current input type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AI_ReadValue (
  float *o_fAIValue
)
```

Parameter

`*o_fAIValue`

[OUT] The true value of AI value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fAIValue[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadValue(o_fAIValue)))
  printf("%d", iErrCode);
  else
  {
    For(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("%.5f\n", o_dwAIValue[iIdx]);
  }
}
```

5.5.11.4. AI_ReadValue (float* o_fAIValue, BYTE* o_byAIChStatus)

Read the real AI value with channel status.

Syntax

```
public int AI_ReadValue (
  float *o_fAIValue
  BYTE *o_byAIChStatus
)
```

Parameter

*o_fAIValue

[OUT] The true value of AI value.

*o_byAIChStatus

[OUT] The bytearray of channel status.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
floato_fAIValue[USBIO_AI_MAX_CHANNEL];
Byte o_byAIChStatus[USBIO_AI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadValue(o_fAIValue, o_byAIChStatus)))
  printf("%d", iErrCode);
  else
  {
    For(iIdx = 0; iIdx <USBIO_AI_MAX_CHANNEL; iIdx++)
    printf("%.5f,  0x%02x\n", o_fAIValue[iIdx], o_byAIChStatus);
  }
}
```

5.5.12. AI_ReadBulkValue

Trigger reading bulk AI value (Fast acquire functionality).
When in callback operation, it will cause the performance in your callback function. Please reduce execute time in this callback function.
The detail of operation is described as follow. When call this API, the AI module operation will be changed from normal to fast acquire mode. In fast acquire mode, AI module follow the parameter of API setting to acquire data.
The API has block and non-block operation. In block operation, user’ s application needs to wait until API finishing all procedure. In contrast with block mode, non-block provides a flexible way for user. In non-block operation, user’ s application can proceed to own other code. To enable non-block operation, it is important to declare a callback function and pass it through last parameter. For block operation, just pass a NULL definition in last parameter.
Due to the USB 2.0 Full-speed transfer rate capability, the maximum sample rate is 10 KHz.

Syntax

```
public int AI_ReadBulkValue (
  BYTE i_byStartCh,
  BYTE i_byChTotal,
  DWORD i_dwSampleWidth,
  Float i_fSampleRate,
  DWORD i_dwBufferWidth,
  DWORD *o_dwDataBuffer,
  OnBulkValueFinishEvent i_CBFunc
)
```

Parameter

i_byStartCh

[IN]The starting acquire channel

i_byChTotal

[IN] The total channels to acquire

i_dwSampleWidth

[IN] The sampling width (ms)

i_fSampleRate

[IN] The sampling rate (Hz). 10KHz maximum.

i_dwBufferWidth

[IN] The width of the buffer for single channel

*o_dwDataBuffer

[OUT] The 2-dimension buffer array to store

i_CBFunc

[IN] Block operation – NULL

[IN] Non-block operation - The address of callback function.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
// To read 0~1 channel for 100ms in 5 KHz sample rate each channel in non-block
operation
// So we have the following variable declaration
#define SampleRate 5000
#define BufferWidth 500; // 5000(Hz) * 0.1(100ms)
DWORD m_dwBuffer[2][BufferWidth];

Void BulkFinishCallback(DWORD dwCount)
{
  // Callback function to handle data
}

Int main()
{
  m_usbIO = new ICPDAS_USBIO();
  if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
  {
    if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadBulkValue(0,
    2,
    100,
    SampleRate,
    BufferWidth
    m_dwBuffer,
    BulkFinishCallback)))
    printf("%d", iErrCode);

    while(1) {Sleep(1);}
  }
}
```

5.5.13. AI_ReadCJCValue

Readthe current CJC value on the module.

Syntax

```
public int AI_ReadCJCValue (
float *o_fCJCValue
)
```

Parameter

*o_fCJCValue

[OUT] The CJC value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float o_fCJCValue;
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ReadCJCValue(o_fCJCValue)))
  printf("%d", iErrCode);
  else
  printf("%.5f\n", o_fCJCValue);
}
```

5.5.14. AI_GetChSampleRate

Get Sample rate of channel.

Syntax

```
public int AI_ GetChSampleRate (
  DWORD *o_dwSample
)
```

Parameter

`*o_dwSample`

[OUT] The Sample rate value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD* o_dwSampleRate;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ GetChSampleRate (o_ dwSampleRate)))
  printf("%d", iErrCode);
}
```

5.5.15. AI_ReadLatchValue

Read Latch value.

Syntax

```
public int AI_ReadLatchValue (
  DWORD *o_dwAIHiLatch,
  DWORD *o_dwAILoLatch
)
```

Parameter

o_ dwAIHiLatch

[OUT] The High Latch value.

o_ dwAILoLatch

[OUT] The Low Latch value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD* o_dwAIHiLatch;
DWORD* o_dwAILoLatch;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ ReadLatchValue (o_ dwAIHiLatch, o_
  dwAILoLatch)))
  printf("%d", iErrCode);
  else
  for(int i=0;i<(int) m_usbIO.GetAITotal();i++)
  printf("%d,%d\n", o_ dwAIHiLatch[i], o_ dwAILoLatch[i]);
}
```

5.5.16. AI_GetAlarmEnable

Get Alarm Enable.

Syntax

```
public int AI_GetAlarmEnable (
  BYTE *o_byChHiEnable,
  BYTE *o_byChLoEnable
)
```

Parameter

o_ byChHiEnable

[OUT] The High Alarm Enable.

o_ byChLoEnable

[OUT] The Low Alarm Enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE * o_byChHiEnable;
BYTE * o_byChLoEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetAlarmEnable (o_byChHiEnable,
  o_byChLoEnable)))
  printf("%d", iErrCode);
  else
  printf("%02X, %02X\n", o_byChHiEnable, o_byChLoEnable);
}
```

5.5.17. AI_GetAlarmMode

Get Alarm Mode.

Syntax

```
public int AI_GetAlarmMode (
  BYTE *o_byChHiAlrmMode,
  BYTE *o_byChLoAlrmMode
)
```

Parameter

o_ byChHiAlrmMode

[OUT] The High Alarm Mode.

o_ byChLoAlrmMode

[OUT] The Low Alarm Mode.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE * o_byChHiAlrmMode;
BYTE * o_byChLoAlrmMode;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_ GetAlarmMode (o_byChHiAlrmMode,
  o_byChLoAlrmMode)))
  printf("%d", iErrCode);
  else
  printf("%02X, %02X\n", o_ byChHiAlrmMode, o_ byChLoAlrmMode);
}
```

5.5.18. AI_GetAlarmLimit

Get Alarm Limit.

Syntax

```
public int AI_GetAlarmLimit (
  WORD *o_wChHiAlrmLimit,
  WORD *o_wChLoAlrmLimit
)
```

Parameter

o_ wChHiAlrmLimit

[OUT] The High Alarm Limit value.

o_ wChLoAlrmLimit

[OUT] The Low Alarm Limit value. Each bit represents the DO value of each channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORD* o_ wChHiAlrmLimit;
WORD* o_ wChLoAlrmLimit;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetAlarmLimit (o_wChHiAlrmLimit,
  o_wChLoAlrmLimit)))
  printf("%d", iErrCode);
  else
  for(int i=0;i<((int) m_usbIO.GetAITotal());i++)
  printf("%04X,%04X\n", o_ wChHiAlrmLimit [i], o_ wChLoAlrmLimit [i]);
}
```

5.5.19. AI_GetAlarmStatus

Get Alarm Status.

Syntax

```
public int AI_GetAlarmLimit (
  BYTE *o_bChHiAlrmStatus,
  BYTE *o_bChLoAlrmStatus
)
```

Parameter

o_ bChHiAlrmStatus

[OUT] The High Alarm Status.

o_ bChLoAlrmStatus

[OUT] The Low Alarm Status.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE* o_bChHiAlrmStatus;
BYTE* o_bChLoAlrmStatus;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_GetAlarmStatus (o_bChHiAlrmStatus,
  o_bChLoAlrmStatus)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.20. AI_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channel.Please refer to user's manual for analog input type code.These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.1.

Name of Methods
AI_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)
AI_SetTypeCode (BYTE *i_byTypeCodes)

5.5.20.1. AI_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)

Set type code for specific channel. The type code can reference to Appendix A.1.

Syntax

```
public int AI_SetTypeCode (
  BYTE i_byChToSet,
  BYTE i_byTypeCode
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_byTypeCode

[IN] The type code for the specific channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetTypeCode(0, 0x10)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.20.2. AI_SetTypeCode (BYTE *i_byTypeCodes)

Set type code for all channels. The type code can reference to Appendix A.1.

Syntax

```
public int AI_SetTypeCode (
  BYTE *i_byTypeCodes
)
```

Parameter

`*i_byTypeCodes`

[IN] The byte array of type code to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTypeCode[USBIO_AI_MAX_CHANNEL];
Int iIdx;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx)
  m_byChTypeCode[iIdx] = 0x10;
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetTypeCode(m_byChTypeCode)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.21. AI_SetChCJCOffset

The class has two overload methods for setting channel CJC offset. One provides specifying channel to set, another for all channel.The valid range of offset is -40.96 ~ +40.95. These two overload methods are listed as following table and described in following section.

Name of Methods
AI_SetChCJCOffset (BYTEi_byChToSet, floati_fChCJCOffset)
AI_SetChCJCOffset (float *i_fChCJCOffsets)

5.5.21.1. AI_SetChCJCOffset (BYTEi_byChToSet, floati_fChCJCOffset)

Set channel CJC offset for specific channel.

Syntax

```
public int AI_SetTypeCode (
  BYTE i_byChToSet,
  float i_fChCJCOffset
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_fChCJCOffset

[IN] The CJC offset for the specific channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChCJCOffset(0, 1.354)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.21.2. AI_SetChCJCOffset (float *i_fChCJCOffsets)

Set channel CJC offset for specific channel.

Syntax

```
public int AI_SetTypeCode (
  float *i_fChCJCOffset
)
```

Parameter

*i_fChCJCOffset

[IN] The float array of channel CJC offset to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
float m_fChCJCOffset[USBIO_AI_MAX_CHANNEL];
Int iIdx;
m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_AI_MAX_CHANNEL; iIdx)
  m_fChCJCOffset[iIdx] = 1.358;
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChCJCOffset(m_fChCJCOffset)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.22. AI_SetChEnable

Set channel enable/disable. Each byte indicates 8 channels enable/disable mask. Ex: Byte 0 -> Channel 0 ~ 7.

Syntax

```
public int AI_SetChEnable (
  BYTE *i_byChEnable
)
```

Parameter

`*o_byChEnable`

[IN] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChEnable[(USBIO_AI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx <( USBIO_AI_MAX_CHANNEL + 7) / 8; iIdx)
  m_byChEnable [iIdx] = 0x5A;
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetChEnable(m_byChEnable)))
  printf(“%d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.23. AI_SetFilterRejection

Set filter rejection.

Rejection Setting	Value
60Hz	0
50Hz	1

Syntax

```
public int AI_SetFilterRejection (
  BYTE i_byFilterRejection
)
```

Parameter

i_byFilterRejection

[IN] The filter rejection.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetFilterRejection(0)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.24. AI_SetCJCOffset

Set CJC offset. The valid range of offset is -40.96 ~ +40.95..

Syntax

```
public int AI_SetCJCOffset (
  float i_fCJCOffset
)
```

Parameter

i_fCJCOffset

[IN] The CJC offset.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetCJCOffset(-20.81)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.25. AI_SetCJCEnable

Set CJC enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_GetCJCEnable (
  BYTE *i_byCJCEnable
)
```

Parameter

`*i_byCJCEnable`

[IN] The CJC enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetCJCEnable(1)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.26. AI_SetWireDetectEnable

Set wire detect enable.

Enable Setting	Value
Disable	0
Enable	1

Syntax

```
public int AI_SetCJCOffset (
  BYTE i_byWireDetectEnable
)
```

Parameter

i_byWireDetectEnable

[IN] The wire detect enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AI_SetWireDetectEnable(0)))
  printf("%d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.27. AI_SetChSampleRate

Set Sample rate of channel.

Syntax

```
public int AI_SetChSampleRate(
  DWORD *i_dwSample
)
```

Parameter

i_ dwSample

[IN] The value for sample rate.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD* i_dwSample

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_ SetChSampleRate (i_ dwSample)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.28. AI_SetAlarmEnable

Set Alarm Enable.

Alarm Enable	Value
Disable & Off	0
Enable & On	1

Syntax

```
public int AI_SetAlarmEnable (
BYTE i_byChHiAlrmEnable,
BYTE i_byChLoAlrmEnable
)
```

Parameter

i_byChHiAlrmEnable

[IN] The Alarm high enable.

i_byChLoAlrmEnable

[IN] The Alarm low enable.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE i_byChHiAlrmEnable;
BYTE i_byChLoAlrmEnable;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_SetAlarmEnable (i_byChHiAlrmEnable, i_byChLoAlrmEnable)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.29. AI_SetAlarmMode

Set Alarm mode.

Alarm Mode	Value
Momentary Mode	0
Latch Mode	1

Syntax

```
public int AI_SetAlarmMode (
BYTE i_byChHiAlrmMode,
BYTE i_byChLoAlrmMode
)
```

Parameter

i_byChHiAlrmMode

[IN] The Alarm high mode.

i_byChLoAlrmMode

[IN] The Alarm low mode.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE i_byChHiAlrmMode;
BYTE i_byChLoAlrmMode;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_SetAlarmEnable (i_byChHiAlrmMode, i_byChLoAlrmMode)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.30. AI_SetAlarmLimit

The class provides two overloaded methods for setting the alarm limit. One method, which takes a parameter in float format, converts the raw value to a true value inside the method. The other method sets the raw value without conversion. An overview of these methods is provided in the following table and described in the subsequent section.

Name of Methods
AI_SetAlarmLimit (WORD* i_wChHiAlrmLimit, WORD* i_wChLoAlrmLimit)
AI_SetAlarmLimit (float* i_fChHiAlrmLimit, float* i_fChLoAlrmLimit)

5.5.30.1. AI_SetAlarmLimit (WORD* i_wChHiAlrmLimit, WORD* i_wChLoAlrmLimit)

Set AI Alarm Limit to specifying channel in word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public int AI_SetAlarmLimit (
  WORD *i_wChHiAlrmLimit,
  WORD *i_wChLoAlrmLimit
)
```

Parameter

i_wChHiAlrmLimit

[IN] The High Alarm Limit of AI channel to be set.

i_wChLoAlrmLimit

[IN] The Low Alarm Limit of AI channel to be set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE i_ wChHiAlrmLimit;
BYTE i_ wChLoAlrmLimit;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_ SetAlarmLimit (i_ wChHiAlrmLimit, i_ wChLoAlrmLimit)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.30.2. AI_SetAlarmLimit (float* i_fChHiAlrmLimit, float* i_fChLoAlrmLimit)

Set AI Alarm Limit to specifying channel in float (digital) format.

Syntax

```
public int AI_SetAlarmLimit (
  float * i_fChHiAlrmLimit,
  float * i_fChLoAlrmLimit
)
```

Parameter

i_ fChHiAlrmLimit

[IN] The High Alarm Limit of AI channel to be set.

i_ fChLoAlrmLimit

[IN] The Low Alarm Limit of AI channel to be set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE i_ wChHiAlrmLimit;
BYTE i_ wChLoAlrmLimit;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_ SetAlarmLimit (i_ fChHiAlrmLimit, i_ fChLoAlrmLimit)
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.31. AI_ClearLatchValue

Clear Latch Value for USB2200 Service.

Clear Latch Value	Value
Latch Low Value	0
Latch High Value	1

Syntax

```
public int AI_ ClearLatchValue (
  BYTE b_HiLo
)
```

Parameter

b_HiLo

[IN] Clear the Latch Value of all channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE b_HiLo ;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_ ClearLatchValue (b_HiLo);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.5.32. AI_ClearAlarmLatchStatus

Clear Alarm Latch Status.

Select hi/lo	Value
Low	0
High	1

Clear Latch Status	Value
Normal	0
Clear	1

.

Syntax

```
public int AI_ ClearLatchValue (
  BYTE i_bHiLo,
  BYTE i_bClrMask
)
```

Parameter

i_bHiLo

[IN] Select hi/lo forClear the Latch Value.

i_ bClrMask

[IN] Set value for Clear the Latch Value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE i_bHiLo ;
BYTE i_bClrMask;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO. AI_ ClearAlarmLatchStatus (i_bHiLo, i_bClrMask);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.6. Analog Output

This section presents methods for Analog Output functionality. It includes output value setting, power-on/safety parameters, resolution configuration, and slew rate control for analog output applications.

Name	Description
AO_GetTotalSupportType	Get total supported amount.
AO_GetSupportTypeCode	Get supported type code.
AO_GetTypeCode	Get type code.
AO_GetChEnable	Get channel enable/disable.
AO_GetResolution	Get resolution.
AO_ReadExpValue	Read AO expects value in double word format.
AO_ReadCurValue	Read AO current value in double word format.
AO_GetPowerOnEnable	Get Power-On Enable.
AO_GetSafetyEnable	Get Safety Enable.
AO_GetPowerOnValue	Get Power-On value.
AO_GetSafetyValue	Get Safety value.
AO_GetSlewRate	Get Slew rate.
AO_SetTypeCode	Analog output function –Set type code for specific channel.
AO_SetChEnable	Set channel enable/disable.
AO_WriteValue	Write AO value in double word format.
AO_SetPowerOnEnable	Set Power-On Enable.
AO_SetSafetyEnable	Set Safety Enable.
AO_SetPowerOnValue	Set Power-On Value.
AO_SetSafetyValue	Set Safety Value.
AO_SetSlewRate	Set Slew rate.

5.6.1. AO_GetTotalSupportType

Get total supported amount.

Syntax

```
public int AO_GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTotalSupportType;
BYTE o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
int iIdx;
bool bRet = true;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTotalSupportType
  (&o_byTotalSupportType)))
  {
    printf(" %d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AO_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
return iErrCode;
```

5.6.2. AO_GetSupportTypeCode

Get supported type code Please refer to Appendix A.2 of user's manual to map AO channels output type.

Syntax

```
public int AO_GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTotalSupportType;
BYTE o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
int iIdx;
bool bRet = true;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTotalSupportType
  (&o_byTotalSupportType)))
  {
    printf(" %d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.AO_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
return iErrCode;
```

5.6.3. AO_GetTypeCode

Get type code Please refer to user's manual to map AO channels input type. The type code can reference to Appendix A.2.

Syntax

```
public int AO_GetTypeCode (
  BYTE *o_byTypeCode
)
```

Parameter

*o_byTypeCode

[OUT] The byte array of type code.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetTypeCode (o_byTypeCode)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%02x\n", o_byTypeCode[iIdx]);
  }
}
return iErrCode;
```

5.6.4. AO_GetChEnable

Get channel enable/disable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AO_GetChCJCOffset (
  BYTE *o_byChEnable
)
```

Parameter

`*o_byChEnable`

[OUT] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byChEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetChEnable (o_byChEnable)))
   printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
    printf("0x%02x\n", o_byChEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.5. AO_GetResolution

Get resolution. Each byte indicates each channel real resolution.

Syntax

```
public int AO_GetResolution (
  BYTE *o_byResolution
)
```

Parameter

`*o_byResolution`

[OUT] The byte array of resolution for each channel.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byResolution[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetResolution (o_byResolution)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%d\n", o_byResolution[iIdx]);
  }
}
return iErrCode;
```

5.6.6. AO_ReadExpValue

The class library provides 2 overload methods to read AO expect value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ReadExpValue (DWORD* o_dwAOExpValue)
AO_ReadExpValue (float* o_fAOExpValue)

5.6.6.1. AO_ReadExpValue (DWORD* o_dwAOExpValue)

Read AO expect value in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

☑ Channel Over

The reading value represents a sign value X indicates how many value over full scale range. This value can be calculated by following formula:
Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $Y = \left(1 + \frac{0X13E}{0xFFFF}\right) \times \left(10 - (-10)\right) + (-10)$

☑ Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

☑ Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AI value cannot detect the channel status. It only read the AI value but has the most efficiency.

Syntax

```
public int AO_ReadExpValue (
  DWORD *o_dwAOExpValue
)
```

Parameter

`*o_dwAOExpValue`

[OUT] The raw value of AO expect value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwAOExpValue[iIdx]);
  }
}
return iErrCode;
```

5.6.6.2. AO_ReadExpValue (float* o_fAOExpValue)

Readthe real AO expect value.
The reading value is calculated, users no need to convert it to real value for expectinput type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AO_ReadExpValue (
  float *o_fAOExpValue
)
```

Parameter

`*o_fAOExpValue`

[OUT] The true value of AO expect value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fAOExpValue[iIdx]);
  }
} return iErrCode;
```

5.6.7. AO_ReadCurValue

The class library provides 2 overload methods to read AO current value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ReadCurValue (DWORD* o_dwAOCurValue)
AO_ReadCurValue (float* o_fAOCurValue)

5.6.7.1. AO_ReadCurValue (DWORD* o_dwAOCurValue)

Read AO current value in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

☑ Channel Over

The reading value represents a sign value X indicates how many value over full scale range.
This value can be calculated by following formula:
Assume current type is -10V ~ +10V with 16 bit resolution and reading value is 0x13E, then we can get the actual value Y is $Y = \left(1 + \frac{0X13E}{0xFFFF}\right) \times \left(10 - (-10)\right) + (-10)$

☑ Channel Under

The reading value represents a sign value X indicates how many value under zero scale range.
This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

☑ Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

The overload API for only reading AI value cannot detect the channel status. It only read the AI value but has the most efficiency.

Syntax

```
public int AO_ReadCurValue (
  DWORD *o_dwAOCurValue
)
```

Parameter

`*o_dwAOCurValue`

[OUT] The raw value of AO current value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOCurValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadCuValue(o_dwAOCurValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwAOCurValue[iIdx]);
  }
}
return iErrCode;
```

5.6.7.2. AO_ReadCurValue (float* o_fAOCurValue)

Readthe real AO current value.
The reading value is calculated, users no need to convert it to real value for currentinput type. Ex: The reading value is 1.316 in -2.5 ~ +2.5V, the input signal is 1.316V.

Syntax

```
public int AO_ReadCurValue (
  float *o_fAOCurValue
)
```

Parameter

`*o_fAOCurValue`

[OUT] The true value of AO current value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOCurValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOCurValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fAOCurValue[iIdx]);
  }
} return iErrCode;
```

5.6.8. AO_GetPowerOnEnable

Get Power-OnEnable.Each channeltakes one byte.

Syntax

```
public int AO_GetPowerOnEnable(
  BYTE *o_byPowerOnEnable
)
```

Parameter

*o_byPowerOnEnable

OUT] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byPwrOnEnable [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnEnable (o_byPwrOnEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%02x\n", o_byPwrOnEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.9. AO_GetSafetyEnable

Get SafetyEnable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AO_GetSafetyEnable (
  BYTE *o_bySafetyEnable
)
```

Parameter

*o_ bySafetyEnables

[OUT] The byte array of Safety enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_bySafetyEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyEnable (o_bySafetyEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
    printf("0x%02x\n", o_bySafetyEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.10. AO_GetPowerOnValue

The class library provides 2 overload methods to read AO Power-On Value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ GetPowerOnValue(DWORD* o_ dwPwrOnValue)
AO_ GetPowerOnValue(float* o_fPwrOnValue)

5.6.10.1. AO_ GetPowerOnValue(DWORD* o_ dwPwrOnValue)

Get Power-On Value.Each channel takes one unit of DWORD array.

Syntax

```
public int AO_GetPowerOnValue(
  DWORD *o_dwPwrOnValue
)
```

Parameter

`*o_dwPwrOnValue`

[OUT] The DWORD array of Power-On Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPwrOnValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPwrOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwPwrOnValue[iIdx]);
  }
}
return iErrCode;
```

5.6.10.2. AO_ GetPowerOnValue(float* o_fPwrOnValue)

GetPower-On Value.Each channel takes one unit of float array.

Syntax

```
public int AO_GetPowerOnValue (
float* o_fPwrOnValue
)
```

Parameter

*o_fPwrOnValue

[OUT] The float array of Power-On Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPwrOnValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_ fPwrOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_ fPwrOnValue [iIdx]);
  }
}
return iErrCode;
```

5.6.11. AO_GetSafetyValue

The class library provides 2 overload methods to read AO Safety Value. One method, the parameter in float format, will convert raw value to true inside the method. The other will return raw value without having conversion. The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_ GetSafetyValue (DWORD* o_ dwSafetyValue)
AO_ GetSafetyValue(float* o_fSafetyValue)

5.6.11.1. AO_ GetSafetyValue (DWORD* o_ dwSafetyValue)

Get Safety value.Each channel takes one unit of DWORD array.

Syntax

```
public int AO_GetSafetyValue(
  DWORD *o_dwSafetyValue
)
```

Parameter

*o_ dwSafetyValue

[OUT] The DWORD array of Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyValue (o_dwSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.11.2. AO_ GetSafetyValue(float* o_fSafetyValue)

Get Safety value.Each channel takes one unit of float array.

Syntax

```
public int AO_ GetSafetyValue(
  float *o_fSafetyValue
)
```

Parameter

`*o_fSafetyValue`

[OUT] The float array of Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyValue (o_fSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.12. AO_GetSlewRate

Get Slew rate.

Syntax

```
public int AO_ GetSlewRate (
  BYTE* o_bySlewRate
);
```

Parameter

`*o_bySlewRate`

[OUT] The AO slew rate Value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE* o_bySlewRate

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO.AO_GetSlewRate(o_bySlewRate);
}
```

5.6.13. AO_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channel. Please refer to user's manual for analog output type code. These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.2.

Name of Methods
AO_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)
AO_SetTypeCode (BYTE *i_byTypeCodes)

5.6.13.1. AO_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)

Set type code for specific channel. The type code can reference to Appendix A.2.

Syntax

```
public int AO_SetTypeCode (
  BYTE i_byChToSet,
  BYTE i_byTypeCode
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_byTypeCode

[IN] The type code for the specific channel.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetTypeCode(0, 0x30)))
  printf(“%d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
else
return iErrCode;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_GetTypeCode (o_byTypeCode)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%02x\n", o_byTypeCode[iIdx]);
  }
}
return iErrCode;
```

5.6.13.2. AO_SetTypeCode (BYTE *i_byTypeCodes)

Set type code for all channels. The type code can reference to Appendix A.2.

Syntax

```
public int AO_SetTypeCode (
  BYTE *i_byTypeCodes
)
```

Parameter

`*i_byTypeCodes`

[IN] The byte array of type code to set.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byTypeCode [USBIO_AO_MAX_CHANNEL];
BYTE m_byChTypeCode[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{

  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  {
    m_byChTypeCode[iIdx] = 0x30;
  }

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetTypeCode(m_byChTypeCode)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetTypeCode(o_byTypeCode)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%02x\n", o_byTypeCode[iIdx]);
  }

  iErrCode = m_usbIO.CloseDevice();
}
return iErrCode;
```

5.6.14. AO_SetChEnable

Set channel enable/disable. Each byte indicates 8 channels enable/disable mask. Ex: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AO_SetChEnable (
  BYTE *i_byChEnable
)
```

Parameter

`*i_byChEnable`

[IN] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE m_byChEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8];
BYTE o_byChEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx <( USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
  m_byChEnable [iIdx] = 0x03;
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetChEnable(m_byChEnable)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetChEnable (o_byChEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
    printf("0x%02x\n", o_byChEnable [iIdx]);
  }

  iErrCode = m_usbIO.CloseDevice();
}
return iErrCode;
```

5.6.15. AO_WriteValue

The class library provides 4 overload methods to write AO expect value. Two method provides specifying channel to write raw data and converted value.The others write AO value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_WriteValue (BYTE i_byChToSet, DWORD i_dwAOVal)
AO_ WriteValue (DWORD* i_dwAOValue)
AO_ WriteValue (BYTE i_byChToSet, float i_fAOExpValue)
AO_ WriteValue (float* i_fAOExpValue)

5.6.15.1. AO_WriteValue (BYTE i_byChToSet, DWORD i_dwAOVal)

Write AO expect value to specifying channel in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public intAO_WriteValue(
  BYTE i_byChToSet,
  DWORD i_dwAOVal
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_dwAOVal

[IN] TheAO value .

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwAOValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(byChannel, dwAOValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwAOExpValue[iIdx]);
  }
}
return iErrCode;
```

5.6.15.2. AO_ WriteValue (DWORD* i_dwAOValue)

Write AO expect value to all channels in double word (digital) format.

Syntax

```
public intAO_WriteValue(
  DWORD *i_dwAOVal
);
```

Parameter

i_dwAOVal

[IN] TheAO value .

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwAOExpValue[USBIO_AO_MAX_CHANNEL];
DWORD dwAOValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  dwAOValue[iIdx] = 0xff;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(dwAOValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_dwAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwAOExpValue[iIdx]);
  }
}
return iErrCode;
```

5.6.15.3. AO_ WriteValue (BYTE i_byChToSet, float i_fAOExpValue)

Write AO expect value to specifying channel in float (analog) format.
The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_WriteValue(
  BYTE i_byChToSet,
  DWORD i_fAOVal
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_fAOVal

[IN] TheAO value .

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fAOValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(byChannel, fAOValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fAOExpValue[iIdx]);
  }
}
return iErrCode;
```

5.6.15.4. AO_ WriteValue (float* i_fAOExpValue)

Write AO expect value to all channels in float (analog) format.
The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_WriteValue(
  float *i_fAOVal
);
```

Parameter

i_ fAOVal

[IN] TheAO value .

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fAOExpValue[USBIO_AO_MAX_CHANNEL];
float fAOValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  fAOValue[iIdx] = 5;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_WriteValue(fAOValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO. AO_ReadExpValue(o_fAOExpValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fAOExpValue[iIdx]);
  }
}
return iErrCode;
```

5.6.16. AO_SetPowerOnEnable

Set Power-OnEnable.Each channeltakes one byte.

Syntax

```
public int AO_SetPowerOnEnable (
  BYTE *o_byPowerOnEnable
)
```

Parameter

*o_byPowerOnEnable

[OUT] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_byPwrOnEnable [USBIO_AO_MAX_CHANNEL];
BYTE i_bySetPwrOnEnable[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  i_bySetPwrOnEnable[iIdx] = 0x01;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnEnable(i_bySetPwrOnEnable)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnEnable (o_byPwrOnEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%02x\n", o_byPwrOnEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.17. AO_SetSafetyEnable

Set SafetyEnable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int AO_SetSafetyEnable (
  BYTE *o_bySafetyEnable
)
```

Parameter

*o_bySafetyEnable

[OUT] The byte array of channel enable/disable mask.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
BYTE o_bySafetyEnable [(USBIO_AO_MAX_CHANNEL + 7) / 8];
BYTE i_bySetSafetyEnable[(USBIO_AO_MAX_CHANNEL + 7) / 8] = {0x03};
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetSafetyEnable(i_bySetSafetyEnable)))
   printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetSafetyEnable (o_bySafetyEnable)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < (USBIO_AO_MAX_CHANNEL + 7) / 8; iIdx++)
    printf("0x%02x\n", o_bySafetyEnable [iIdx]);
  }
}
return iErrCode;
```

5.6.18. AO_SetPowerOnValue

The class library provides 4 overload methods to write AO Power-On Value. Two method provides specifying channel to write raw data and converted value.The others write AO Power-On Value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_SetPowerOnValue (BYTE i_byChToSet, DWORD i_dwPowerOnValue)
AO_SetPowerOnValue (DWORD* i_dwPowerOnValue)
AO_SetPowerOnValue (BYTE i_byChToSet, float i_fPowerOnValue)
AO_SetPowerOnValue (float* i_fPowerOnValue)

5.6.18.1. AO_SetPowerOnValue (BYTE i_byChToSet, DWORD i_dwPowerOnValue)

Set AO Power-On Value to specifying channel in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public intAO_ SetPowerOnValue(
  BYTE i_byChToSet,
  DWORD i_dwPowerOnValue
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_dwPowerOnValue

[IN] TheAO PowerOnValue .

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPowerOnValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwSetPowerOnValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel,
  dwSetPowerOnValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPowerOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwPowerOnValue[iIdx]);
  }
}
return iErrCode;
```

5.6.18.2. AO_SetPowerOnValue (DWORD* i_dwPowerOnValue)

Set AO Power-On Value to all channels in double word (digital) format.

Syntax

```
public intAO_SetPowerOnValue(
  DWORD* i_dwPowerOnValue
);
```

Parameter

i_dwPowerOnValue

[IN] TheAO Power-On Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwPowerOnValue [USBIO_AO_MAX_CHANNEL];
DWORD dwSetPowerOnValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  dwSetPowerOnValue[iIdx] = 0xff;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (dwSetPowerOnValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwPowerOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwPowerOnValue[iIdx]);
  }
}
return iErrCode;
```

5.6.18.3. AO_SetPowerOnValue (BYTE i_byChToSet, float i_fPowerOnValue)

Set AO Power-On Value to specifying channel in float (analog) format.
The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_ SetPowerOnValue(
  BYTE i_byChToSet,
  DWORD i_fPowerOnValue
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_ fPowerOnValue

[IN] TheAO Power-On Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPowerOnValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fSetPowerOnValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel,
  fSetPowerOnValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fPowerOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fPowerOnValue[iIdx]);
  }
}
return iErrCode;
```

5.6.18.4. AO_SetPowerOnValue (float* i_fPowerOnValue)

Set AO Power-On Value to all channels in float (analog) format.
The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_ SetPowerOnValue(
  float* i_ fPowerOnValue
);
```

Parameter

i_fPowerOnValue

[IN] TheAO Power-On Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fPowerOnValue [USBIO_AO_MAX_CHANNEL];
float fSetPowerOnValue[USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  fSetPowerOnValue[iIdx] = 5;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (fSetPowerOnValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fPowerOnValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fPowerOnValue[iIdx]);
  }
}
return iErrCode;
```

5.6.19. AO_SetSafetyValue

The class library provides 4 overload methods to write AO Safety Value. Two method provides specifying channel to write raw data and converted value.The others write AO Safety Value for all channels.The overview of these methods is as following table, and will describe in the following section.

Name of Methods
AO_SetSafetyValue (BYTE i_byChToSet, DWORD i_dwSafetyValue)
AO_SetSafetyValue (DWORD* i_dwSafetyValue)
AO_SetSafetyValue (BYTE i_byChToSet, float i_fSafetyValue)
AO_SetSafetyValue (float* i_fSafetyValue)

5.6.19.1. AO_SetSafetyValue (BYTE i_byChToSet, DWORD i_dwSafetyValue)

Set AO Safety Value to specifying channel in double word (digital) format.
In the digital format, the value represents the value from zero to full scale. Ex: For type -10V ~ +10V, the value 0x0 indicates -10V and 0xFFFF (16bit resolution) indicates +10V.
Please note that, when channel was not in good status, the reading value no longer represents zero to full scale. Different channel status follows the following rule:

Channel Under

The reading value represents a sign value X indicates how many value under zero scale range. This value can be calculated by following formula:
Assume current type is -5V ~ +5V with 16 bit resolution and reading value is 0x53E, then we can get the actual value Y is $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times \left(5 - (-5)\right) + (-5)$

Channel Open & Channel Close

The reading value of these two statuses will be the full scale for channel open and zero scale for channel close.

Syntax

```
public intAO_ SetSafetyValue(
  BYTE i_byChToSet,
  DWORD i_dwSafetyValue
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_dwSafetyValue

[IN] TheAO Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
DWORD dwSetSafetyValue = 0xff;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel,
  dwSetSafetyValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.19.2. AO_SetSafetyValue (DWORD* i_dwSafetyValue)

Set AO Safety Value to all channels in double word (digital) format.

Syntax

```
public intAO_ SetSafetyValue(
  DWORD* i_dwSafetyValue
);
```

Parameter

i_dwSafetyValue

[IN] TheAO Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
DWORD o_dwSafetyValue [USBIO_AO_MAX_CHANNEL];
DWORD dwSetSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  dwSetSafetyValue[iIdx] = 0xff;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (dwSetSafetyValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_dwSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("0x%04x\n", o_dwSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.19.3. AO_SetSafetyValue (BYTE i_byChToSet, float i_fSafetyValue)

Analog output function - Set AO Safety Value to specifying channel in float (analog) format. The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_ SetSafetyValue(
  BYTE i_byChToSet,
  DWORD i_fSafetyValue
);
```

Parameter

i_byChToSet

[IN] The specific AO channel to be set.

i_ fSafetyValue

[IN] TheAO Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue [USBIO_AO_MAX_CHANNEL];
BYTE byChannel = 0;
float fSetSafetyValue = 5;
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (byChannel,
  fSetSafetyValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.19.4. AO_SetSafetyValue (float* i_fSafetyValue)

Set AO Safety Value to all channels in float (analog) format.
The writting value is calculated, users write real value for currentoutput type. Ex: The writting value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

Syntax

```
public intAO_ SetSafetyValue(
  float* i_fSafetyValue
);
```

Parameter

i_fSafetyValue

[IN] TheAO Safety Value.

Return Value

Error code.

Example

```
int iErrCode;
ICPDAS_USBIO m_usbIO;
float o_fSafetyValue [USBIO_AO_MAX_CHANNEL];
float fSetSafetyValue [USBIO_AO_MAX_CHANNEL];
int iIdx;

if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
  fSetSafetyValue[iIdx] = 5;

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_SetPowerOnValue (fSetSafetyValue)))
  printf("%d", iErrCode);

  if(ERR_NO_ERR != (iErrCode = m_usbIO.AO_GetPowerOnValue (o_fSafetyValue)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx < USBIO_AO_MAX_CHANNEL; iIdx++)
    printf("%04f\n", o_fSafetyValue[iIdx]);
  }
}
return iErrCode;
```

5.6.20. AO_SetSlewRate

Set Slew rate.

Syntax

```
public int AO_ SetSlewRate (
  BYTE* i_bySlewRate
);
```

Parameter

i_bySlewRate

[IN] The AO slew rate Value.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
BYTE* i_bySlewRate

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  iErrCode = m_usbIO.AO_ SetSlewRate(i_bySlewRate);
}
```

5.7. Pulse Input

This section explains methods for Pulse Input functionality. It supports reading frequency and count values, configuring trigger modes, filtering parameters, and channel isolation for high-speed signal acquisition and event detection.

Name	Description
PI_GetTotalSupportType	Get total supported amount.
PI_GetSupportTypeCode	Get supported type code.
PI_GetTypeCode	Get type code.
PI_GetTriggerMode	Get trigger mode.
PI_GetLPFilterEnable	Get low-pass filter enable.
PI_GetChIsolatedFlag	Get channel isolated flag.
PI_GetLPFilterWidth	Get low-pass filter width.
PI_ReadValue	Read PI value.
PI_ReadCntValue	Read the count value of counters
PI_ReadFreqValue	Read the frequency value of counters
PI_ReadBulkValue	Get bulk PI value (Fast acquire functionality)
PI_SetTypeCode	Set type code for specific channel. (Overload)
PI_ClearChCount	Clear channel count with clear mask.
PI_ClearSinglelChCount	Clear single channel count.
PI_ClearChStatus	Clear channel status with clear mask.
PI_ClearSinglelChStatus	Clear single channel status.
PI_SetTriggerMode	Set trigger mode. (Overload)
PI_SetChIsolatedFlag	Set channel isolated flag. (Overload)
PI_SetLPFilterEnable	Set low-pass filter enable. (Overload)
PI_SetLPFilterWidth	Set low-pass filter width. (Overload)

5.7.1. PI_GetTotalSupportType

Get total supported amount.

Syntax

```
public int PI_GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetTotalSupportType(&o_
  byTotalSupportType)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.PI_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
```

5.7.2. PI_GetSupportTypeCode

Get supported type code.Please refer to Appendix A.3 of user's manual to map PI channels input type.

Syntax

```
public int PI_ GetTotalSupportType (
  BYTE *o_bySupportTypeCode
)
```

Parameter

*o_bySupportTypeCode

[OUT] The number of total support type.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byteo_byTotalSupportType;
Byte o_bySupportTypeCode[USBIO_MAX_SUPPORT_TYPE];
Int iIdx;
Bool bRet = true;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetTotalSupportType(&o_
  byTotalSupportType)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  if(ERR_NO_ERR != (iErrCode =
  m_usbIO.PI_GetSupportTypeCode(o_bySupportTypeCode)))
  {
    printf("%d", iErrCode);
    bRet = false;
  }
  If(bRet)
  {
    printf("%d\n", o_byTotalSupportType);
    for(iIdx = 0; iIdx < o_byTotalSupportType; iIdx++)
    printf("%02x\n", o_bySupportTypeCode[iIdx]);
  }
}
```

5.7.3. PI_GetTypeCode

Get type code. Please refer to user's manual to map PI channels input type. The type code can reference to Appendix A.3.

Syntax

```
public int PI_GetTypeCode (
  BYTE *o_byTypeCode
)
```

Parameter

*o_byTypeCode

[OUT] The byte array of type code.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTypeCode[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if(ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if(ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetTypeCode(o_byTypeCode)))
  printf("%d", iErrCode);
  else
  {
    for(iIdx = 0; iIdx <USBIO_PI_MAX_CHANNEL; iIdx++)
    printf("%02x\n", o_byTypeCode[iIdx]);
  }
}
```

5.7.4. PI_GetTriggerMode

Get trigger mode.

Trigger Mode	Code
Falling edge	0
Rising edge	1
Both edge	2 & 3

Syntax

```
public int PI_GetTriggerMode (
  BYTE *o_byTriggerMode
)
```

Parameter

*o_byTriggerMode

[OUT] The byte array of trigger mode.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byTriggerMode[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetTriggerMode(o_byTriggerMode)))
    printf("% d", iErrCode);
  else
  {
    for (iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
      printf("% 02x\n", o_byTriggerMode[iIdx]);
  }
}
```

5.7.5. PI_GetChIsolatedFlag

Get channel isolated flag.Each byte indicates 8 channels flag. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_GetChIsolatedFlag(
  BYTE *o_byChIsolatedFlag
)
```

Parameter

*o_byChIsolatedFlag

[OUT] The byte arrays of channel isolated flag.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byChIsolatedFlag[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode =
m_usbIO.PI_GetChIsolatedFlag(o_byChIsolatedFlag)))
    printf("% d", iErrCode);
  else
  {
    for (iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx++)
      printf("% 02x\n", o_byChIsolatedFlag[iIdx]);
  }
}
```

5.7.6. PI_GetLPFilterEnable

Get low-pass filter enable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_GetLPFilterEnable(
  BYTE* o_byLPFilterEnable
)
```

Parameter

E* o_byLPFilterEnable

[OUT] The byte array of the low-pass filter enable mask

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte o_byLPFilterEnable[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode =
m_usbIO.PI_GetLPFilterEnable(o_byLPFilterEnable)))
    printf("% d", iErrCode);
  else
  {
    for (iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx++)
      printf("% 02x\n", o_byLPFilterEnable[iIdx]);
  }
}
```

5.7.7. PI_GetLPFilterWidth

Get low-pass filter width. The unit of the width is uS. The maximum value of width is 32767uS. Note: Each channel does not use own low-pass filter width. Please refer to following table to see what low-pass filter width is referred to.

Channel Index	Set
0 & 1	0
2 & 3	1
4, 5, 6, 7	2 & 3

Syntax

```
public int PI_GetLPFilterWidth(
  WORD* o_wLPFilterWidth
)
```

Parameter

* o_wLPFilterWidth

[OUT] The byte array of the low-pass filter width in uS.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
WORDo_wLPFilterWidth[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_GetLPFilterWidth(o_wLPFilterWidth)))
    printf(“% d”, iErrCode);
  else
  {
    for (iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
      printf(“% d\n”, o_wLPFilterWidth[iIdx]);
  }
}
```

5.7.8. PI_ReadValue

Get PI value in double-word format. This method provides two formats in a function call.
NOTE: If the type of the channel is frequency, users have to convert these 4 bytes into float format.

Syntax

```
public int PI_ReadValue(
  DWORD* o_dwPIValue
  BYTE* o_byChStatus
)
```

Parameter

*o_dwPIValue

[OUT] The byte array of the PI channel counter value.

*o_byChStatus

[OUT] The byte array of the channel status.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwPIValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ReadValue(o_dwPIValue, o_byChStatus)))
    printf("% d", iErrCode);
  else
  {
    for (iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
      printf("% d\n", o_dwPIValue[iIdx]);
  }
}
```

5.7.9. PI_ReadCntValue

Read PI value in double-word format. This method reads the all counter value of channels.
NOTE: If the channel is in the type of frequency. The value of the related channel of the o_dwCntValue will be 0, and the value of the related channels of o_byChStatus will indicate the type not support.

Syntax

```
public int PI_ReadValue(
  DWORD* o_dwCntValue
  BYTE* o_byChStatus
)
```

Parameter

*o_dwCntValue

[OUT] The unsigned long array of the PI channel counter value.

*o_byChStatus

[OUT] The byte array of the channel status.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
DWORD o_dwCntValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ReadCntValue(o_dwCntValue,
o_byChStatus)))
    printf("% d", iErrCode);
  else
  {
    for (iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
      printf("% d\n", o_dwCntValue[iIdx]);
  }
}
```

5.7.10. PI_ReadFreqValue

Readthe frequency value.This method reads the all frequency value of channels.
NOTE: If the channel is not in the type of frequency. The value of the related channel of the o_dwCntValue will be -1, and the value of the related channels of o_byChStatus will indicate the type not support.

Syntax

```
public int PI_ReadValue(
  float* o_fFreqValue
  BYTE* o_byChStatus
)
```

Parameter

*o_fFreqValue

[OUT] The float array of the PI channel frequency value

*o_byChStatus

[OUT] The byte array of the channel status

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
floato_fFreqValue[USBIO_PI_MAX_CHANNEL];
BYTE o_byChStatus[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ReadValue(o_fFreqValue, o_byChStatus)))
    printf(“% d”, iErrCode);
  else
  {
    for (iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx++)
      printf(“% f\n”, o_fFreqValue[iIdx]);
  }
}
```

5.7.11. PI_ReadBulkValue

Trigger reading bulk PI value (Fast acquire functionality).
When in callback operation, it will cause the performance in your callback function. Please reduce execute time in this callback function.
The detail of operation is described as follow. When call this API, the PI module operation will be changed from normal to fast acquire mode. In fast acquire mode, PI module follow the parameter of API setting to acquire data.
The API has block and non-block operation. In block operation, user's application needs to wait until API finishing all procedure. In contrast with block mode, non-block provides a flexible way for user. In non-block operation, user's application can proceed to own other code. To enable non-block operation, it is important to declare a callback function and pass it through last parameter. For block operation, just pass a NULL definition in last parameter.
Due to the USB 2.0 Full-speed transfer rate capability, the maximum sample rate is 10 KHz.

Syntax

```
public int PI_ReadBulkValue(
  BYTE i_byStartCh,
  BYTE i_byChTotal,
  DWORD i_dwSampleWidth,
  Float i_fSampleRate,
  DWORD i_dwBufferWidth,
  DWORD* o_dwDataBuffer,
  OnBulkValueFinishEvent i_CBFunc
)
```

Parameter

i_byStartCh

[IN]The starting acquire channel

i_byChTotal

[IN] The total channels to acquire

i_dwSampleWidth

[IN] The sampling width (ms)

i_fSampleRate

[IN] The sampling rate (Hz). 10KHz maximum.

i_dwBufferWidth

[IN] The width of the buffer for single channel

*o_dwDataBuffer

[OUT] The 2-dimension buffer array to store

i_CBFunc

[IN] Block operation – NULL

[IN] Non-block operation - The address of callback function.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
// To read 0~1 channel for 100ms in 5 KHz sample rate each channel in non-block
operation
// So we have the following variable declaration
#define SampleRate 5000
#define BufferWidth 500; // 5000(Hz) * 0.1(100ms)
DWORD m_dwBuffer[2][BufferWidth] ;

Void BulkFinishCallback(DWORD dwCount)
{
  // Callback function to handle data
}

Int main()
{
  m_usbIO = new ICPDAS_USBIO();
  if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
  {
    if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ReadBulkValue(0, 2, 100, SampleRate,
    BufferWidth m_dwBuffer, BulkFinishCallback)))
      printf("% d", iErrCode);

    while (1) { Sleep(1); }
  }
}
```

5.7.12. PI_SetTypeCode

The class has two overload methods for setting type code. One provides specifying channel to set, another for all channels. Please refer to user's manual for pulse input type code. These two overload methods are listed as following table and described in following section. The corresponding type code can be found in Appendix A.3.

Name of Methods
PI_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)
PI_SetTypeCode (BYTE *i_byTypeCodes)

5.7.12.1. PI_SetTypeCode (BYTEi_byChToSet, BYTEi_byTypeCode)

Set type code for specific channel. The type code can reference to Appendix A.3.

Syntax

```
public int PI_SetTypeCode (
  BYTE i_byChToSet,
  BYTE i_byTypeCode
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_byTypeCode

[IN] The type code for the specific channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTypeCode(0, 0x10)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.12.2. PI_SetTypeCode (BYTE *i_byTypeCodes)

Set type code for all channels. The type code can reference to Appendix A.3..

Syntax

```
public int PI_SetTypeCode (
  BYTE i_byTypeCodes
)
```

Parameter

i_byTypeCodes

[IN] The byte array of type code to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTypeCode[USBIO_PI_MAX_CHANNEL];
Int iIdx;
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
    m_byChTypeCode[iIdx] = 0x50;
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTypeCode(m_byChTypeCode)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.13. PI_ClearChCount

Clear channel count with clear mask. Each byte indicates 8 channels clear mask, set for channel clear. Ex: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_ClearChCount (
  BYTE* i_byClrMask
)
```

Parameter

i_byClrMask

[IN] The byte array of channel count clear mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChClrMask[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
    m_byChClrMask[iIdx] = 0x5A;
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearChCount(m_byChClrMask)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.14. PI_ClearSinglelChCount

Clear specific channel count.

Syntax

```
public int PI_ClearSingleChCount (
  BYTE i_byChToClr
)
```

Parameter

i_byChToClr

[IN] The channel index for clearing.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearSingleChCount(7)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.15. PI_ClearChStatus

Clear channel status with clear mask. Each byte indicates 8 channels clear mask, set for channel clear. Ex: Byte0 -> Channel0 ~ 7.

Syntax

```
public int PI_ClearChStatus (
  BYTE* i_byClrMask
)
```

Parameter

i_byClrMask

[IN] The byte array of channel status clear mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChClrMask[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
    m_byChClrMask[iIdx] = 0x5A;
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearChStatus(m_byChClrMask)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.16. PI_ClearSinglelChStatus

Clear specific channel status.

Syntax

```
public int PI_ClearSingleChStatus (
  BYTE i_byChToClr
)
```

Parameter

i_byChToClr

[IN] The channel index for clearing.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_ClearSingleChStatus(7)))
  printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.17. PI_SetTriggerMode

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels.

The trigger mode is shown as following table.

Trigger Mode	Code
Falling edge	0
Rising edge	1
Both edge	2 & 3

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetTriggerMode (BYTEi_byChToSet, BYTEi_byTypeCode)
PI_SetTriggerMode (BYTE *i_byTypeCodes)

5.7.17.1. PI_SetTriggerMode (BYTEi_byChToSet, BYTEi_byTypeCode)

Set trigger mode to specific channel.

Syntax

```
public int PI_SetTriggerMode(
  BYTE i_byChToSet,
  BYTE i_byTriggerMode
)
```

Parameter

i_byChToSet

[IN] The specific channel to set

i_byTriggerMode

[IN] The type code for the specific channel

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTriggerMode(0, 0x1)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.17.2. PI_SetTriggerMode (BYTE *i_byTypeCodes)

Set trigger mode to all channel.

Syntax

```
public int PI_SetTriggerMode (
  BYTE* i_byTriggerMode
)
```

Parameter

*i_byTriggerMode

[IN] The byte array of trigger mode to set.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChTriggerMode[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
m_byChTriggerMode[iIdx] = 0x2;
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetTriggerMode(m_byChTriggerMode)))
    printf(“% d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.18. PI_SetChIsolatedFlag

The class has two overload methods for setting channel isolated flag. One provides specifying channel to set, the other for all channels. Set 1 for setting channel to isolated. The parameter of the method for all channels is constructed in byte array for all channel isolated flag, ex: Byte0 -> Channel0 ~ 7.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetChIsolatedFlag (BYTEi_byChToSet, BOOL i_bChIsolatedFlag)
PI_SetChIsolatedFlag (BYTE *i_byChIsolatedFlags)

5.7.18.1. PI_SetChIsolatedFlag (BYTEi_byChToSet, BOOL i_bChIsolatedFlag)

Set channel isolated flag.

Syntax

```
public int PI_SetChIsolatedFlag(
  BYTE i_byChToSet,
  BOOL i_bChIsolatedFlag
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_bChIsolatedFlag

[IN] The isolated flag for the specific channel.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetChIsolatedFlag(5, 0x1)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.18.2. PI_SetChIsolatedFlag (BYTE *i_byChIsolatedFlags)

Set channel isolated flag to all channels.

Syntax

```
public int PI_SetChIsolatedFlag(
  BOOL i_byChIsolatedFlags
)
```

Parameter

i_ byChIsolatedFlags

[IN] The byte arrays ofchannel isolated flag.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byChIsolatedFlags[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
m_byChIsolatedFlag[iIdx] = 0x5a;
  if (ERR_NO_ERR != (iErrCode =
m_usbIO.PI_SetChIsolatedFlag(m_byChIsolatedFlag)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.19. PI_SetLPFilterEnable

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels. Set 1 to enable low-pass filter. The parameter of the method for all channels is a byte array for all channel enable mask, ex: Byte0 -> Channel0 ~ 7.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetLPFilterEnable (BYTEi_byChToSet, BOOLi_bLPFilterEnable)
PI_SetLPFilterEnable (BYTE *i_byLPFilterEnable)

5.7.19.1. PI_SetLPFilterEnable (BYTEi_byChToSet, BOOLi_bLPFilterEnable)

Set low-pass filter enable to specific channel.

Syntax

```
public int PI_SetLPFilterEnable(
  BYTE i_byChToSet,
  BOOL i_bLPFilterEnable
)
```

Parameter

i_byChToSet

[IN] The specific channel to set

i_bLPFilterEnable

[IN] The enable flag for the specific channel

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterEnable(5, 0x1)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.19.2. PI_SetLPFilterEnable (BYTE *i_byLPFilterEnable)

Set low-pass filter enable to all channel.

Syntax

```
public int PI_SetLPFilterEnable(
  BYTE i_byLPFilterEnable
)
```

Parameter

i_byLPFilterEnable

[IN] The byte array oflow-pass filter enable mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byLPFilterEnable[(USBIO_PI_MAX_CHANNEL + 7) / 8];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < (USBIO_PI_MAX_CHANNEL + 7) / 8; iIdx)
m_byLPFilterEnable[iIdx] = 0x5a;
  if (ERR_NO_ERR != (iErrCode =
m_usbIO.PI_SetLPFilterEnable(m_byLPFilterEnable)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.20. PI_SetLPFilterWidth

The class has two overload methods for setting trigger mode. One provides specifying channel to set, another for all channels. The low-pass filter width is for filtering noise or bouncing. The unit of the filter width is uS.

These two overload methods are listed as following table and described in following section.

Name of Methods
PI_SetLPFilterWidth (BYTEi_byChToSet, WORDi_wLPFilterWidth)
PI_SetLPFilterWidth (WORD *i_wLPFilterWidths)

5.7.20.1. PI_SetLPFilterWidth (BYTEi_byChToSet, WORDi_wLPFilterWidth)

Set low-pass filter width.

Syntax

```
public int PI_SetLPFilterWidth(
  BYTE i_byChToSet,
  BOOL i_wLPFilterWidth
)
```

Parameter

i_byChToSet

[IN] The specific channel to set.

i_wLPFilterWidth

[IN] The low-pass filter width. (uS)

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterWidth(5, 10000)))
    printf("% d", iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.7.20.2. PI_SetLPFilterWidth (WORD *i_wLPFilterWidths)

Set low-pass filter enable to all channel.

Syntax

```
public int PI_SetLPFilterEnable(
  BYTE i_byLPFilterEnable
)
```

Parameter

i_byLPFilterEnable

[IN] The byte array oflow-pass filter enable mask.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;
Byte m_byLPFilterWidth[USBIO_PI_MAX_CHANNEL];
Int iIdx;

m_usbIO = new ICPDAS_USBIO();
if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
{
  For(iIdx = 0; iIdx < USBIO_PI_MAX_CHANNEL; iIdx)
  m_byLPFilterEnable[iIdx] = 20000;
  if (ERR_NO_ERR != (iErrCode = m_usbIO.PI_SetLPFilterWidth(m_byLPFilterWidth)))
    printf(“% d”, iErrCode);
  iErrCode = m_usbIO.CloseDevice();
}
```

5.8. Other/Internal Methods

This section lists methods intended for debugging or internal use only. These functions are not recommended for standard applications and should be used with caution in special cases.

Name	Description
GetCurrentAccessObj	INTERNAL USE. DO NOT USE THIS METHOD.
SetNormalPktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
SetActivePktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
ClearActivePktBuffer	INTERNAL USE. DO NOT USE THIS METHOD.
GetActivePktByteArray	INTERNAL USE. DO NOT USE THIS METHOD.
SetNormalPktEvent	INTERNAL USE. DO NOT USE THIS METHOD.
IsDevMonitorThreadStop	INTERNAL USE. DO NOT USE THIS METHOD.
IsCommWithDevice	INTERNAL USE. DO NOT USE THIS METHOD.
GetLastCmdTime	INTERNAL USE. DO NOT USE THIS METHOD.
ToString	Show all information of device.

5.8.1. toString

Show all information of device.

Syntax

```
public void toString(
  void
)
```

Parameter

None.

Return Value

Error code.

Example

```
Int iErrCode
ICPDAS_USBIO m_usbIO;

if (ERR_NO_ERR == (iErrCode = m_usbIO.OpenDevice(USB22XX, 1)))
  m_usbIO.toString();
iErrCode = m_usbIO.CloseDevice();
```

Appendix

A. Type Code

This appendix lists the type codes used for configuring analog input, analog output, and pulse input functions, as well as channel status codes. These codes help developers correctly interpret and configure measurement parameters for the USB-2255-32/USB-2255-64 modules.

A.1. Analog Input

This table below lists the type codes for analog input channels, covering voltage, current, thermocouple (TC), and resistance temperature detector (RTD) input types. By applying the correct code, users can configure the module for accurate data acquisition.

Code	Input Type	Code	Input Type
0x00	-15 mV ~ +15 mV	0x17	Type L TC, -200 ~ +800°C
0x01	-50 mV ~ + 50 mV	0x18	Type M TC, -200 ~ +100°C
0x02	-100 mV ~ +100 mV	0x19	Type LDIN43710TC, -200 ~ +900°C
0x03	-500 mV ~ +500 mV	0x1A	0 ~ +20 mA
0x04	-1 V ~ +1 V	0x1B	-150 V ~ +150 V
0x05	-2.5 V ~ +2.5 V	0x1C	-50 V ~ +50 V
0x06	-20 mA ~ +20 mA	0x20	Pt 100, α=.00385, -100 ~ +100°C
0x07	+4 mA ~ +20 mA	0x21	Pt 100, α=.00385, 0 ~ +100°C
0x08	-10 V ~ +10 V	0x22	Pt 100, α=.00385, 0 ~ +200°C
0x09	-5 V ~ +5 V	0x23	Pt 100, α=.00385, 0 ~ +600°C
0x0A	-1 V ~ +1 V	0x24	Pt 100, α=.003916, -100 ~ +100°C
0x0B	-500 mV ~ +500 mV	0x25	Pt 100, α=.003916, 0 ~ +100°C
0x0C	-150 mV ~ +150 mV	0x26	Pt 100, α=.003916, 0 ~ +200°C

Code	Input Type	Code	Input Type
0x0D	-20 mA ~ +20 mA	0x27	Pt 100, α=.003916, 0 ~ +600°C
0x0E	Type J TC, -210 ~ +760°C	0x28	Nickel 120, -80 ~ +100°C
0x0F	Type K TC, -210 ~ +1372°C	0x29	Nickel 120, 0 ~ +100°C
0x10	Type T TC, -270 ~ +400°C	0x2A	Pt 1000, α=.00392, -200 ~ +600°C
0x11	Type E TC, -270 ~ +1000°C	0x2B	Cu 100, α=.00421, -20 ~ +150°C
0x12	Type R TC, 0 ~ +1768°C	0x2C	Cu 100, α=.00427, 0 ~ +200°C
0x13	Type S TC, 0 ~ +1768°C	0x2D	Cu 1000, α=.00421, -20 ~ +150°C
0x14	Type B TC, 0 ~ +1820°C	0x2E	Pt 100, α=.00385, -200 ~ +200°C
0x15	Type N TC, -270 ~ +1300°C	0x2F	Pt 100, α=.003916, -200 ~ +200°C
0x16	Type C TC, 0 ~ +2320°C		

A.2. Analog Output

This table below lists the type codes for analog output channels. The supported output formats include current output and voltage output across various ranges, allowing flexible configuration for industrial control and signal simulation.

Code	Input Type
0x30	0 ~ +20 mA
0x31	4 ~ +20 mA
0x32	0 V ~ +10 V
0x33	-10 V ~ +10 V
0x34	0 V ~ +5 V
0x35	-5 V ~ +5 V

A.3. Pulse Input

This table below lists the type codes for pulse input channels. The supported functions include counters, encoders, and frequency measurement, enabling precise monitoring of pulse-based signals in automation applications.

Code	Input Type
0x50	Up counter
0x51	Frequency
0x52	Counter with battery backup
0x53	Encoder
0x54	Up/Down counter
0x55	Pulse/Direction counter
0x56	AB phase
0x50	Up counter

A.4. Channel Status

This table below provides the status codes for input/output channels. Each code indicates the operating state of a channel, helping users quickly diagnose conditions such as over-range, under-range, or unsupported types.

Code	Input Type
0x00	Good
0x01	Over Range / Overflow
0x02	Under Range / Underflow
0x03	Open
0x04	Close
0x05	Type Not Supported

B. Error Code

This appendix provides the error codes returned by the USB-2255-32/USB-2255-64, the DEV-library, and the IO-library. Each error code corresponds to a specific fault condition. By referencing these codes, developers can identify issues more efficiently and apply the appropriate troubleshooting measures.

Constant	Value (Hex, Dec)	Description
ERR_NO_ERR	0x00000000, 0	The operation completed successfully.
ERR_DEV_ILLEGAL_FUNC	0x00000001, 1	The function is invalid.
ERR_ DEV_ILLEGAL_INDEX	0x00000002, 2	The index is invalid.
ERR_DEV_ILLEGAL_LENGTH	0x00000003, 3	The length is invalid.
ERR_DEV_ILLEGAL_PARAMETER	0x00000004, 4	The parameter is invalid.
ERR_DEV_ILLEGAL_MAPTABSIZE	0x00000005, 5	The size of mapping table is invalid.
ERR_DEV_ILLEGAL_MAPTABINDEX	0x00000006, 6	The index in mapping table is invalid.
ERR_ DEV_READONLY	0x00000007, 7	The index is read only.
ERR_ DEV_WRITEONLY	0x00000008, 8	The index is written only.
ERR_ DEV_BUFFERFULL	0x00000009, 9	The buffer in transceiver is full.
ERR_ DEV_LTTIMEOUT	0x0000000A, 10	The operation of large transfer has timeout.
ERR_ DEV_LTMODEFAIL	0x0000000B, 11	The current mode is not in large transfer mode.
ERR_ DEV_LTPKGLOST	0x0000000C, 12	The large transfer packet is lost.
ERR_ DEV_LTINDEXNOTMACH	0x0000000D, 13	The offset index is not match while operating in large transfer.
ERR_ DEV_LTNOTFINISH	0x0000000E, 14	Another large transfer is operating.
ERR_DEV_DO_RELATED_ERR	0x00004000 ~ 0x000047FFF	The digital output related error in this region.
ERR_DEV_DI_RELATED_ERR	0x00004800 ~ 0x00004FFF	The digital input related error in this region.

Constant	Value (Hex/Dec)	Description
ERR_DEV_AO_RELATED_ERR	0x00005000 ~ 0x000057FF	The analog output related error in this region.
ERR_DEV_AI_RELATED_ERR	0x00005800 ~ 0x00005FFF	The analog input related error in this region.
ERR_DEV_PO_RELATED_ERR	0x00006000 ~ 0x000067FF	The pulse output related error in this region.
ERR_DEV_PI_RELATED_ERR	0x00006800 ~ 0x00006FFF	The pulse input related error in this region.
ERR_USBDEV_INVALID_DEV	0x00010000, 65536	The handle of device is invalid.
ERR_USBDEV_DEV_OPENED	0x00010001, 65537	The device has been opened by class library.
ERR_USBDEV_DEVNOTEXISTS	0x00010002, 65538	The class library cannot find the device.
ERR_USBDEV_GETDEVINFO	0x00010003, 65539	An error was made to scan device.
ERR_USBDEV_ERROR_PKTSIZE	0x00010004, 65540	The packet size is invalid.
ERR_USBDEV_ERROR_WRITEFILE	0x00010004, 65541	An error occurs while writing packet to module.
ERR_USBIO_COMM_TIMEOUT	0x00010100, 65792	The communication between computer and device has been timeout.
ERR_USBIO_DEV_OPENED	0x00010101, 65793	The device has been opened by class library.
ERR_USBIO_DEV_NOTOPEN	0x00010102, 65794	The device has not opened for operating.
ERR_USBIO_INVALID_RESP	0x00010103, 65795	The data returned by device is invalid.
ERR_USBIO_IO_NOTSUPPORT	0x00010104, 65796	The method is not supported.
ERR_USBIO_PARA_ERROR	0x00010105, 65797	The parameter of method is invalid.
ERR_USBIO_BULKVALUE_ERR	0x00010106, 65798	An error occurs while getting bulk value.
ERR_USBIO_GETDEVINFO	0x00010107, 65799	An error occurs while getting device information while device opening procedure.

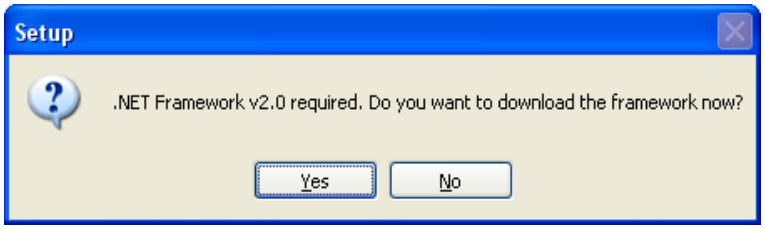
Constant	Value (Hex/Dec)	Description
ERR_USBDEV_INVALID_DEV	0x00010000, 65536	The handle of device is invalid.
ERR_USBDEV_DEV_OPENED	0x00010001, 65537	The device has been opened by class library.
ERR_USBDEV_DEVNOTEXISTS	0x00010002, 65538	The class library cannot find the device.
ERR_USBDEV_GETDEVINFO	0x00010003, 65539	An error was made to scan device.
ERR_USBDEV_ERROR_PKTSIZE	0x00010004, 65540	The packet size is invalid.
ERR_USBDEV_ERROR_WRITEFILE	0x00010004, 65541	An error occurs while writing packet to module.
ERR_USBIO_COMM_TIMEOUT	0x00010100, 65792	The communication between computer and device has been timeout.
ERR_USBIO_DEV_OPENED	0x00010101, 65793	The device has been opened by class library.
ERR_USBIO_DEV_NOTOPEN	0x00010102, 65794	The device has not opened for operating.
ERR_USBIO_INVALID_RESP	0x00010103, 65795	The data returned by device is invalid.
ERR_USBIO_IO_NOTSUPPORT	0x00010104, 65796	The method is not supported.
ERR_USBIO_PARA_ERROR	0x00010105, 65797	The parameter of method is invalid.
ERR_USBIO_BULKVALUE_ERR	0x00010106, 65798	An error occurs while getting bulk value.
ERR_USBIO_GETDEVINFO	0x00010107, 65799	An error occurs while getting device information while device opening procedure.

C. Troubleshooting

This appendix provides troubleshooting guidelines for common issues encountered when installing or operating the USB-2255-32/USB-2255-64. The listed problems, possible causes, and recommended solutions help users quickly identify and resolve system errors.

C.1. Unable to Install ICP DAS USB I/O Package

During installation, the system shows an error message indicating that the installation cannot proceed.



Cause

The ICP DAS USB I/O package requires .NET Framework v2.0. If this framework is not installed, the package will prompt for installation.

Solution

Click “Yes” when prompted to automatically download and install .NET Framework v2.0 via the internet.
If internet access is unavailable, install .NET Framework v2.0 manually from the folder “net_framework” located in the root directory of the installation CD.

C.2. Timeout Error Code (65792) When Accessing USB I/O

A timeout error code (65792) occurs when the system accesses the USB I/O module.

Causes

The USB module is connected to a USB hub instead of a local USB port on the computer, which increases communication latency.
The module may have malfunctioned due to unknown reasons. Refer to the LED Indicators section for diagnostic information.

Solution

Increase the communication timeout setting to prevent error occurrences.
Connect the USB module directly to a local USB port on the computer instead of using a USB hub.
If the issue persists, check the LED indicators for hardware failure diagnosis.

D. FAQ

This appendix provides frequently asked questions (FAQ) and answers to common issues encountered when using the USB-2255/USB-2200 series modules.

D.1. ICPDAS_USBIO_dotNET DLL in Visual Studio

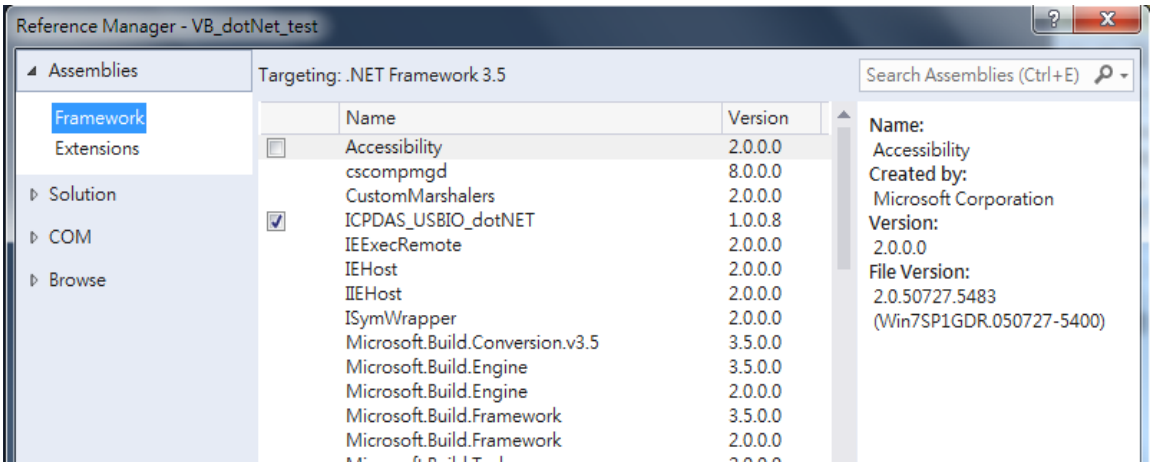
This FAQ explains how to locate the ICPDAS_USBIO_dotNET DLL in Visual Studio and add it to a project reference.

Q:

How to find ICPDAS_USBIO_dotNET DLL in Visual Studio 2013?

A:

Click Project → Reference → Assemblies → Framework → ICPDAS_USBIO_dotNET.



D.2. Build Project with DLL

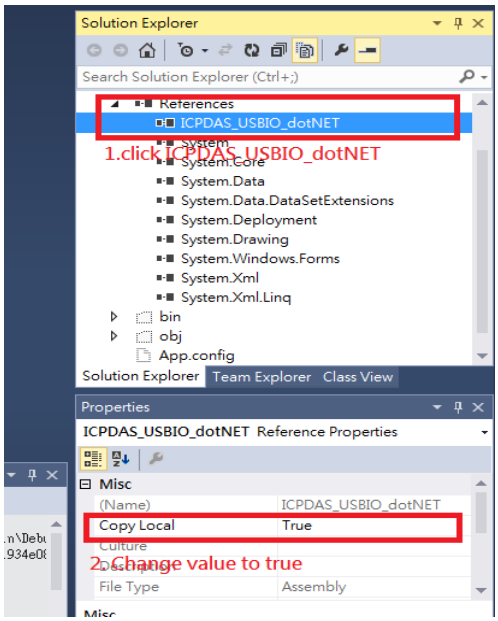
This FAQ describes the steps required to correctly build a project with the ICPDAS USB I/O DLL, including dependency handling for C# applications.

Q:

How to build a project with ICPDAS_USBIO_dotNET DLL in Visual Studio?

A:

1. Set the DLL property Copy Local = True.
2. For C# projects, copy ICPDAS_USBIO.dll and USBDEV_LIB.dll into the executable folder.

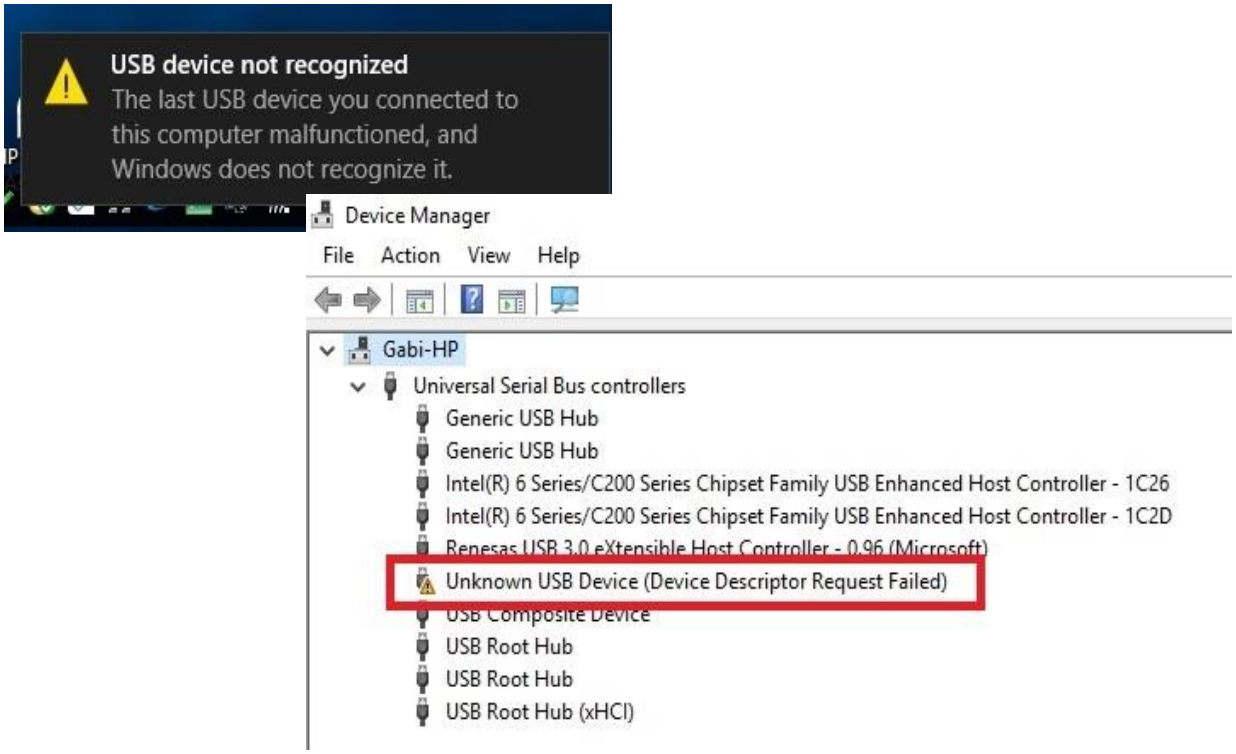


D.3. Device Not Recognized

This FAQ outlines the possible reasons why the USB-2200 module may not be recognized or may fail to operate, with emphasis on power supply reliability.

Q:

The USB-2200 module cannot be recognized or does not work normally. What should I do?



A:

- The issue may be caused by insufficient power supply, often when using:
  1. Front USB ports of a desktop PC
  2. Notebook powered only by battery
  3. USB hub without external power supply
- To improve stability, use a secondary DC power input (+10 ~ +30 VDC).

D.4. Safety Enable and Safety Value

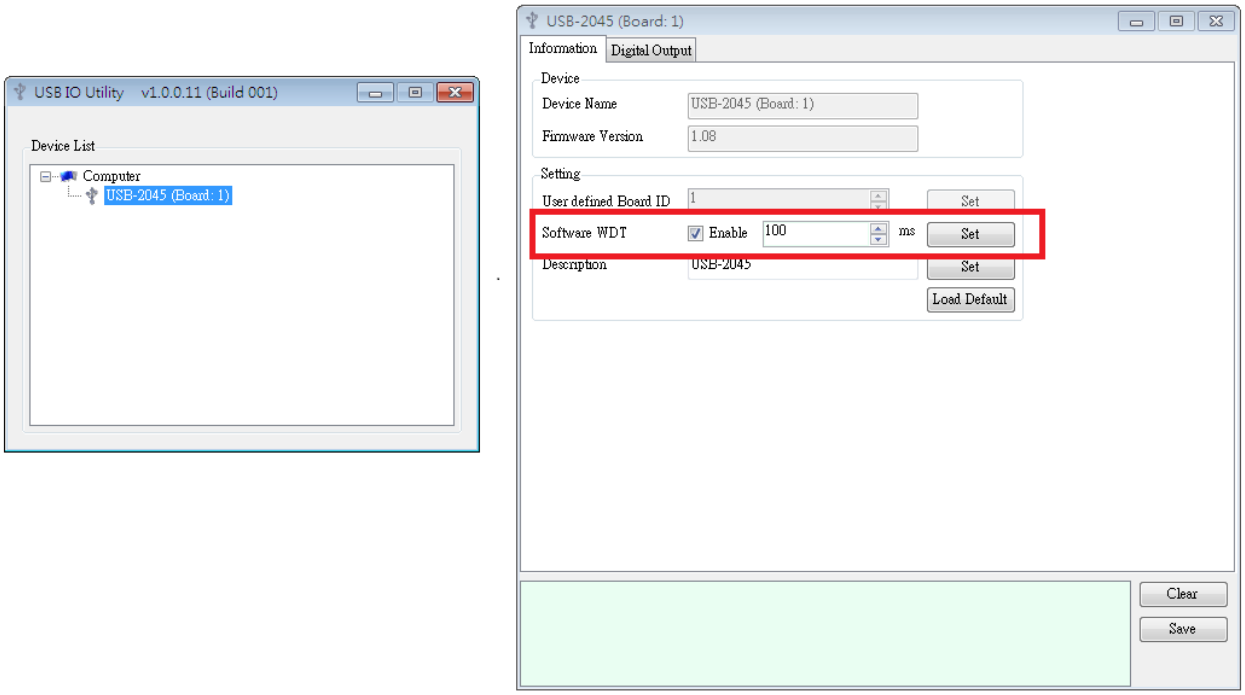
This FAQ explains how to enable the safety mechanism and configure the safety value used when communication between host and module is lost.

Q:

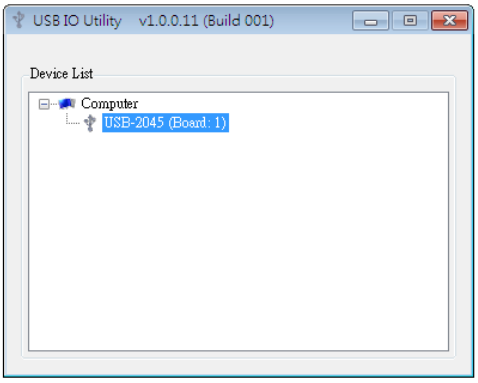
How to set the Safety Enable and Safety Value?

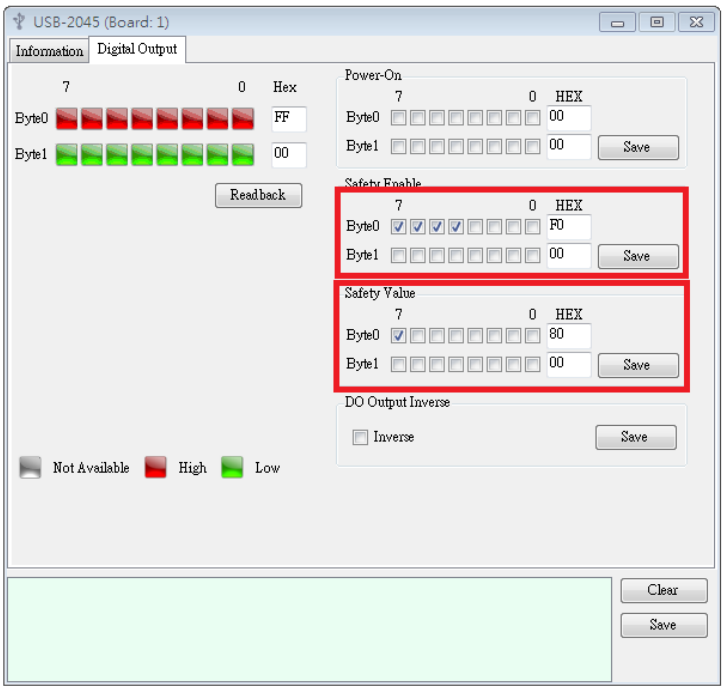
A:

1. Enable the Software Watchdog Timer (WDT) in the Information Page.

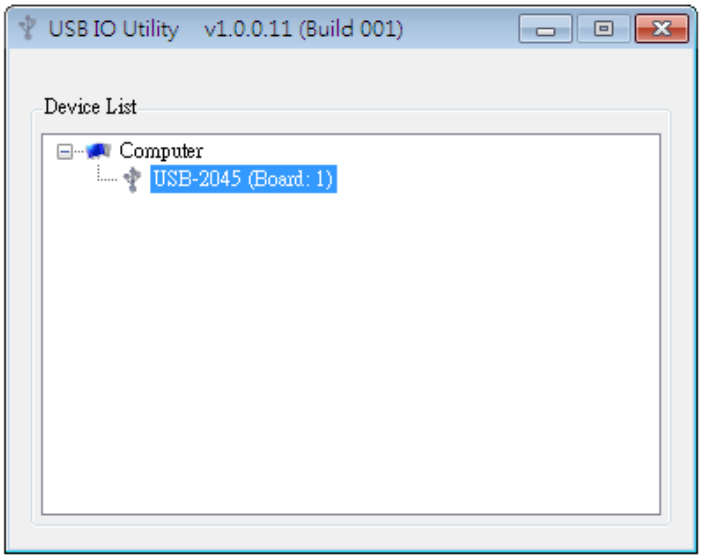


2. Set Safety Enable and define Safety Value (ON/OFF). Remember to click Save.

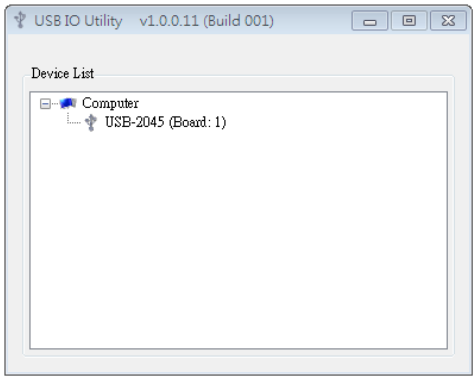


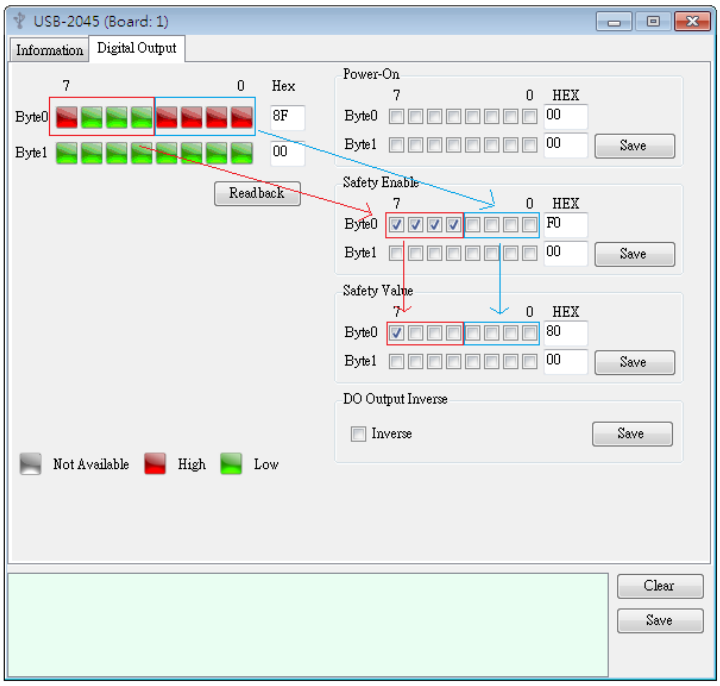


3. Close the Utility I/O Control page to simulate communication fault.



4. Re-open the Utility after timeout; outputs will switch to the Safety Value.





5. If Safety Enable is disabled, output values will remain unchanged.

D.5. Disable USB AutoPlay/AutoRun

This FAQ provides step-by-step instructions for disabling USB AutoPlay/AutoRun in different Windows versions.

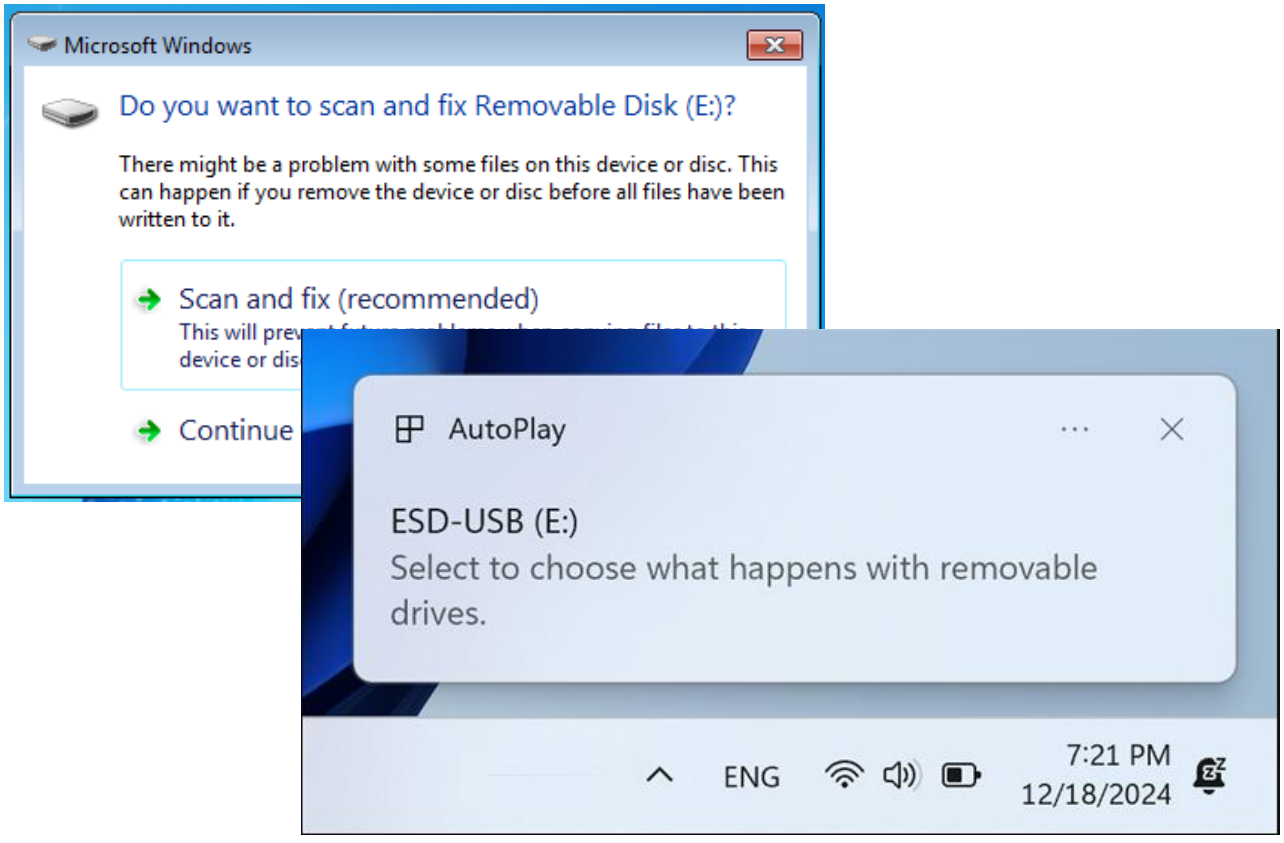
Q:

How to disable USB AutoPlay/AutoRun when the module is plugged in?

A:

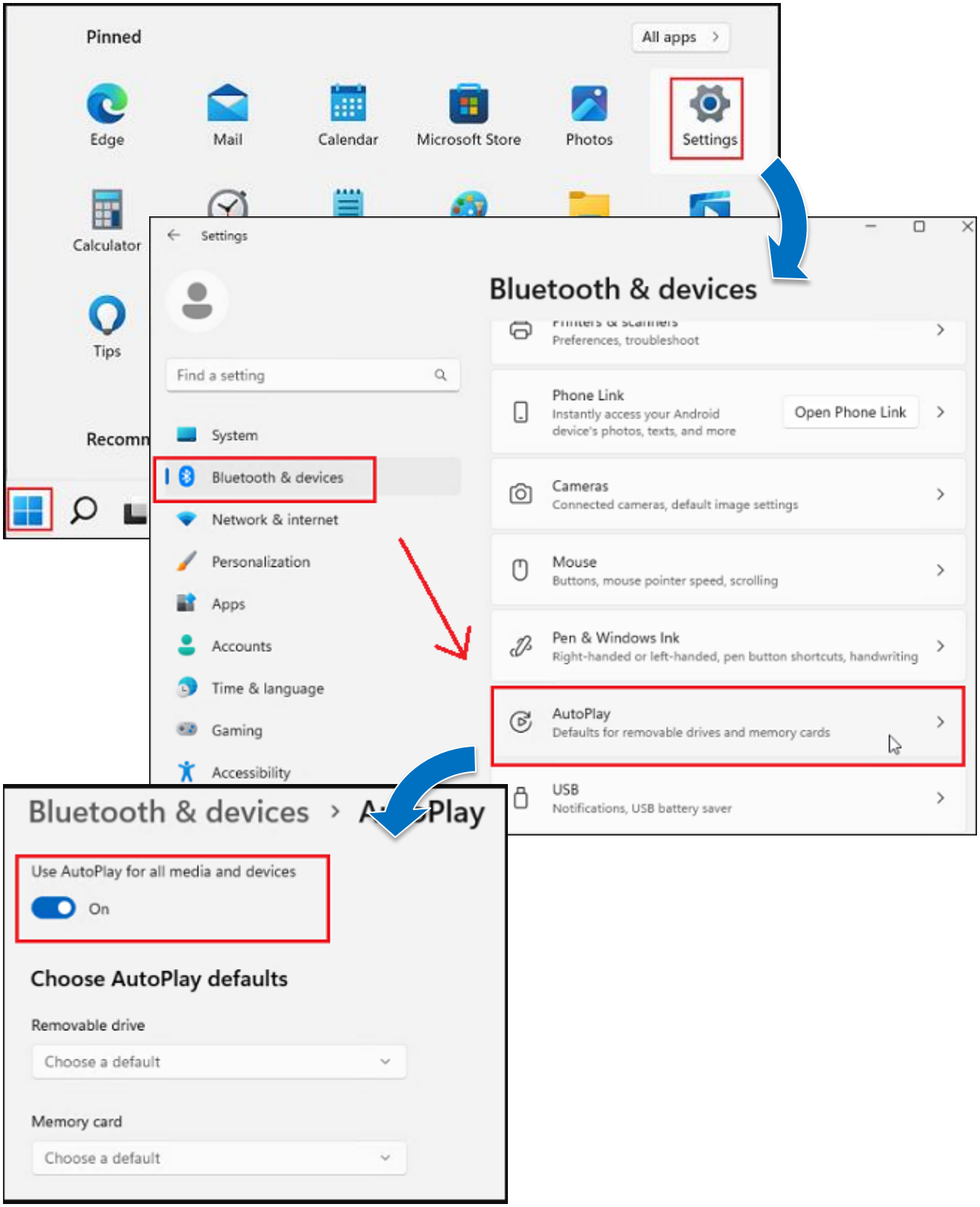
The steps depend on your Windows operating system. Please refer to the following appendix subsections for details:

- Appendix D.5.1 Windows 11
- Appendix D.5.2 Windows 10
- Appendix D.5.3 Windows Vista / 7 / Server 2008
- Appendix D.5.4 Windows XP / Server 2003 / 2000



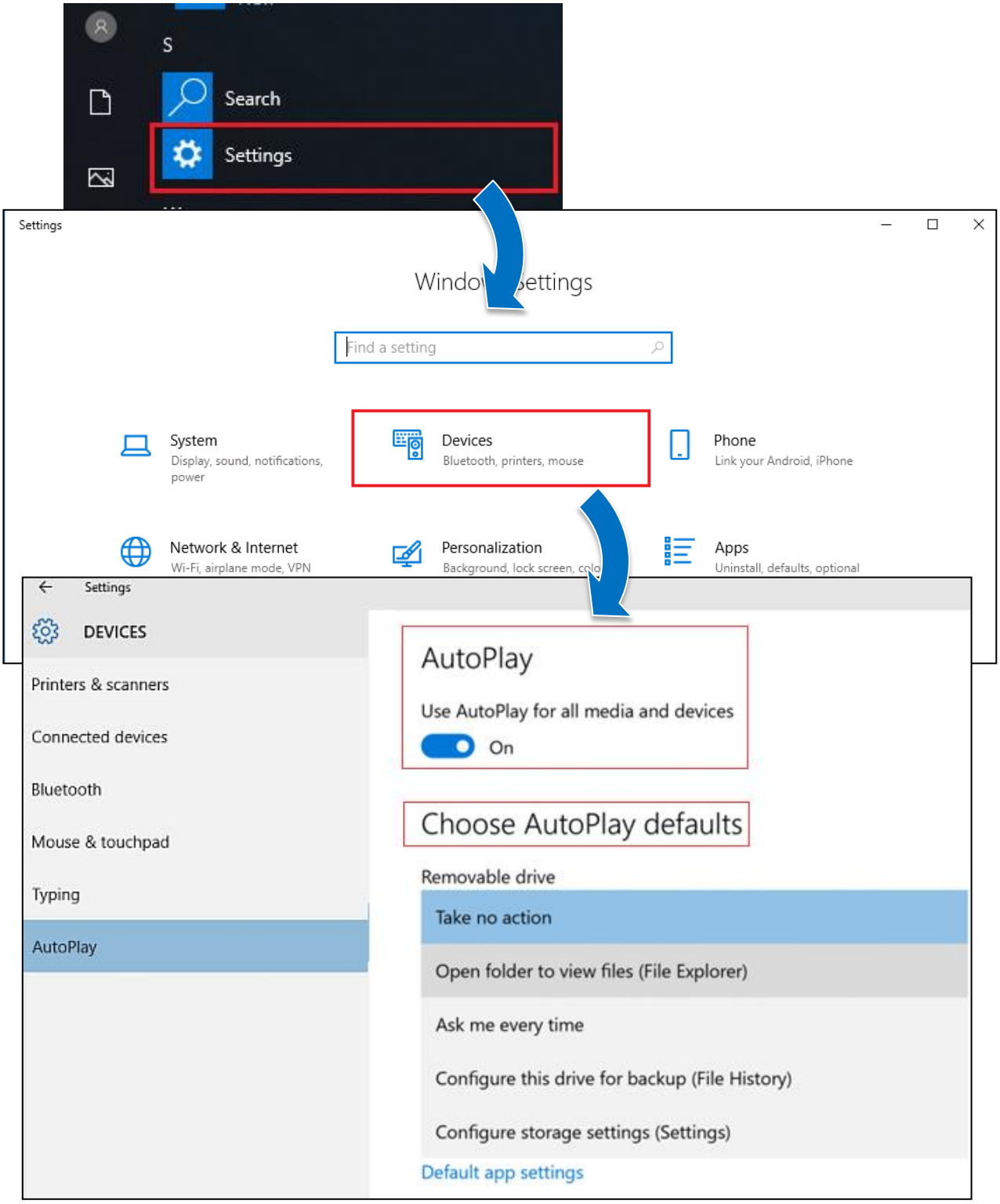
D.5.1. Windows 11

1. **Open** Settings → Bluetooth & Devices → AutoPlay**.**
2. **Toggle AutoPlay on/off and configure default actions.**



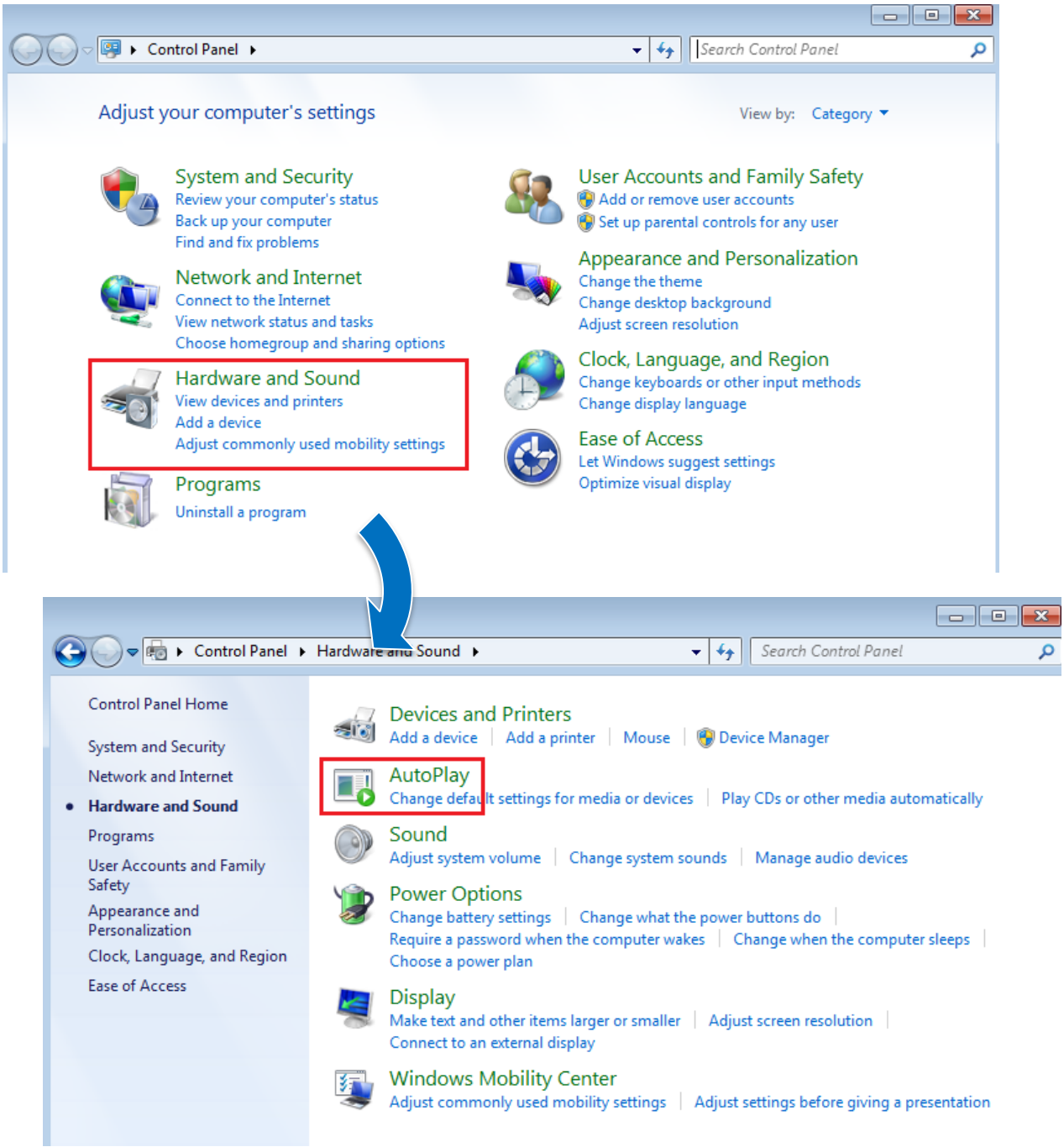
D.5.2. Windows 10

1. **Go to** Start → Settings → Devices → AutoPlay**.**
2. **Enable/disable AutoPlay using the toggle switch.**

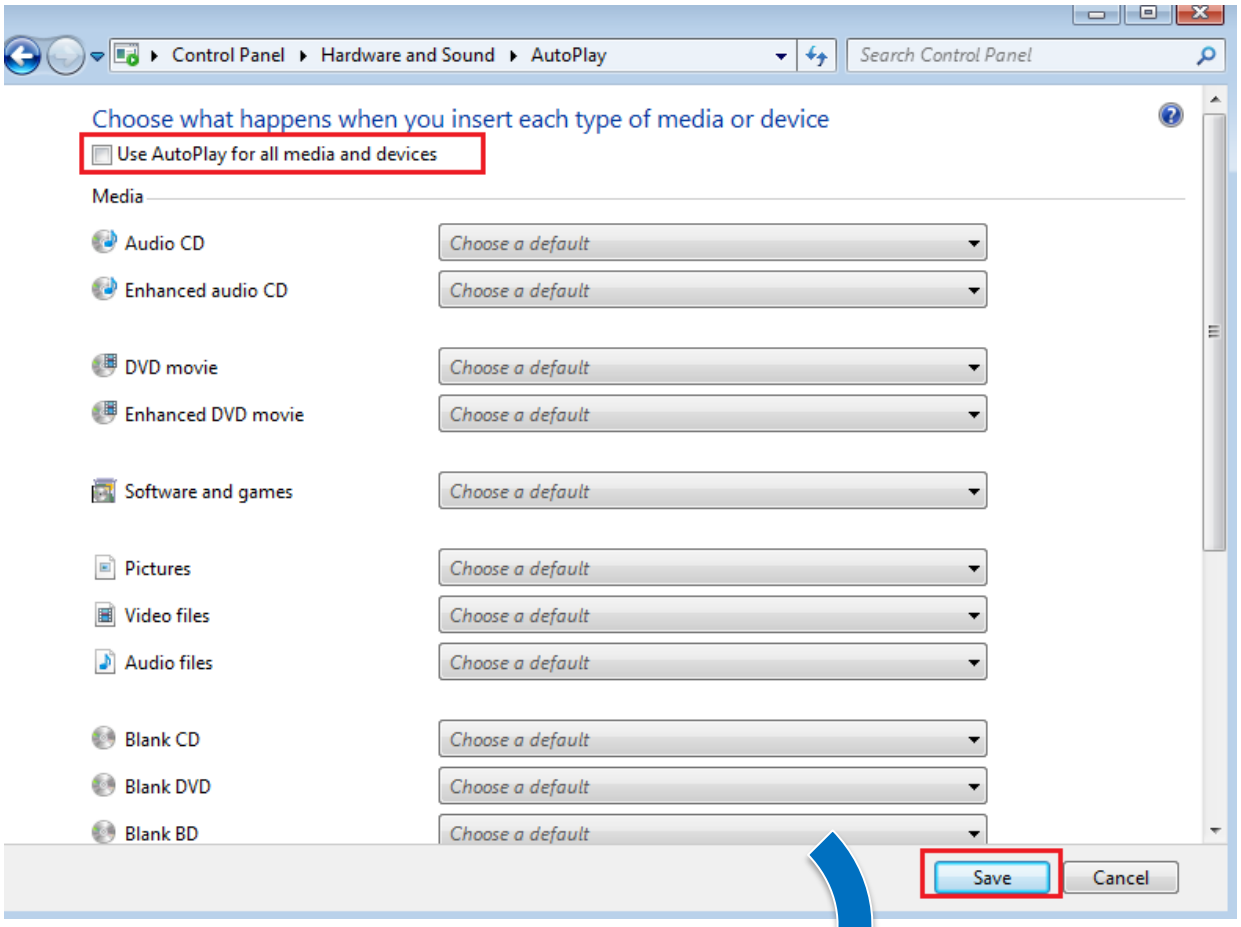


D.5.3. Windows Vista / 7 / Server 2008

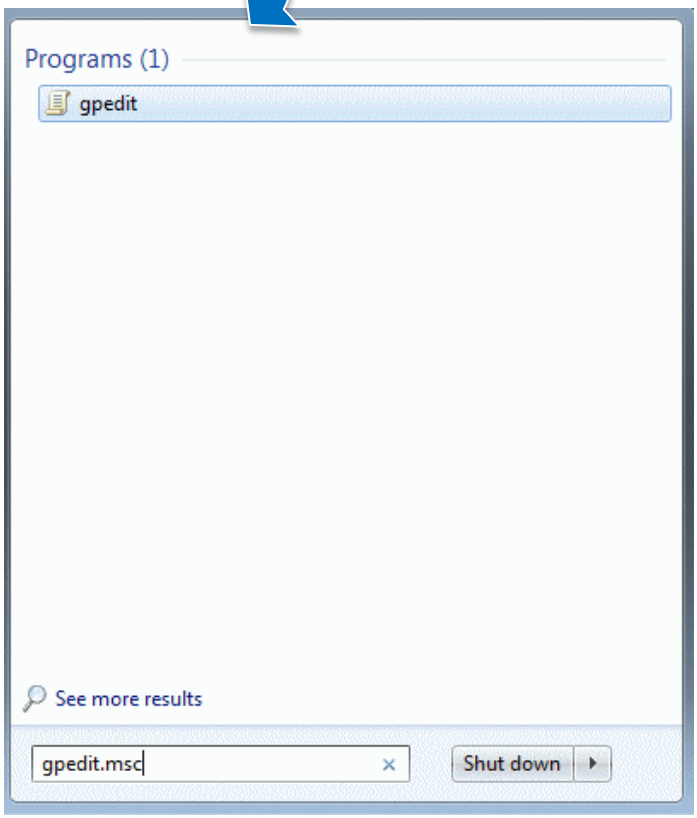
1. Open **Control Panel → Hardware and Sound → AutoPlay**.



2. Uncheck **Use AutoPlay for all media and devices**, then click **Save**.

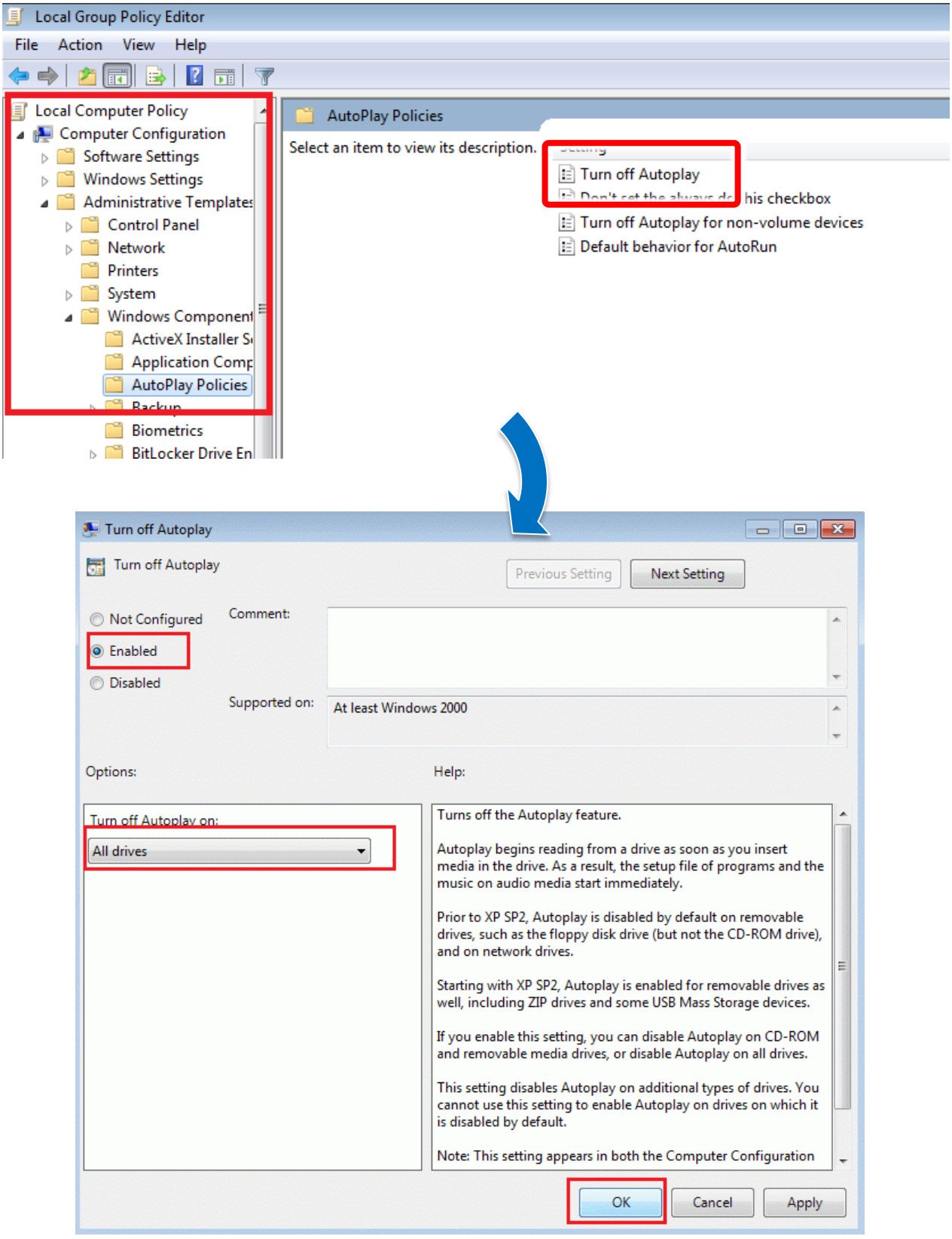


3. Open **Start → Run**, type `gpedit.msc`, press **Enter**.

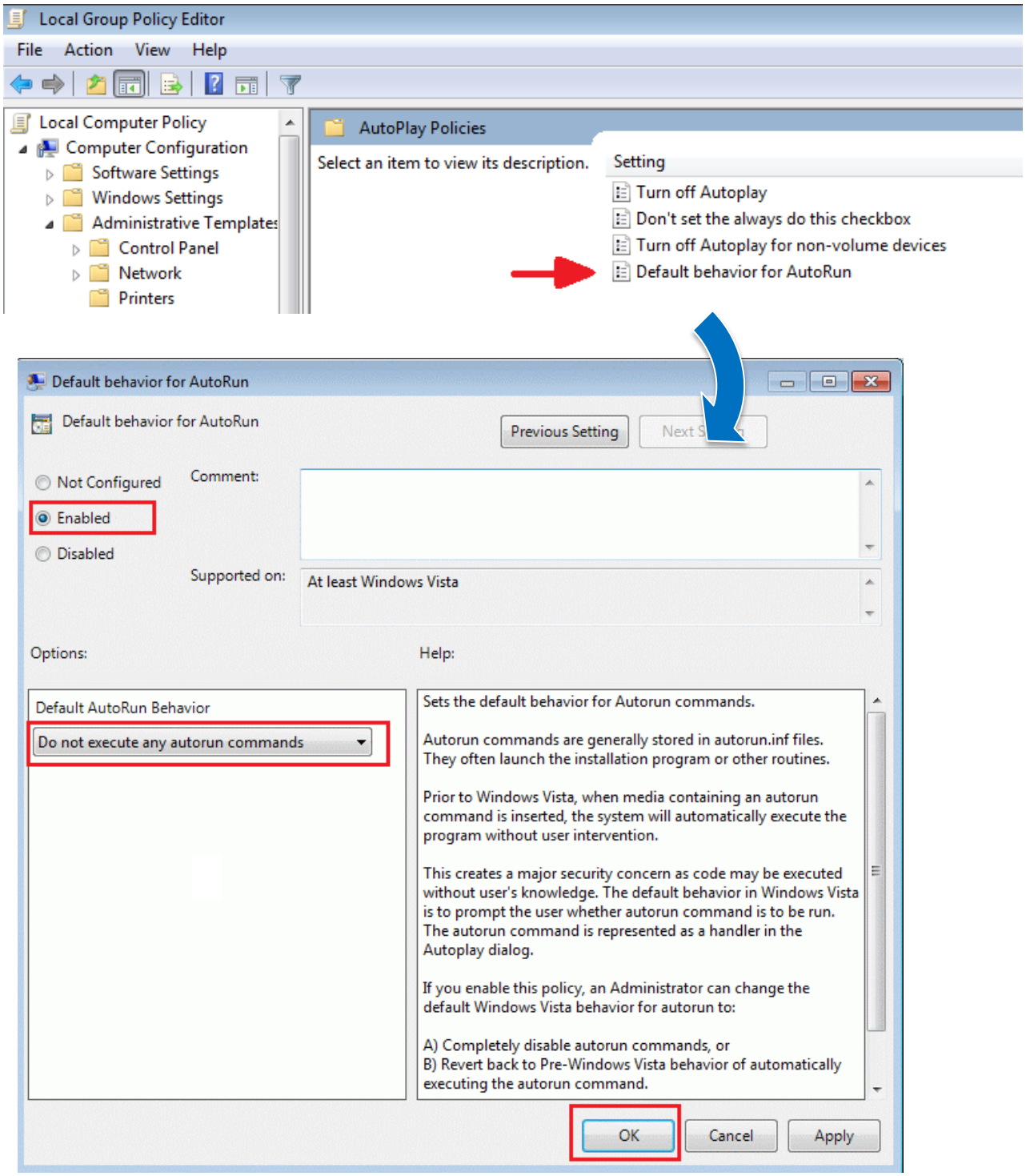


4. Navigate to **Computer Configuration → Administrative Templates → Windows Components → AutoPlay Policies**.

5. Click **Turn off AutoPlay**, set to **Enabled**, apply to **All drives**, click **OK**.



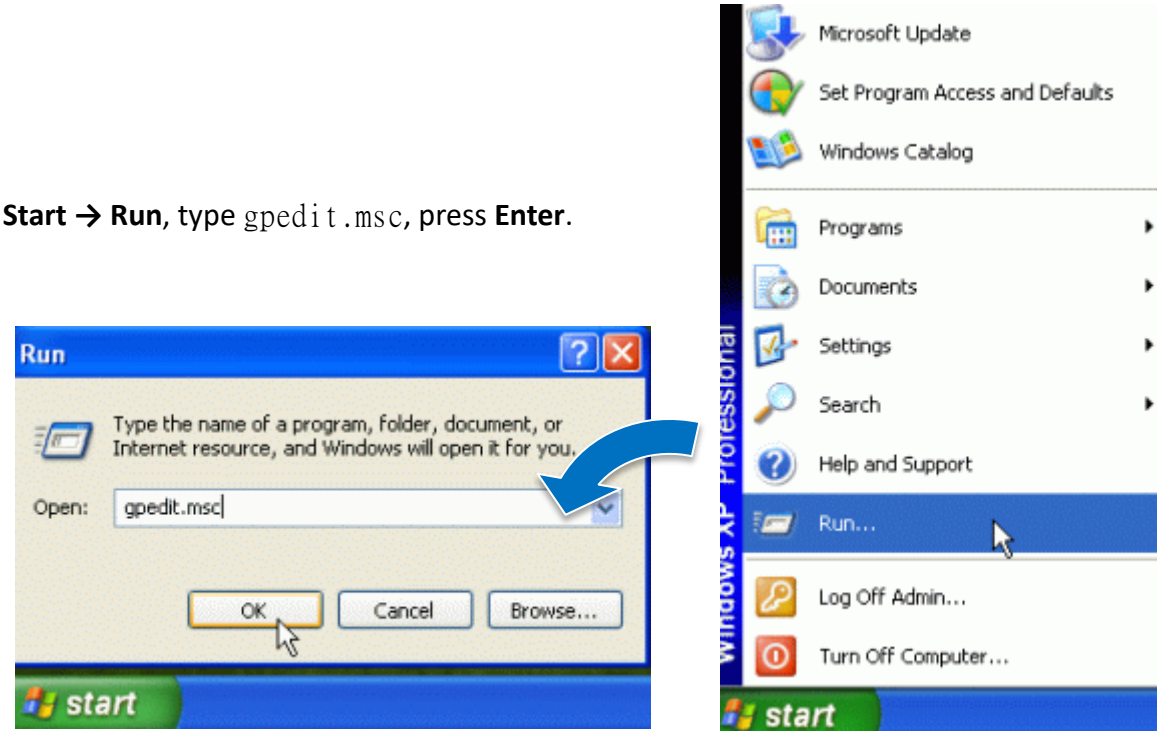
6. Open **Set the default behavior for AutoRun**, set to **Enabled**, choose **Do not execute any auto run commands**, click **OK**.



7. Close the Group Policy Editor and restart the computer.

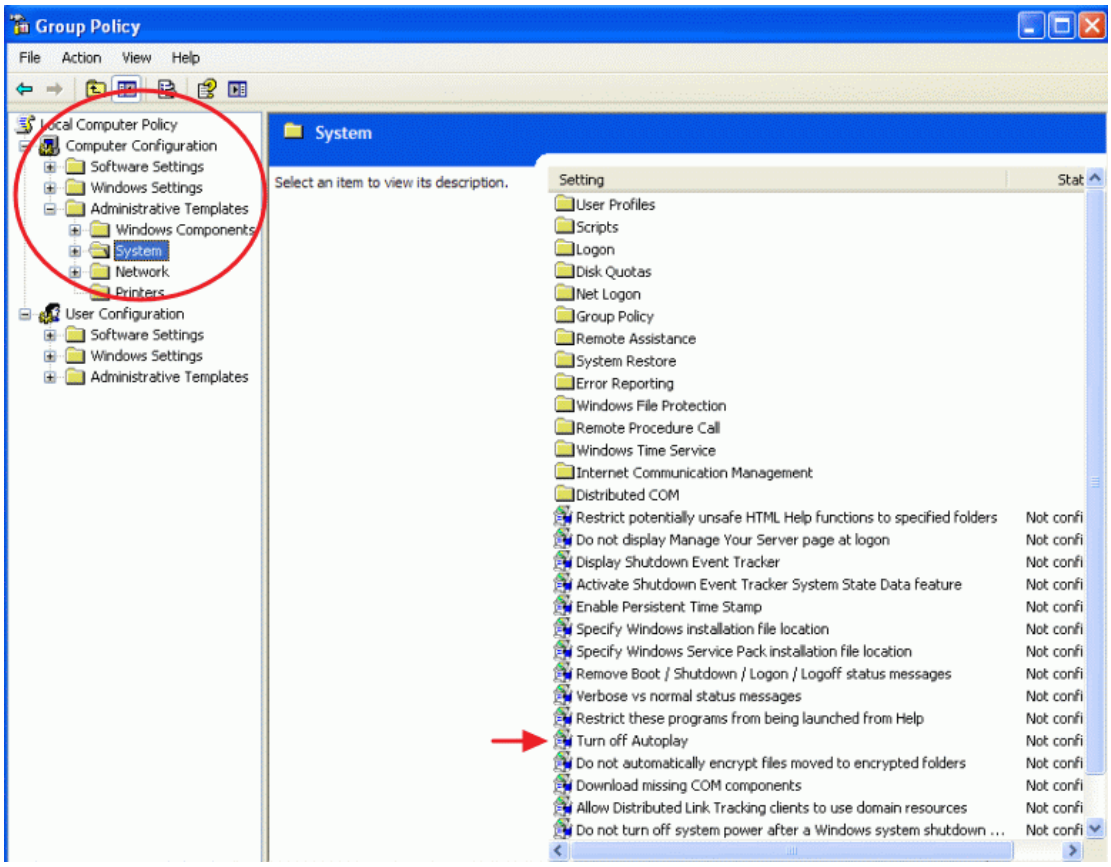
D.5.4. Windows XP / Server 2003 / 2000

1. Open **Start → Run**, type `gpedit.msc`, press **Enter**.

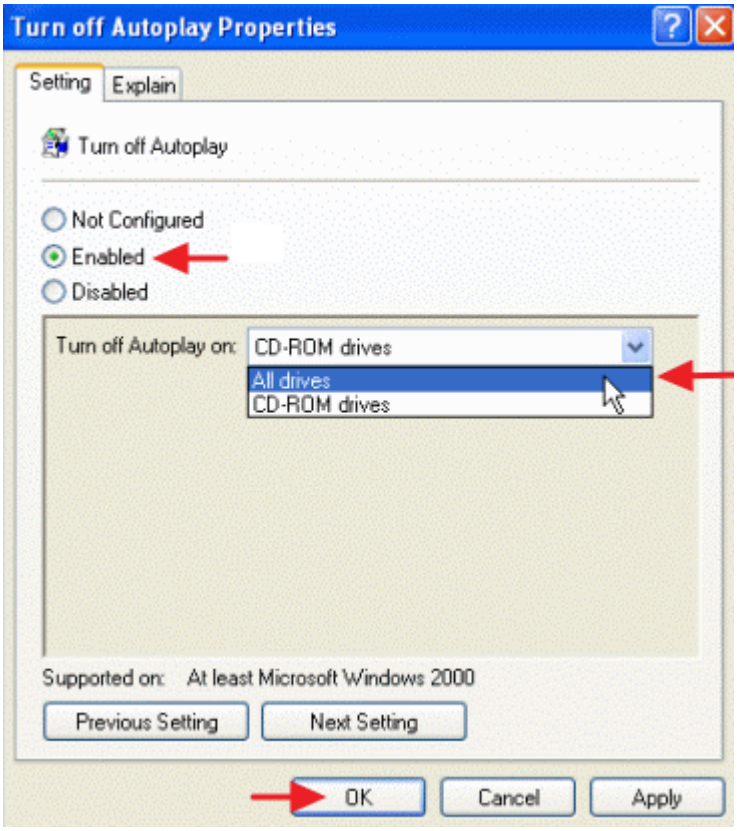


2. Navigate to **Computer Configuration → Administrative Templates → System → AutoPlay Policies**.

3. Click **Turn off AutoPlay**.

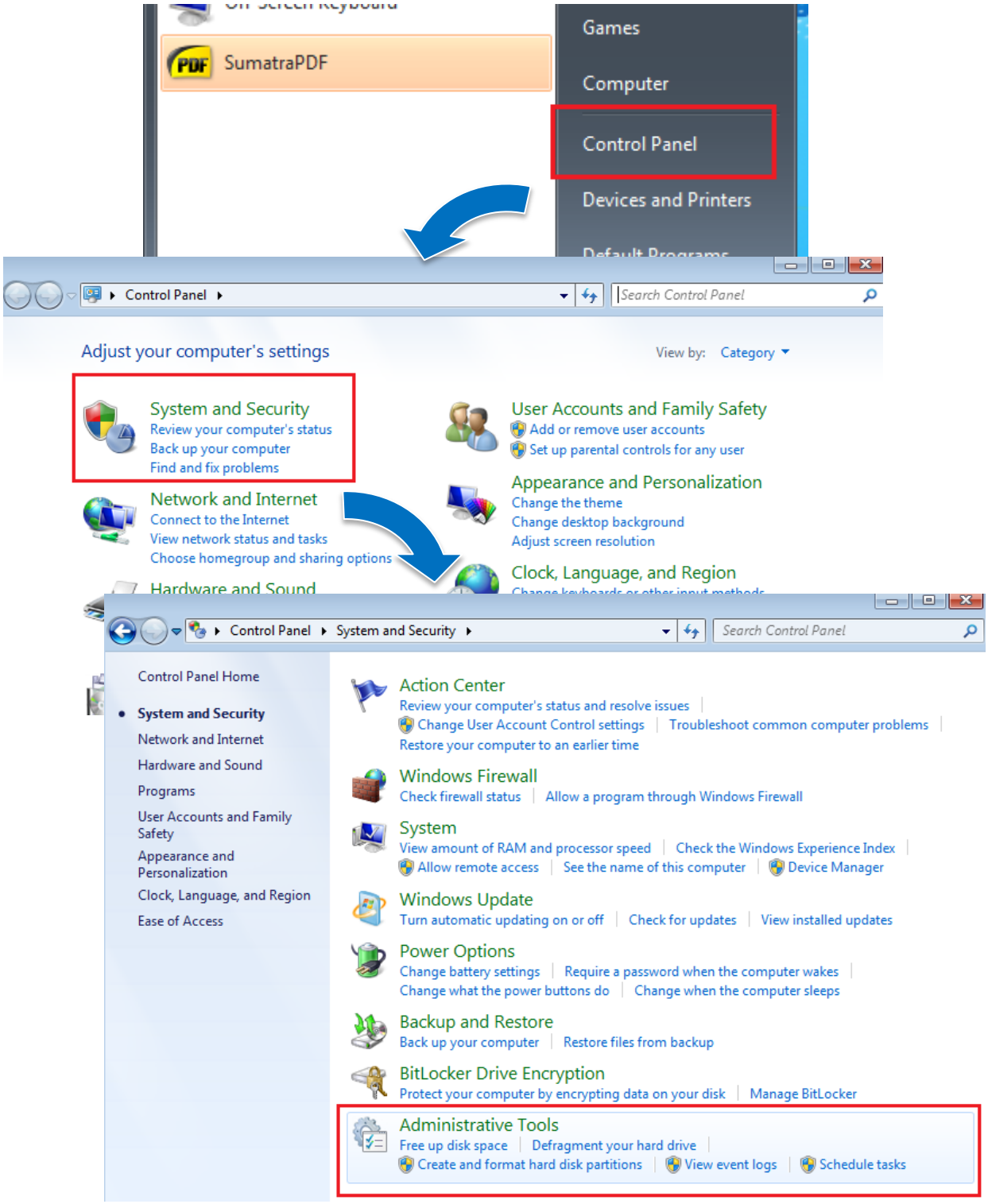


4. Set to **Enabled**, apply to **All drives**, click **OK**.

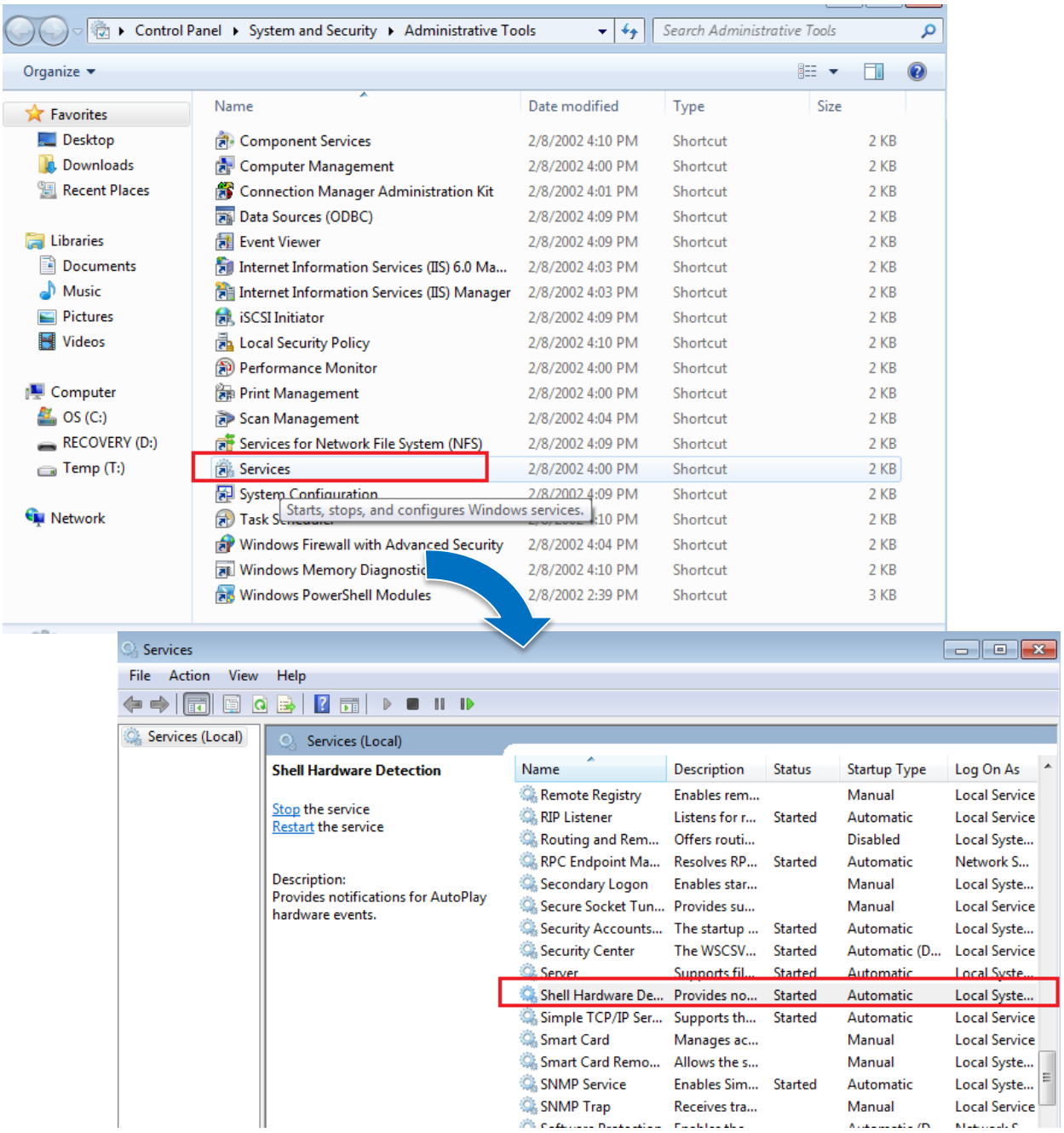


Alternative Method (not recommended)

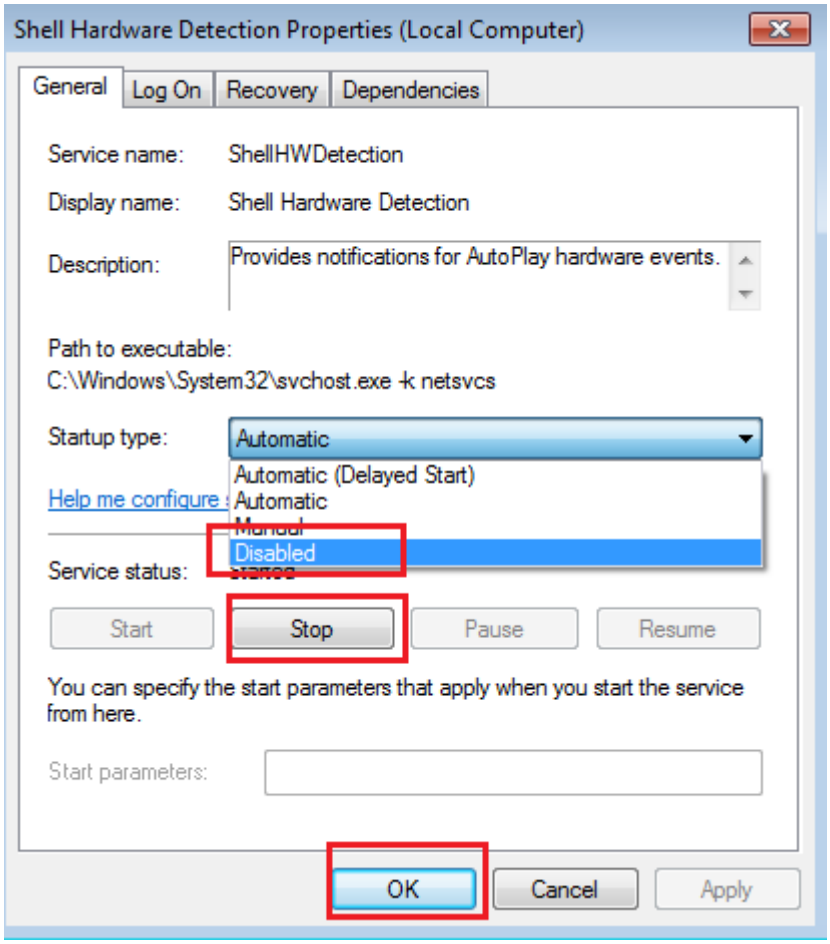
1. Open **Control Panel → System and Security → Administrative Tools**.



2. Navigate to **Services → Shell Hardware Detection**.



3. Set to **Disabled**, and stop the service.



E. Product Comparison

This appendix provides a brief comparison between the USB-2200 series and the USB-2000 series. The comparison highlights differences in hardware specifications, connectivity, power supply, speed, and software support, assisting users in selecting the appropriate product for their applications.

Item	USB-2000	USB-2200
USB Connector	USB type B	IN: Type A to type C OUT: Type C to Type C
Daisy-Chain Wiring	-	Yes (USB)
Lockable Cable	-	Yes
Protocol	USB HID	Mass Storage
Driver	Built-in Windows /Linux	
Power Supply	USB port	USB port , +10 ~ +30 VDC
Speed	12 Mbit/s (USB 2.0 Full-Speed)	480 Mbit/s (USB 2.0 High-Speed)
Dimensions (W x L x H, mm)	33 x 78 x 107 31 x 129 x 147	31 x 166 x 129 56 x 180 x 144
Utility	USB I/O Utility	
SDK/Sample	VB, C++, C#.Net, VB.Net , LabVIEW, Python, Linux driver	
Performance (Polling, Op0 – Read)	200Hz (5 ms)	1K ~ 5KHz
Performance (Polling, Op0 – Write)	200Hz (5 ms)	1K ~ 5KHz

F. Revision History

This appendix provides the revision history of the USB-2255-32/USB-2255-64 series user manual.

The table below shows the revision history.

Revision	Date	Description
1.0.1	August 2025	Initial issue