# USB-2000 Series
# Software Manual

User Manual

## Warranty

All products manufactured by ICP DAS are warranted against defective materials for a period of one year from the date of delivery to the original purchaser.

## Warning

ICP DAS assume no liability for damages consequent to 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

## Trademark

The names used for identification only may be registered trademarks of their respective companies.

# Table of Contents

# 1.USB-2000 Series Linux Driver Installation

The USB-2000 Series can be used in Linux O.S, when user uses USB-2000 Series on Linux OS, The Linux OS will detect and install the device driver automatically.

# 2.USB-2000 Series Static Library Function Description

The static library is the collection of function calls of the USB-2000 Series for Linux system. The application structure is presented as below figure “Figure 2-1”. The user application program developed by C (C++) language can call library “libUSBIO.a” for USB-2000 Series in user mode. And then static library will call the module command to access the hardware system.

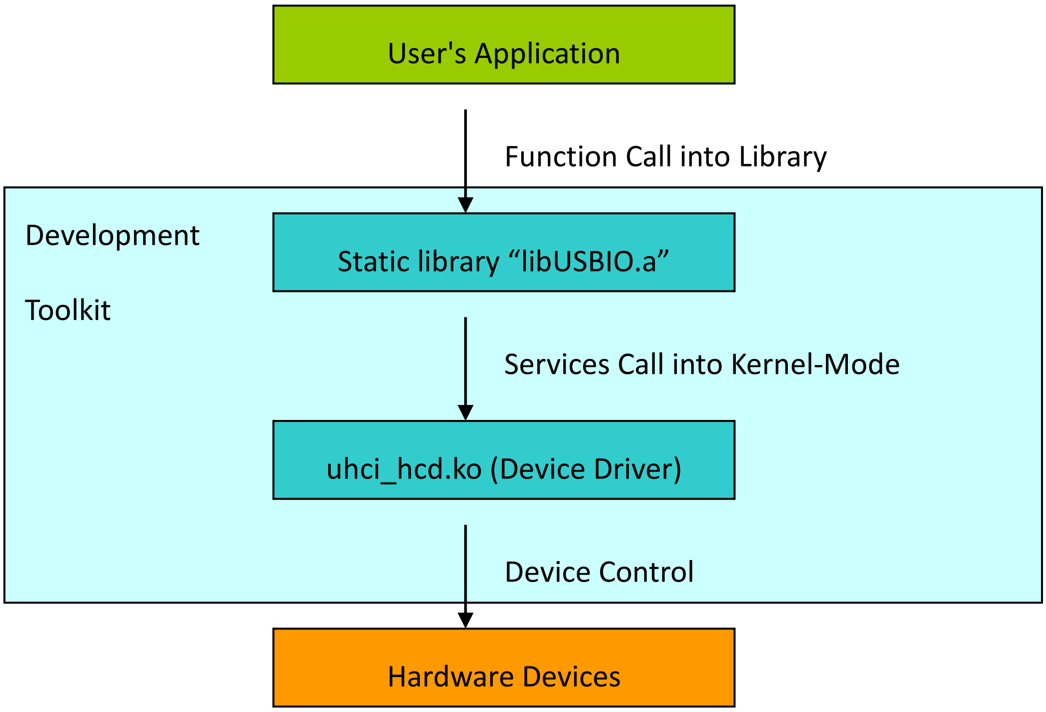

User's Application
Function Call into Library
Development
Toolkit
Static library “libUSBIO.a”
Services Call into Kernel-Mode
uhci_hcd.ko (Device Driver)
Device Control
Hardware Devices

Figure 2-1

## 2.1 Table of ErrorCode and ErrorString

Table 2.1

| Error code | Error ID | Error String |
|---|---|---|
| 0 | ERR_NO_ERR | OK (No error) |
| 0x10000 | ERR_USBDEV_INVALID_DEV | The handle of device is Invalid |
| 0x10001 | ERR_USBDEV_DEV_OPENED | The device has been opened by class library. |
| 0x10002 | ERR_USBDEV_DEVNOTEXISTS | The class library cannot find the device. |
| 0x10003 | ERR_USBDEV_GETDEVINFO | An error was made to scan device. |
| 0x10004 | ERR_USBDEV_ERROR_PKTSIZE | The packet size is invalid. |
| 0x10005 | ERR_USBDEV_ERROR_WRITEFILE | An error occurs while writing packet to module. |
| 0x10006 | ERR_USBDEV_ERROR_OPENFILE | Open device file error |
| 0x10007 | ERR_USBDEV_ERROR_CreateRxThread | Creating Rx thread error |
| 0x10008 | ERR_USBDEV_ERROR_RestartRxThread | Restarting Rx thread error. |
| 0x10100 | ERR_USBIO_COMM_TIMEOU T | The communication between computer and device has been timeout. |
| 0x10101 | ERR_USBIO_DEV_OPENED | The device has been opened by class |

| | | |
|---|---|---|
| | | library. |
| 0x10102 | ERR_USBIO_DEV_NOTOPEN | The device has not opened for operating. |
| 0x10103 | ERR_USBIO_INVALID_RESP | The data returned by device is invalid. |
| 0x10104 | ERR_USBIO_IO_NOTSUPPORT | The method is not supported. |
| 0x10105 | ERR_USBIO_PARA_ERROR | The parameter of method is invalid. |
| 0x10106 | ERR_USBIO_BULKVALUE_ERR | An error occurs while getting bulk value. |
| 0x10107 | ERR_USBIO_GETDEVINFO | An error occur while getting device information while device opening procedure. |

Table 2.2

## 2.2 System Functions

### 2.2.1 USBIO_GetLibraryVersion

- **Description:**

  To get USB-2000 Series lib version.

- **Syntax:**

  char *USBIO_GetLibraryVersion(void)

- **Parameter:**

  None

- **Return:**

  Release lib version.

### 2.2.2 USBIO_ListDevice

- **Description:**

  Show all connected USB-2000 Series Device ID and Board ID.

- **Syntax:**

  char USBIO_ListDevice(USBIO_list *list, char *count);

- **Parameter:**

  list: record module Device ID and Board ID.

  count: how much modules you use now.

- **Return:**

  Error code

### 2.2.3 USBIO_OpenDevice

- **Description:**

  Open specific device with device file.

- **Syntax:**

  int USBIO_OpenDevice(WORD DEVICEID, BYTE BOARDID, int *DevNum)

- **Parameter:**

DEVICEID: Device module ID (Check in ICPDAS_USBIO.h or demo "usbio_list")

BoardID: The BoardID to match correct device.

*DevNum: The serial number to control the correct device.

- **Return:**

  Error code

---

### 2.2.4 USBIO_CloseDevice

- **Description:**

  Close specific device with device file, and release resource.

- **Syntax:**

  int USBIO_CloseDevice (BYTE HIDDev)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

- **Return:**

  Error code

---

### 2.2.5 USBIO_SetCommTimeout

- **Description:**

  Set the communication timeout between packet send and receive.

  The default value when first initial is 100ms.

- **Syntax:**

  int USBIO_SetCommTimeout(BYTE HIDDev, DWORD i_dwCommTimeout)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_dwCommTimeout: The communication timeout in millisecond.(ms)

- **Return:**

  Error code

### 2.2.6 USBIO_GetCommTimeout

- **Description:**

  Get the communication timeout between packet send and receive.

  The default value when first initial is 100ms.

- **Syntax:**

  int USBIO_GetCommTimeout(BYTE HIDDev, DWORD

  * o_dwCommTimeout)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwCommTimeout: The communication timeout in millisecond.(ms)

- **Return:**

  Error code

---

### 2.2.7 USBIO_SetAutoResetWDT

- **Description:**

  Enable / disable the handle of watchdog by library.

  The library takes care of the watchdog automatically

  when first loaded.

  This advantage brings an easy way to access with USB modules.

  But in other side,

  sometimes users want to handle watchdog themselves.

  This API offers this functionality to disable the library

  to automatically handle watchdog.

- **Syntax:**

  int USBIO_SetAutoResetWDT(BYTE HIDDev, BOOL i_bEnable )

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_bEnable: To enable/disable the library to automatically handle
  watchdog.

- **Return:**

Error code

## 2.3 Device Functions

### 2.3.1 USBIO_RefreshDeviceInfo

- **Description:**

  Refresh all information of this device.

- **Syntax:**

  int USBIO_RefreshDeviceInfo(BYTE HIDDev)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

- **Return:**

  Error code

### 2.3.2 USBIO_GetSoftWDTTimeout

- **Description:**

  Get the software WDT timeout of I/O module.

- **Syntax:**

  int USBIO_GetSoftWDTTimeout(BYTE HIDDev, DWORD * o_dwSoftWDTTimeout)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwSoftWDTTimeout: The software WDT timeout in millisecond.(ms)

- **Return:**

  Error code

### 2.3.3 USBIO_GetSupportIOMask

- **Description:**

  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 |

  This mask can help you to identify what types of IO are supported in the device.

- **Syntax:**

  int USBIO_GetSupportIOMask(BYTE HIDDev, BYTE * o_bySupportIOMask)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_bySupportIOMask: The support IO mask of the device.

- **Return:**

  Error code

### 2.3.4 USBIO_GetDeviceID

- **Description:**

  Get ID of the device.

- **Syntax:**

  int USBIO_GetDeviceID(BYTE HIDDev, DWORD * o_dwDeviceID)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwDeviceID: To get Device ID.

- **Return:**

  Error code

### 2.3.5 USBIO_GetFwVer

- **Description:**

  Get firmware version of the device.

- **Syntax:**

  int USBIO_GetFwVer(BYTE HIDDev, WORD * o_wFwVer)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_wFwVer: To get firmware version.

- **Return:**

  Error code

### 2.3.6 USBIO_GetDeviceNickName

- **Description:**

  Get nick name of the device.

- **Syntax:**

  int USBIO_GetDeviceNickName(BYTE HIDDev, BYTE * o_byDeviceNickName)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byDeviceNickName: To get nickname.

- **Return:**

  Error code

### 2.3.7 USBIO_GetDeviceSN

- **Description:**

  Get serial number of the device.

- **Syntax:**

  int USBIO_GetDeviceSN(BYTE HIDDev, BYTE* o_byDeviceSN)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

*o_byDeviceSN: To get Device SN.

- **Return:**

  Error code

---

### 2.3.8 USBIO_GetDITotal

- **Description:**

  Get DI total number of channels of the device.

- **Syntax:**

  int USBIO_GetDITotal(BYTE HIDDev, BYTE * o_byDITotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byDITotal: The DI total number of channels.

- **Return:**

  Error code

---

### 2.3.9 USBIO_GetDOTotal

- **Description:**

  Get DO total number of channels of the device.

- **Syntax:**

  int USBIO_GetDOTotal(BYTE HIDDev, BYTE * o_byDOTotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byDOTotal: The DO total number of channels.

- **Return:**

  Error code

---

### 2.3.10 USBIO_GetAITotal

- **Description:**

  Get AI total number of channels of the device.

- **Syntax:**

  int USBIO_GetAITotal(BYTE HIDDev, BYTE * o_byAITotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byAITotal: The AI total number of channels.

- **Return:**

  Error code

---

### 2.3.11 USBIO_GetAOTotal

- **Description:**

  Get AO total number of channels of the device.

- **Syntax:**

  int USBIO_GetAOTotal(BYTE HIDDev, BYTE * o_byAOTotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byAOTotal: The AO total number of channels.

- **Return:**

  Error code

---

### 2.3.12 USBIO_GetPITotal

- **Description:**

  Get PI total number of channels of the device.

- **Syntax:**

  int USBIO_GetPITotal(BYTE HIDDev, BYTE * o_byPITotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byPITotal: The PI total number of channels.

- **Return:**

  Error code

### 2.3.13 USBIO_GetPOTotal

- **Description:**

  Get PO total number of channels of the device.

- **Syntax:**

  int USBIO_GetPOTotal(BYTE HIDDev, BYTE * o_byPOTotal)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byPOTotal: The PO total number of channels.

- **Return:**

  Error code

### 2.3.14 USBIO_ModuleName

- **Description:**

  Get Device module name.

- **Syntax:**

  int USBIO_ModuleName(BYTE HIDDev, char *module);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *module: char array to get module name.

- **Return:**

  Error code

### 2.3.15 USBIO_SetUserDefinedBoardID

- **Description:**

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

- **Syntax:**

  int USBIO_SetUserDefinedBoardID(BYTE HIDDev, BYTE i_byBID)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  I_byBID: The board ID to set.

- **Return:**

  Error code

---

### 2.3.16 USBIO_SetDeviceNickName

- **Description:**

  Set nick name of this device.

  The maximum number of the character of this device is 32.

- **Syntax:**

  int USBIO_SetDeviceNickName(BYTE HIDDev, BYTE* i_byDeviceNickName)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byDeviceNickName:  The byte array of the nick name to set.

- **Return:**

  Error code

---

### 2.3.17 USBIO_SetSoftWDTTimeout

- **Description:**

  Set the software WDT timeout.

  The minimum value of timeout is 100ms, and maximum is 30 minutes.

- **Syntax:**

  int USBIO_SetSoftWDTTimeout(BYTE HIDDev, DWORD i_dwSoftWDTTimeout)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  I_dwSoftWDTTimeout: The software WDT timeout in millisecond(ms)

- **Return:**

  Error code

## 2.3.18 USBIO_LoadDefault

- **Description:**

  Load default setting.

- **Syntax:**

  int USBIO_LoadDefault(BYTE HIDDev)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

- **Return:**

  Error code

## 2.3.19 USBIO_StopBulk

- **Description:**

  Stop current bulk process.

- **Syntax:**

  int USBIO_StopBulk(BYTE HIDDev)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

- **Return:**

  Error code

## 2.4 Digital Output Functions

### 2.4.1 USBIO_DO_GetPowerOnEnable

- **Description:**

  Digital Output function - Get Power-On Enable.

- **Syntax:**

  int USBIO_DO_GetPowerOnEnable(BYTE HIDDev, BYTE * o_byPowerOnEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byPowerOnEnable: The power-on enable mask. Each byte represents the power-on enable / disable configuration of each channel.

- **Return:**

  Error code

### 2.4.2 USBIO_DO_GetSafetyEnable

- **Description:**

  Digital Output function - Get Safety Enable.

- **Syntax:**

  int USBIO_DO_GetSafetyEnable(BYTE HIDDev, BYTE * o_bySafetyEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_bySafetyEnable: The safety enable mask. Each bit of the mask represents the safety enable / disable configuration of each channel.

- **Return:**

  Error code

## 2.4.3 USBIO_DO_GetSafetyValue

- **Description:**

  Digital Output function - Get Safety Value.

- **Syntax:**

  int USBIO_DO_GetSafetyValue(BYTE HIDDev, BYTE * o_bySafetyValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_bySafetyValue: The safety value. Each bit represents the safety value of each channel.

- **Return:**

  Error code

## 2.4.4 USBIO_DO_GetDigitalOutputInverse

- **Description:**

  Digital output function - Get DO Output Inverse.

| Inverse | Value |
|---|---|
| No | 0 |
| Yes | 1 |

- **Syntax:**

  USBIO_DO_GetDigitalOutputInverse(BYTE HIDDev, DWORD* o_dwInverse);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwInverse: The inverse setting.

- **Return:**

  Error code

### 2.4.5 USBIO_DO_ReadValue

- **Description:**

  Digital Output function - Read DO Value.

- **Syntax:**

  int USBIO_DO_ReadValue(BYTE HIDDev, BYTE* o_byDOValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byDOValue: The DO value. Each bit represents the DO value of each channel.

- **Return:**

  Error code

### 2.4.6 USBIO_DO_SetPowerOnEnableToChannel

- **Description:**

  Digital Output function - Set Power-On enable for specific channel.

  The value of the enable byte is listed below.

| Enable Byte | Value |
|---|---|
| Off | 0 |
| On | 1 |

- **Syntax:**

  int USBIO_DO_SetPowerOnEnableToChannel(BYTE HIDDev, BYTE i_byChToSet, BYTE i_byPowerOnEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_byPowerOnEnable: The power-on enable for the specific channel.

- **Return:**

  Error code

### 2.4.7 USBIO_DO_SetPowerOnEnable

- **Description:**

  Digital Output function - Set Power-On enable for all channel.

  The value of the enable byte is listed below.

| Enable Byte | Value |
|---|---|
| Off | 0 |
| On | 1 |

- **Syntax:**

  int USBIO_DO_SetPowerOnEnable(BYTE HIDDev, BYTE* i_byPowerOnEnables)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byPowerOnEnables: The power-on enable for the specific channel.

- **Return:**

  Error code

### 2.4.8 USBIO_DO_SetSafetyEnable

- **Description:**

  Digital Output function - Set Safety Enable.

  Each bit represents channel safety enable.

  The value of the safety enable is shown below.

| Enable Byte | Value |
|---|---|
| Off | 0 |
| On | 1 |

- **Syntax:**

  int USBIO_DO_SetSafetyEnable(BYTE HIDDev, BYTE* i_bySafetyEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_bySafetyEnable: The safety enable mask. Each bit represents the safety configuration of each channel.

- **Return:**

  Error code

---

## 2.4.9 USBIO_DO_SetSafetyValue

- **Description:**

  Digital Output function - Set Safety Value. Each bit represents safety value. The value of the safety is shown below.

| Enable Byte | Value |
|---|---|
| Off | 0 |
| On | 1 |

- **Syntax:**

  int USBIO_DO_SetSafetyValue(BYTE HIDDev, BYTE* i_bySafetyValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_bySafetyValue: The safety value. Each bit represents the safety value of each channel.

- **Return:**

  Error code

## 2.4.10 USBIO_DO_SetDigitalOutputInverse

- **Description:**

  Digital output function - Set DO Output Inverse

- **Syntax:**

  int USBIO_DO_SetDigitalOutputInverse(BYTE HIDDev, DWORD i_dwInverse);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_dwInverse: The inverse setting.

- **Return:**

  Error code

## 2.4.11 USBIO_DO_WriteValue

- **Description:**

  Digital Output function - Write DO Value.

  Each bit represents channel value.

  The value of the digital output is shown below.

| Enable Byte | Value |
|---|---|
| Off | 0 |
| On | 1 |

- **Syntax:**

  int USBIO_DO_WriteValue(BYTE HIDDev, BYTE* i_byDOValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byDOValue: The DO value. Each bit represents the digital output value of each channel.

- **Return:**

  Error code

### 2.4.12 USBIO_DO_WriteValueToChannel

- **Description:**

  Digital Output function - Write DO Value to specific channel.

  The value of the digital output is shown below.

  | Enable Byte | Value |
  |---|---|
  | Off | 0 |
  | On | 1 |

- **Syntax:**

  int USBIO_DO_WriteValueToChannel(BYTE HIDDev, BYTE i_byChannel, BYTE i_byValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChannel: The specific DO channel to be set.

  i_byValue: The DO on / off bit.

- **Return:**

  Error code

## 2.5 Digital Input Functions

### 2.5.1 USBIO_DI_GetDigitalFilterWidth

- **Description:**

  Digital Input function – Get DI channel digital filter time.

  The default value when first initial is 0.1ms.

- **Syntax:**

  int USBIO_DI_GetDigitalFilterWidth(BYTE HIDDev, WORD* o_wFilterWidth)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_wFilterWidth: Get digital filter time.

- **Return:**

  Error code

### 2.5.2 USBIO_DI_GetDigitalValueInverse

- **Description:**

  Digital Input function – Get DI channel inverse setting.

- **Syntax:**

  int USBIO_DI_GetDigitalValueInverse(BYTE HIDDev, DWORD* o_dwInverse)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwInverse: Get inverse value.

- **Return:**

  Error code

### 2.5.3 USBIO_DI_GetCntEdgeTrigger

- **Description:**

  Digital Input function – Get DI channel counter edge trigger setting.

- **Syntax:**

  int USBIO_DI_GetCntEdgeTrigger(BYTE HIDDev, DWORD* o_dwEdgeTrig)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwEdgeTrig: Get edge trigger status value.

- **Return:**

  Error code

### 2.5.4 USBIO_DI_ReadValue

- **Description:**

  Digital Input function – Read DI value, every channel use one bit, and eight channels shared one byte.

- **Syntax:**

  int USBIO_DI_ReadValue(BYTE HIDDev, BYTE* o_byDIValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byDIValue: Get DI value.

- **Return:**

  Error code

### 2.5.5 USBIO_DI_ReadCounterValue

- **Description:**

  Digital Input function – Get DI channel counters,

  when DI channel on or off, counter will add one.

- **Syntax:**

int USBIO_DI_ReadCounterValue(BYTE HIDDev, WORD* o_wDICntValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_wDICntValue: Get counters with every channel.

- **Return:**

  Error code

---

### 2.5.6 USBIO_DI_SetDigitalFilterWidth

- **Description:**

  Digital Input function – Set DI channel digital filter time.

- **Syntax:**

  int USBIO_DI_SetDigitalFilterWidth(BYTE HIDDev, WORD i_wFilterWidth)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_wFilterWidth: To set DI digital filter time.

- **Return:**

  Error code

---

### 2.5.7 USBIO_DI_SetDigitalValueInverse

- **Description:**

  Digital Input function – Set Digital Value Inverse for all channel.

  The value of the Digital Inverse is shown below.

| Value | Inverse |
|---|---|
| 0 | No |
| 1 | Yes |

- **Syntax:**

  int USBIO_DI_SetDigitalValueInverse(BYTE HIDDev, DWORD i_dwInverse);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_dwInverse: Set value inverse status.

- **Return:**

  Error code

### 2.5.8 USBIO_DI_SetCntEdgeTrigger

- **Description:**

  Digital Input function – Set Counter Edge Trigger for all channel.

  The value of the Edge Trigger is shown below.

| Value | Edge |
|---|---|
| 0 | Falling |
| 1 | Raising |

- **Syntax:**

  int USBIO_DI_SetCntEdgeTrigger(BYTE HIDDev, DWORD i_dwEdgeTrig)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_dwEdgeTrig; Set value Edge Trigger status.

- **Return:**

  Error code

### 2.5.9 USBIO_DI_WriteClearCounter

- **Description:**

  Digital Input function – Clear DI counter with specific channel.

- **Syntax:**

  int USBIO_DI_WriteClearCounter(BYTE HIDDev, BYTE i_byChToClr)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToClr: Set value to clear DI channel.(0~7)

- **Return:**

  Error code

### 2.5.10 USBIO_DI_WriteClearCounterByMask

- **Description:**

  Digital Input function - Clear DI counter by mask.

- **Syntax:**

  int USBIO_DI_WriteClearCounterByMask(BYTE HIDDev, DWORD i_dwCntClrMask)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_dwCntClrMask: Set value to mask specific channels.(0~ff)

- **Return:**

  Error code

## 2.6 Analog Output Functions

### 2.6.1 USBIO_AO_GetTotalSupportType

- **Description:**

  Analog output function - Get total supported amount.

- **Syntax:**

  int USBIO_AO_GetTotalSupportType(BYTE HIDDev, BYTE * o_byTotalSupportType);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byTotalSupportType: The number of total support type.

- **Return:**

  Error code

### 2.6.2 USBIO_AO_GetSupportTypeCode

- **Description:**

  Analog output function - Get supported type code Please refer to Appendix 2 of user's manual to map AO channels output type

- **Syntax:**

  int USBIO_AO_GetSupportTypeCode(BYTE HIDDev, BYTE* o_bySupportTypeCode);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_bySupportTypeCode: The number of total support type.

- **Return:**

Error code

### 2.6.3 USBIO_AO_GetTypeCode

- **Description:**

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

- **Syntax:**

  int USBIO_AO_GetTypeCode(BYTE HIDDev, BYTE* o_byTypeCode);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_byTypeCode: The byte array of type code.

- **Return:**

Error code

### 2.6.4 USBIO_AO_SetTypeCode

- **Description:**

  Analog output function - Set type code for all channels.

  The type code can reference to Appendix 2.

- **Syntax:**

  int USBIO_AO_SetTypeCode(BYTE HIDDev, BYTE* i_byTypeCodes);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * i_byTypeCodes: The byte array of type code to set.

- **Return:**

Error code

### 2.6.5 USBIO_AO_SetTypeCodeToChannel

- **Description:**

  Analog output function - Set type code for specific channel.

  The type code can reference to Appendix 2

- **Syntax:**

  int USBIO_AO_SetTypeCodeToChannel(BYTE HIDDev, BYTE i_byChToSet, BYTE i_byTypeCode);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_byTypeCode: The type code for the specific channel

- **Return:**

Error code

### 2.6.6 USBIO_AO_GetChEnable

- **Description:**

  Analog output function - Get channel enable/disable.

  Each byte indicates 8 channels enable/disable mask.

  EX: Byte0 -> Channel0 ~ 7

- **Syntax:**

  int USBIO_AO_GetChEnable(BYTE HIDDev, BYTE* o_byChEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_byChEnable: The byte array of channel enable/disable mask.

- **Return:**

Error code

### 2.6.7 USBIO_AO_SetChEnable

- **Description:**

  Analog output function - Set channel enable/disable.

  Each byte indicates 8 channels enable/disable mask.

  Ex: Byte0 -> Channel0 ~ 7

- **Syntax:**

  int USBIO_AO_SetChEnable(BYTE HIDDev, BYTE* i_byChEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * i_byChEnable : The byte array of channel enable/disable mask.

- **Return:**

Error code

### 2.6.8 USBIO_AO_GetResolution

- **Description:**

  Analog output function - Get resolution.

  Each byte indicates each channel real resolution.

- **Syntax:**

  int USBIO_AO_GetResolution(BYTE HIDDev, BYTE* o_byResolution);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_byResolution: The byte array of resolution for each channel.

- **Return:**

Error code

### 2.6.9 USBIO_AO_ReadExpValueHex

- **Description:**

Analog output function - 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

$Y = ( 1+\frac{0x13E}{0xFFFF}) \times (5 - (-5)) + (-5)$

- 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

$Y = ( 0-\frac{0x53E}{0xFFFF}) \times (5 - (-5)) + (-5)$

- 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 AO value cannot detect the channel status. It only read the AO value but has the most efficiency.

- **Syntax:**

int USBIO_AO_ReadExpValueHex(BYTE HIDDev, DWORD* o_dwAOValue);

- **Parameter:**

HIDDev: The serial number to control the correct device.

* o_dwAOValue: The raw value of AO expect value.

- **Return:**

Error code

---

### 2.6.10 USBIO_AO_ReadExpValueFloat

- **Description:**

Analog output function – Read the real AO expect value.

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

- **Syntax:**

int USBIO_AO_ReadExpValueFloat(BYTE HIDDev, float* o_fAOValue);

- **Parameter:**

HIDDev: The serial number to control the correct device.

* o_fAOValue: The true value of AO expect value.

- **Return:**

Error code

### 2.6.11 USBIO_AO_ReadCurValueHex

- **Description:**

  Analog output function - 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

  $Y = ( 1+\frac{0x13E}{0xFFFF}) \times (5 - (-5)) + (-5)$

  - 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

  $Y = ( 0-\frac{0x53E}{0xFFFF}) \times (5 - (-5)) + (-5)$

  - 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 AO value cannot detect the channel status. It only read the AO value but has the most efficiency

- **Syntax:**

  int USBIO_AO_ReadCurValueHex(BYTE HIDDev, DWORD* o_dwAOValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_dwAOValue: The raw value of AO current value.

- **Return:**

Error code

---

### 2.6.12 USBIO_AO_ReadCurValueFloat

- **Description:**

  Analog output function - Readthe real AO current value.

  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:**

  int USBIO_AO_ReadCurValueFloat(BYTE HIDDev, float* o_fAOValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_fAOValue: The true value of AO current value.

- **Return:**

Error code

## 2.6.13 USBIO_AO_WriteValueHexToChannel

- **Description:**

Analog output function - 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

$$Y = ( 0-\frac{0x53E}{0xFFFF}) \times (5 - (-5)) + (-5)$$

- 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:**

int USBIO_AO_WriteValueHexToChannel(BYTE HIDDev, BYTE i_byChToSet, DWORD i_dwAOVal);

- **Parameter:**

HIDDev: The serial number to control the correct device.

i_byChToSet: The specific AO channel to be set.

i_dwAOVal: The AO value.

- **Return:**

Error code

### 2.6.14 USBIO_AO_WriteValueHex

- **Description:**

  Analog output function - Write AO expect value to all channels in double word (digital) format.

- **Syntax:**

  int USBIO_AO_WriteValueHex(BYTE HIDDev, DWORD* i_dwAOValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * i_dwAOValue: The AO value.

- **Return:**

Error code

### 2.6.15 USBIO_AO_WriteValueFloatToChannel

- **Description:**

  Analog output function - Write AO expect value to specifying channel in float (analog) format.

  The writing value is calculated, users write real value for current output type. Ex: The writing value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

- **Syntax:**

  int USBIO_AO_WriteValueFloatToChannel(BYTE HIDDev, BYTE i_byChToSet, float i_fAOExpValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific AO channel to be set.

i_fAOExpValue: The AO value.

- **Return:**

Error code

---

### 2.6.16 USBIO_AO_WriteValueFloat

- **Description:**

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

- **Syntax:**

  int USBIO_AO_WriteValueFloat(BYTE HIDDev, float* i_fAOExpValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * i_fAOExpValue: The AO value.

- **Return:**

Error code

---

### 2.6.17 USBIO_AO_GetPowerOnEnable

- **Description:**

  Analog output function - Get Power-OnEnable. Each channeltakes one byte.

- **Syntax:**

  int USBIO_AO_GetPowerOnEnable(BYTE HIDDev, BYTE* o_byPowerOnEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byPowerOnEnable: The byte array of channel enable/disable mask.

- **Return:**

Error code

---

### 2.6.18 USBIO_AO_SetPowerOnEnable

- **Description:**

  Analog output function - Set Power-OnEnable. Each channel takes one byte.

- **Syntax:**

  int USBIO_AO_SetPowerOnEnable(BYTE HIDDev, BYTE* i_byPowerOnEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byPowerOnEnable: The byte array of channel enable/disable mask

- **Return:**

Error code

---

### 2.6.19 USBIO_AO_GetPowerOnValueHex

- **Description:**

  Analog output function - GetPower-On Value. Each channel takes one unit of DWORD array.

- **Syntax:**

  int USBIO_AO_GetPowerOnValueHex(BYTE HIDDev, DWORD* o_dwPwrOnValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_dwPwrOnValue: The DWORD array of Power-On Value.

- **Return:**

Error code

### 2.6.20 USBIO_AO_GetPowerOnValueFloat

- **Description:**

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

- **Syntax:**

  int USBIO_AO_GetPowerOnValueFloat(BYTE HIDDev, float* o_fPwrOnValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  * o_fPwrOnValue: The float array of Power-On Value.

- **Return:**

Error code

### 2.6.21 USBIO_AO_SetPowerOnValueHexToChannel

- **Description:**

  Analog output function - 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

$Y = ( 0-\frac{0x53E}{0xFFFF}) \times (5 - (-5)) + (-5)$

- 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:**

int USBIO_AO_SetPowerOnValueHexToChannel(BYTE HIDDev, BYTE i_byChToSet, DWORD i_dwPwrOnValue);

- **Parameter:**

HIDDev: The serial number to control the correct device.

i_byChToSet: The specific AO channel to be set.

i_dwPwrOnValue: The AO PowerOnValue .

- **Return:**

Error code

---

### 2.6.22 USBIO_AO_SetPowerOnValueHex

- **Description:**

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

- **Syntax:**

int USBIO_AO_SetPowerOnValueHex(BYTE HIDDev, DWORD* i_dwPwrOnValue);

- **Parameter:**

HIDDev: The serial number to control the correct device.

* i_dwPwrOnValue: The AO Power-On Value.

- **Return:**

Error code

### 2.6.23 USBIO_AO_SetPowerOnValueFloatToChannel

- **Description:**

  Analog output function - Set AO Power-On Value to specifying channel in float (analog) format.

  The writing value is calculated, users write real value for current output type. Ex: The writing value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

- **Syntax:**

  int USBIO_AO_SetPowerOnValueFloatToChannel(BYTE HIDDev, BYTE i_byChToSet, float i_fPwrOnValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific AO channel to be set.

  i_fPwrOnValue: The AO Power-On Value.

- **Return:**

Error code

### 2.6.24 USBIO_AO_SetPowerOnValueFloat

- **Description:**

  Analog output function - Set AO Power-On Value to all channels in float (analog) format.

  The writing value is calculated, users write real value for current output type. Ex: The writing value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

- **Syntax:**

  int USBIO_AO_SetPowerOnValueFloat(BYTE HIDDev, float* i_fPwrOnValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_fPwrOnValue: The AO Power-On Value.

- **Return:**

Error code

---

### 2.6.25 USBIO_AO_GetSafetyEnable

- **Description:**

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

- **Syntax:**

  int USBIO_AO_GetSafetyEnable(BYTE HIDDev, BYTE* o_bySafetyEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_bySafetyEnable: The byte array of Safety enable/disable mask.

- **Return:**

Error code

---

### 2.6.26 USBIO_AO_SetSafetyEnable

- **Description:**

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

- **Syntax:**

  int USBIO_AO_SetSafetyEnable(BYTE HIDDev, BYTE* i_bySafetyEnable);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_bySafetyEnable: The byte array of channel enable/disable mask.

- **Return:**

Error code

### 2.6.27 USBIO_AO_GetSafetyValueHex

- **Description:**

  Analog output function - GetSafety value. Each channel takes one unit of DWORD array.

- **Syntax:**

  int USBIO_AO_GetSafetyValueHex(BYTE HIDDev, DWORD* o_dwSafetyValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_dwSafetyValue: The DWORD array of Safety Value.

- **Return:**

Error code

### 2.6.28 USBIO_AO_GetSafetyValueFloat

- **Description:**

  Analog output function - GetSafety value. Each channel takes one unit of float array.

- **Syntax:**

  int USBIO_AO_GetSafetyValueFloat(BYTE HIDDev, float* o_fSafetyValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_fSafetyValue: The float array of Safety Value.

- **Return:**

Error code

### 2.6.29 USBIO_AO_SetSafetyValueHexToChannel

- **Description:**

  Analog output function - 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

$$Y = ( 0-\frac{0x53E}{0xFFFF}) \times (5 - (-5)) + (-5)$$

- 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:**

int USBIO_AO_SetSafetyValueHexToChannel(BYTE HIDDev, BYTE i_byChToSet, DWORD i_dwSafetyValue);

- **Parameter:**

HIDDev: The serial number to control the correct device.

i_byChToSet: The specific AO channel to be set.

i_dwSafetyValue: The AO Safety Value.

- **Return:**

Error code

### 2.6.30 USBIO_AO_SetSafetyValueHex

- **Description:**

  Analog output function - Set AO Safety Value to all channels in double word (digital) format.

- **Syntax:**

  int USBIO_AO_SetSafetyValueHex(BYTE HIDDev, DWORD* i_dwSafetyValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_dwSafetyValue: The AO Safety Value.

- **Return:**

Error code

### 2.6.31 USBIO_AO_SetSafetyValueFloatToChannel

- **Description:**

  Analog output function - Set AO Safety Value to specifying channel in float (analog) format.

  The writing value is calculated, users write real value for current output type. Ex: The writing value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

- **Syntax:**

  int USBIO_AO_SetSafetyValueFloatToChannel(BYTE HIDDev, BYTE i_byChToSet, float i_fSafetyValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific AO channel to be set.

  i_fSafetyValue: The AO Safety Value.

- **Return:**

Error code

### 2.6.32 USBIO_AO_SetSafetyValueFloat

- **Description:**

  Analog output function - Set AO Safety Value to all channels in float (analog) format.

  The writing value is calculated, users write real value for current output type. Ex: The writing value is 1.316 in -2.5 ~ +2.5V, the output signal is 1.316V.

- **Syntax:**

  int USBIO_AO_SetSafetyValueFloat(BYTE HIDDev, float* i_fSafetyValue);

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_fSafetyValue: The AO Safety Value.

- **Return:**

Error code

## 2.7 Analog Input Functions

### 2.7.1 USBIO_AI_GetTotalSupportType

- **Description:**

  Analog input function - Get total supported amount.

- **Syntax:**

  int USBIO_AI_GetTotalSupportType(BYTE HIDDev, BYTE * o_byTotalSupportType)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byTotalSupportType: The number of total support type.

- **Return:**

  Error code

### 2.7.2 USBIO_AI_GetSupportTypeCode

- **Description:**

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

- **Syntax:**

  int USBIO_AI_GetSupportTypeCode(BYTE HIDDev, BYTE* o_bySupportTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.
  *o_bySupportTypeCode: The number of total support type.

- **Return:**

  Error code

### 2.7.3 USBIO_AI_GetTypeCode

- **Description:**

  Analog input function - Get type code.

Refer to user's manual to map AI channels input type.

The type code can reference to Appendix 1.

- **Syntax:**

  int USBIO_AI_GetTypeCode(BYTE HIDDev, BYTE* o_byTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byTypeCode: The byte array of type code.

- **Return:**

  Error code

### 2.7.4 USBIO_AI_GetChCJCOffset

- **Description:**

  Analog input function - Get channel CJC offset

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

- **Syntax:**

  int USBIO_AI_GetChCJCOffset(BYTE HIDDev, float* o_fChCJCOffset)

- **Parameter:**

  HIDDev: The serial number to control the correct device.
  *o_fChCJCOffset: The float array of channel CJC offset

- **Return:**

  Error code

### 2.7.5 USBIO_AI_GetChEnable

- **Description:**

  Analog input function - Get channel enable/disable.

  Each byte indicates 8 channels enable/disable mask.

  EX: Byte0 -> Channel 0 ~ 7

- **Syntax:**

  int USBIO_AI_GetChEnable(BYTE HIDDev, BYTE* o_byChEnable)

- **Parameter:**

HIDDev: The serial number to control the correct device.
*o_byChEnable: The byte array of channel enable/disable mask

- **Return:**

  Error code

---

### 2.7.6 USBIO_AI_GetFilterRejection

- **Description:**

  Analog input function - Get filter rejection.

  The value of the rejection setting is shown below.

| Rejection Setting | Value |
|---|---|
| 60Hz | **0** |
| 50Hz | **1** |

- **Syntax:**

  int USBIO_AI_GetFilterRejection(BYTE HIDDev, BYTE* o_byFilterRejection)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byFilterRejection: The filter rejection.

- **Return:**

  Error code

---

### 2.7.7 USBIO_AI_GetCJCOffset

- **Description:**

  Analog input function - Get CJC offset

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

- **Syntax:**

  int USBIO_AI_GetCJCOffset(BYTE HIDDev, float* o_fCJCOffset)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_fCJCOffset: The CJC offset.

- **Return:**

  Error code

### 2.7.8 USBIO_AI_GetCJCEnable

- **Description:**

  Analog input function - Get CJC enable.

  The value of the CJC enable is shown below.

| Enable Setting | Value |
|---|---|
| Disable | **0** |
| Enable | **1** |

- **Syntax:**

  int USBIO_AI_GetCJCEnable(BYTE HIDDev, BYTE* o_byCJCEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byCJCEnable: The CJC enables.

- **Return:**

  Error code

### 2.7.9 USBIO_AI_GetWireDetectEnable

- **Description:**

  Analog input function - Get wire detect enable.

  The value of the wire detect enable is shown below.

| Enable Setting | Value |
|---|---|
| Disable | **0** |
| Enable | **1** |

- **Syntax:**

int USBIO_AI_GetWireDetectEnable(BYTE HIDDev, BYTE* o_byWireDetectEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byWireDetectEnable: The wire detect enable.

- **Return:**

  Error code

---

## 2.7.10 USBIO_AI_GetResolution

- **Description:**

  Analog input function - Get resolution.

  Each byte indicates each channel real resolution.

- **Syntax:**

  int USBIO_AI_GetResolution(BYTE HIDDev, BYTE* o_byResolution)

- **Parameter**

  HIDDev: The serial number to control the correct device.
  *o_byResolution: The byte array of resolution for each channel

- **Return:**

  Error code

---

## 2.7.11 USBIO_AI_ReadValueHex

- **Description:**

  Analog input function - Read AI value in double word (digital) format without channel status. 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.

- **Syntax:**

  int USBIO_AI_ReadValueHex(BYTE HIDDev, DWORD* o_dwAIValue)

- **Parameter:**

HIDDev: The serial number to control the correct device.

*o_dwAIValue: The raw value of AI value.

- **Return:**

  Error code

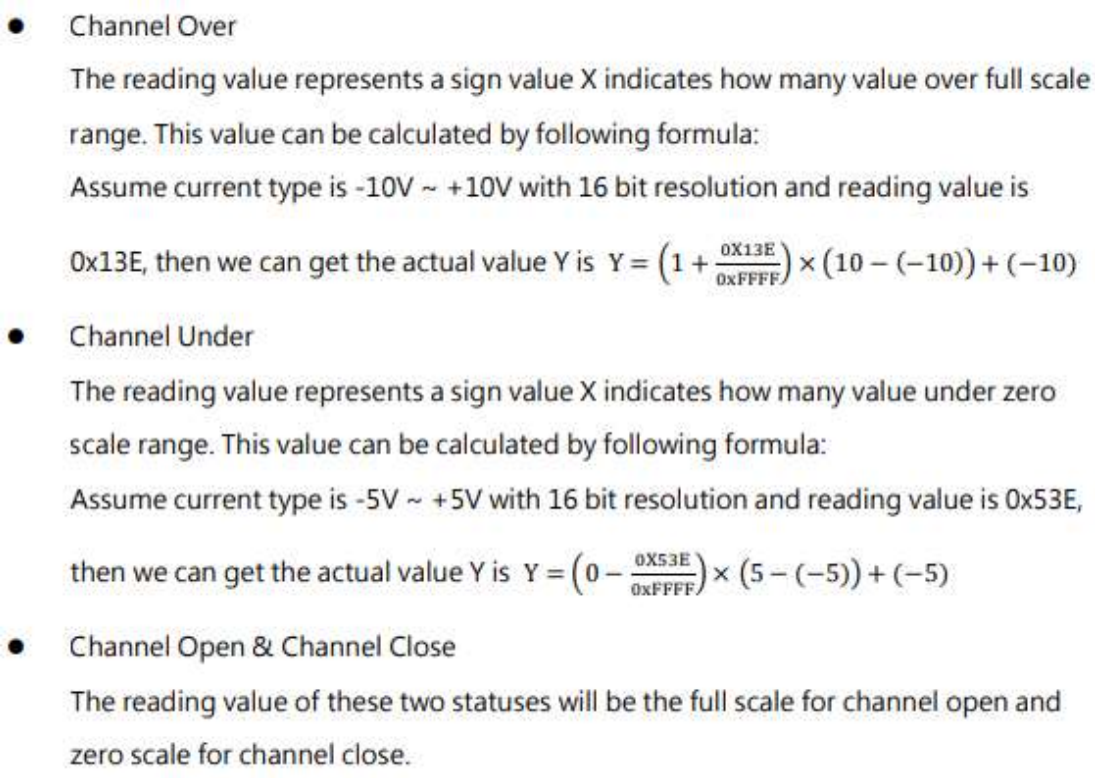
---

### 2.7.12 USBIO_AI_ReadValueHexWithChSta

- **Description:**

  Analog input function - Read AI value in double word (digital) format with channel status. 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 $Y = \left(1 + \frac{0X13E}{0xFFFF}\right) \times (10 - (-10)) + (-10)$

- 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 $Y = \left(0 - \frac{0X53E}{0xFFFF}\right) \times (5 - (-5)) + (-5)$

- 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:**

int USBIO_AI_ReadValueHexWithChSta(BYTE HIDDev, DWORD* o_dwAIValue, BYTE* o_byChStatus)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  o_dwAIValue: The raw value of AI value.

  *o_byChStatus: The byte array of channel status.

- **Return:**

  Error code

---

## 2.7.13 USBIO_AI_ReadValueFloat

- **Description:**

  Analog input function - 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:**

  int USBIO_AI_ReadValueFloat(BYTE HIDDev, float* o_fAIValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_fAIValue: The true value of AI value.

- **Return:**

  Error code

---

## 2.7.14 USBIO_AI_ReadValueFloatWithChSta

- **Description:**

  Analog input function - Read the real AI value with channel status.

- **Syntax:**

  int USBIO_AI_ReadValueFloatWithChSta(BYTE HIDDev, float* o_fAIValue, BYTE* o_byChStatus)

- **Parameter:**

HIDDev: The serial number to control the correct device.

*o_fAIValue: The true value of AI value.

*o_byChStatus: The byte array of channel status.

- **Return:**

  Error code

## 2.7.15 USBIO_AI_ReadBulkValue

- **Description:**

  Analog input function - Get bulk AI value (Fast acquire functionality).

- **Syntax:**

  int USBIO_AI_ReadBulkValue(BYTE HIDDev,

  BYTE i_byStartCh, BYTE i_byChTotal, DWORD i_dwSampleWidth,

  float i_fSampleRate, DWORD i_dwBufferWidth,

  DWORD* o_dwDataBuffer, OnBulkValueFinishEvent i_CBFunc)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byStartCh: The starting acquire channel.

  i_byChTotal: The total channels to acquire.

  i_dwSampleWidth: The sampling width (ms).

  i_fSampleRate: The sampling rate (Hz).

  i_dwBufferWidth: The width of the buffer.

  *o_dwDataBuffer: The buffer to store.

  i_CBFunc: The callback function.

- **Return:**

  Error code

## 2.7.16 USBIO_AI_ReadCJCValue

- **Description:**

  Analog input function - Read the current CJC value on the module.

- **Syntax:**

  int USBIO_AI_ReadCJCValue(BYTE HIDDev, float* o_fCJCValue)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_fCJCValue: The CJC value.

- **Return:**

  Error code

---

### 2.7.17 USBIO_AI_SetTypeCodeToChannel

- **Description:**

  Analog input function - Set type code for specific channel. The type code can reference to Appendix 1.

- **Syntax:**

  int USBIO_AI_SetTypeCodeToChannel(BYTE HIDDev, BYTE i_byChToSet, BYTE i_byTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_byTypeCode: The type code for the specific channel.

- **Return:**

  Error code

---

### 2.7.18 USBIO_AI_SetTypeCode

- **Description:**

  Analog input function - Set type code for all channels. The type code can reference to Appendix 1.

- **Syntax:**

  int USBIO_AI_SetTypeCode(BYTE HIDDev, BYTE* i_byTypeCodes)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byTypeCodes: The byte array of type code to set.

- **Return:**

  Error code

## 2.7.19 USBIO_AI_SetChCJCOffsetToChannel

- **Description:**

  Analog input function - Set channel CJC offset for specific channel. The valid range of offset is -40.96 ~ +40.95.

- **Syntax:**

  int USBIO_AI_SetChCJCOffsetToChannel(BYTE HIDDev, BYTE i_byChToSet, float i_fChCJCOffset)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_fChCJCOffset: The CJC offset for the specific channel.

- **Return:**

  Error code

## 2.7.20 USBIO_AI_SetChCJCOffset

- **Description:**

  Analog input function - Set channel CJC offset for every channel. The valid range of offset is -40.96 ~ +40.95.

- **Syntax:**

  int USBIO_AI_SetChCJCOffset(BYTE HIDDev, float* i_fChCJCOffsets)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_fChCJCOffsets: The float array of channel CJC offset to set.

- **Return:**

  Error code

## 2.7.21 USBIO_AI_SetChEnable

- **Description:**

  Analog input function - Set channel enable/disable.

  Each byte indicates 8 channels enable/disable mask.

  Ex: Byte0 -> Channel 0 ~ 7

- **Syntax:**

  int USBIO_AI_SetChEnable(BYTE HIDDev, BYTE* i_byChEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byChEnable: The byte array of channel enable/disable mask.

- **Return:**

  Error code

## 2.7.22 USBIO_AI_SetFilterRejection

- **Description:**

  Analog input function - Set filter rejection.

  The value of the rejection setting is shown below.

| Rejection Setting | Value |
|---|---|
| 60Hz | **0** |
| 50Hz | **1** |

- **Syntax:**

  int USBIO_AI_SetFilterRejection(BYTE HIDDev, BYTE i_byFilterRejection)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byFilterRejection: The filter rejection.

- **Return:**

  Error code

## 2.7.23 USBIO_AI_SetCJCOffset

- **Description:**

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

- **Syntax:**

  int USBIO_AI_SetCJCOffset(BYTE HIDDev, float i_fCJCOffset)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_fCJCOffset: The CJC offset.

- **Return:**

  Error code

## 2.7.24 USBIO_AI_SetCJCEnable

- **Description:**

  Analog input function - Set CJC enable.

  The value of the set CJC enable is shown below.

| Enable Setting | Value |
|---|---|
| Disable | **0** |
| Enable | **1** |

- **Syntax:**

  int USBIO_AI_SetCJCEnable(BYTE HIDDev, BYTE i_byCJCEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byCJCEnable: The CJC enable.

- **Return:**

  Error code

### 2.7.25 USBIO_AI_SetWireDetectEnable

- **Description:**

  Analog input function - Set wire detect enable.

  The value of the set wire detect enable is shown below.

| Enable Setting | Value |
|---|---|
| Disable | **0** |
| Enable | **1** |

- **Syntax:**

  int USBIO_AI_SetWireDetectEnable(BYTE HIDDev, BYTE i_byWireDetectEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byWireDetectEnable: The wire detect enable.

- **Return:**

  Error code

## 2.8 Pulse Input Functions

### 2.8.1 USBIO_PI_GetTotalSupportType

- **Description:**

  Pulse input function - Get total supported amount.

- **Syntax:**

  int USBIO_PI_GetTotalSupportType(BYTE HIDDev, BYTE* o_byTotalSupportType)

- **Parameter:**

  HIDDev: The serial number to control the correct device.
  *o_byTotalSupportType: The number of total support type

- **Return:**

  Error code

### 2.8.2 USBIO_PI_GetSupportTypeCode

- **Description:**

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

- **Syntax:**

  int USBIO_PI_GetSupportTypeCode(BYTE HIDDev, BYTE* o_bySupportTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_bySupportTypeCode: Get every support type code.

- **Return:**

  Error code

### 2.8.3 USBIO_PI_GetTypeCode

- **Description:**

  Pulse input function - Get type code. Refer to user's manual to map PI channels input type. The type code can reference to Appendix 3.

- **Syntax:**

  int USBIO_PI_GetTypeCode(BYTE HIDDev, BYTE* o_byTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byTypeCode: The byte array of type code.

- **Return:**

  Error code

### 2.8.4 USBIO_PI_GetTriggerMode

- **Description:**

  Pulse input function - Get trigger mode.

  The value of the trigger mode is shown below.

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

- **Syntax:**

  int USBIO_PI_GetTriggerMode(BYTE HIDDev, BYTE* o_byTriggerMode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byTriggerMode: The byte array of trigger mode.

- **Return:**

  Error code

### 2.8.5 USBIO_PI_GetChIsolatedFlag

- **Description:**

  Pulse input function - Get channel isolated flag. Each byte indicates 8 channels flag. EX: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_GetChIsolatedFlag(BYTE HIDDev, BYTE* o_byChIsolatedFlag)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byChIsolatedFlag: The byte arrays of channel isolated flag.

- **Return:**

  Error code

### 2.8.6 USBIO_PI_GetLPFilterEnable

- **Description:**

  Pulse input function - Get low-pass filter enable. Each byte indicates 8 channels enable/disable mask. EX: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_GetLPFilterEnable(BYTE HIDDev, BYTE* o_byLPFilterEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_byLPFilterEnable: The byte array of the low-pass filter enable mask.

- **Return:**

  Error code

### 2.8.7 USBIO_PI_GetLPFilterWidth

- **Description:**

Pulse input function - 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.

Refer to following table to see what low-pass filter width is referred to.

The value of the channel index is shown below.

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

- **Syntax:**

int USBIO_PI_GetLPFilterWidth(BYTE HIDDev, WORD* o_wLPFilterWidth)

- **Parameter:**

HIDDev: The serial number to control the correct device.

*o_wLPFilterWidth: The byte array of low-pass filter width in us.

- **Return:**

Error code

### 2.8.8 USBIO_PI_ReadValue

- **Description:**

Pulse input function - 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:**

int USBIO_PI_ReadValue(BYTE HIDDev, DWORD* o_dwPIValue, BYTE* o_byChStatus)

- **Parameter:**

HIDDev: The serial number to control the correct device.

*o_dwPIValue: The byte array of the PI channel counter value.

*o_byChStatus: The byte array of the channel status.

- **Return:**

Error code

---

### 2.8.9 USBIO_PI_ReadCntValue

- **Description:**

Pulse input function - 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:**

int USBIO_PI_ReadCntValue(BYTE HIDDev, DWORD* o_dwCntValue, BYTE* o_byChStatus)

- **Parameter:**

HIDDev: The serial number to control the correct device.

*o_dwCntValue: The unsigned long array of the PI channel counter value.

*o_byChStatus: The byte array of the channel status.

- **Return:**

Error code

### 2.8.10 USBIO_PI_ReadFreqValue

- **Description:**

  Pulse input function - Read the 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:**

  int USBIO_PI_ReadFreqValue(BYTE HIDDev, float* o_fFreqValue, BYTE* o_byChStatus)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *o_fFreqValue: The float array of the PI channel frequency value.

  *o_byChStatus: The byte array of the channel status.

- **Return:**

  Error code

### 2.8.11 USBIO_PI_ReadBulkValue

- **Description:**

  Pulse input function - Get bulk PI value (Fast acquire functionality).

- **Syntax:**

  Int USBIO_PI_ReadBulkValue(BYTE HIDDev, BYTE i_byStartCh, BYTE i_byChTotal, DWORD i_dwSampleWidth, float i_fSampleRate, DWORD i_dwBufferWidth, DWORD* o_dwDataBuffer, OnBulkValueFinishEvent i_CBFunc))

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byStartCh: The starting acquire channel.

i_byChTotal: The total channels to acquire.

i_dwSampleWidth: The sampling width (ms).

i_fSampleRate: The sampling rate (Hz).

i_dwBufferWidth: The width of the buffer.

*o_dwDataBuffer: The buffer to store.

i_CBFunc: The callback function.

- **Return:**

  Error code

---

### 2.8.12 USBIO_PI_SetTypeCodeToChannel

- **Description:**

  Pulse input function - Set type code for specific channel. The type code can reference to Appendix 3.

- **Syntax:**

  int USBIO_PI_SetTypeCodeToChannel(BYTE HIDDev, BYTE i_byChToSet, BYTE i_byTypeCode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_byTypeCode: The type code for the specific channel.

- **Return:**

  Error code

---

### 2.8.13 USBIO_PI_SetTypeCode

- **Description:**

  Pulse input function - Set type code for all channels. The type code can reference to Appendix 3.

- **Syntax:**

  int USBIO_PI_SetTypeCode(BYTE HIDDev, BYTE* i_byTypeCodes)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

*i_byTypeCodes: The byte array of the type code to set.

- **Return:**

  Error code

---

### 2.8.14 USBIO_PI_ClearSingleChCount

- **Description:**

  Pulse input function - Clear specific channel count.

- **Syntax:**

  int USBIO_PI_ClearSingleChCount(BYTE HIDDev, BYTE i_byChToClr)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToClr: The channel index for clearing.

- **Return:**

  Error code

---

### 2.8.15 USBIO_PI_ClearChCount

- **Description:**

  Pulse input function - Clear channel count with clear mask.

  Each byte indicates 8 channels clear mask, set for channel clear.

  Ex: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_ClearChCount(BYTE HIDDev, BYTE* i_byClrMask)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byClrMask: The byte array of channel count clear mask.

- **Return:**

  Error code

### 2.8.16 USBIO_PI_ClearSingleChStatus

- **Description:**

  Pulse input function - Clear specific channel count.

- **Syntax:**

  int USBIO_PI_ClearSingleChStatus(BYTE HIDDev, BYTE i_byChToClr)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToClr: The channel index for clearing.

- **Return:**

  Error code

### 2.8.17 USBIO_PI_ClearChStatus

- **Description:**

  Pulse input function - Clear channel status with clear mask.

  Each byte indicates 8 channels clear mask, set for channel clear.

  Ex: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_ClearChStatus(BYTE HIDDev, BYTE* i_byClrMask)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byClrMask: The byte array of channel status clear mask.

- **Return:**

  Error code

### 2.8.18 USBIO_PI_SetTriggerModeToChannel

- **Description:**

  Pulse input function - Set trigger mode to specific channel.

  The value of the trigger is shown below.

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

- **Syntax:**

  int USBIO_PI_SetTriggerModeToChannel(BYTE HIDDev, BYTE i_byChToSet, BYTE i_byTriggerMode)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_byTriggerMode: The type code for the specific channel.

- **Return:**

  Error code

---

### 2.8.19 USBIO_PI_SetTriggerMode

- **Description:**

  Pulse input function - Set trigger mode to all channel.

  The value of the trigger mode is shown below.

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

- **Syntax:**

  int USBIO_PI_SetTriggerMode(BYTE HIDDev, BYTE* i_byTriggerModes)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byTriggerModes: The byte array of trigger mode to set.

- **Return:**

  Error code

### 2.8.20 USBIO_PI_SetChIsolatedFlagToChannel

- **Description:**

  Pulse input function - Set channel isolated flag.

- **Syntax:**

  int USBIO_PI_SetChIsolatedFlagToChannel(BYTE HIDDev, BYTE i_byChToSet, BOOL i_bChIsolatedFlag)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_bChIsolatedFlag: The isolated flag for the specific channel.

- **Return:**

  Error code

### 2.8.21 USBIO_PI_SetChIsolatedFlag

- **Description:**

  Pulse input function - Set channel isolated flag to all channels.

  Each byte indicates 8 channels isolated flag,

  Ex: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_SetChIsolatedFlag(BYTE HIDDev, BYTE* i_byChIsolatedFlag)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_byChIsolatedFlag: The byte arrays of channel isolated flag.

- **Return:**

  Error code

## 2.8.22 USBIO_PI_SetLPFilterEnableToChannel

- **Description:**

  Pulse input function - Set low-pass filter enable to specific channel.

- **Syntax:**

  int USBIO_PI_SetLPFilterEnableToChannel(BYTE HIDDev, BYTE i_byChToSet, BOOL i_bLPFilterEnable)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_bLPFilterEnable: The enable flag for the specific channel.

- **Return:**

  Error code

## 2.8.23 USBIO_PI_SetLPFilterEnable

- **Description:**

  Pulse input function - Set low-pass filter enable to all channel.

  Each byte indicates 8 channels enable mask,

  Ex: Byte0 -> Channel 0 ~ 7.

- **Syntax:**

  int USBIO_PI_SetLPFilterEnable(BYTE HIDDev, BYTE* i_byLPFilterEnables)

- **Parameter:**

  HIDDev: The serial number to control the correct device.
  *i_byLPFilterEnables: The byte array of low-pass filter enable mask.

- **Return:**

  Error code

## 2.8.24 USBIO_PI_SetLPFilterWidthToChannel

- **Description:**

  Pulse input function - Set low-pass filter width

- **Syntax:**

  int USBIO_PI_SetLPFilterWidthToChannel(BYTE HIDDev, BYTE i_byChToSet, WORD i_wLPFilterWidth)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  i_byChToSet: The specific channel to set.

  i_wLPFilterWidth: The low-pass filter width. (us)

- **Return:**

  Error code

## 2.8.25 USBIO_PI_SetLPFilterWidth

- **Description:**

  Pulse input function - Set low-pass filter enable to all channel.

- **Syntax:**

  int USBIO_PI_SetLPFilterWidth(BYTE HIDDev, WORD* i_wLPFilterWidths)

- **Parameter:**

  HIDDev: The serial number to control the correct device.

  *i_wLPFilterWidths: The byte array of low-pass filter enable mask.

- **Return:**

  Error code

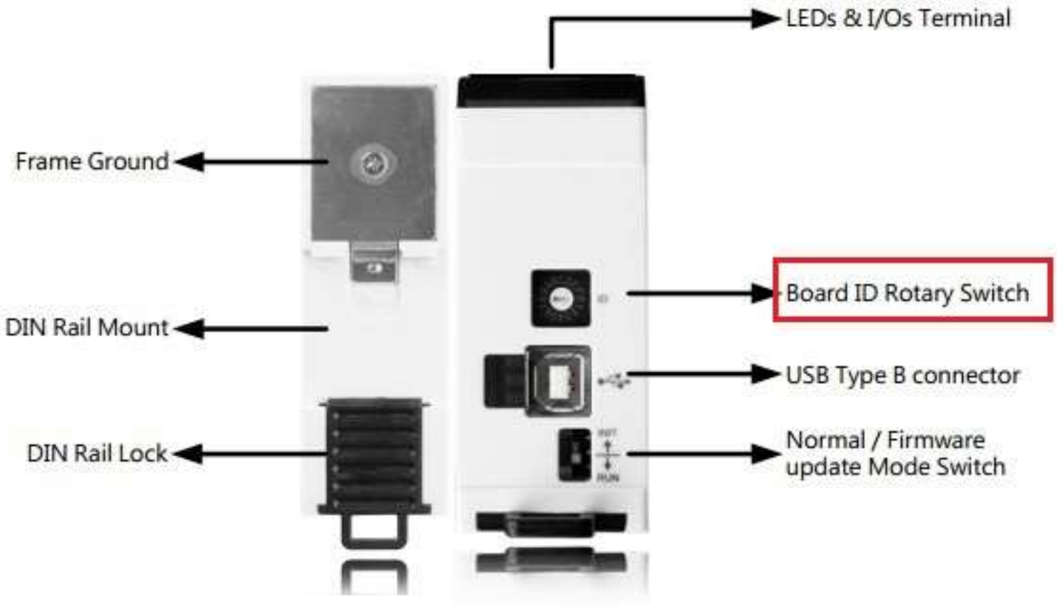
# 3.USB-2000 series Demo Program For Linux

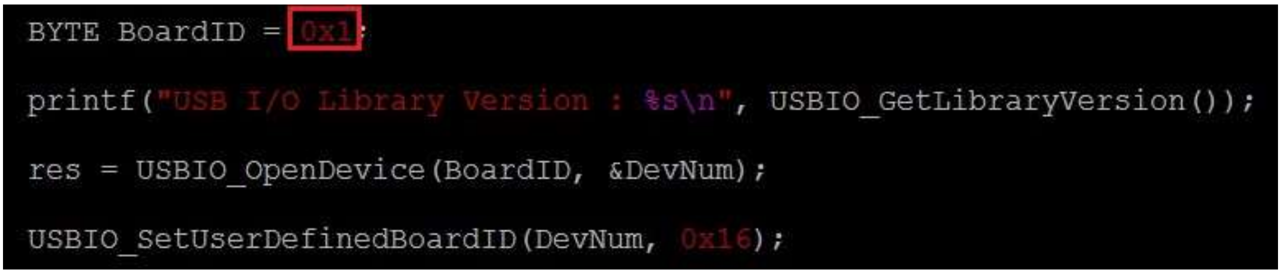
Table3.1

| Directory Path | File Name | Description |
|---|---|---|
| Include | ICPDAS_USBIO.h | The header of USB-2000 series library. |
| | | |
| lib | libUSBIO_32.a<br>libUSBIO_64.a | The USB-2000 series static lib for X86 & X64 Linux PC |
| | libUSBIO_32.so<br>libUSBIO_64.so | The USB-2000 series dynamic lib for X86 & X64 Linux PC |
| | libUSBIO_arm.a<br>libUSBIO_arm.so | The USB-2000 series lib for arm Linux PC |
| | | |
| Doc | USB-2000-Series-Linux-Manual.pdf | The linux manual for USB-2000 Series. |
| | | |
| examples | usbio_list.c | Get module list. |
| | usbio_module_info.c | Generic USB-2000 device setting. |
| | usbio_do.c | DO control. |
| | usbio_do_channel.c | DO control by channel. |
| | usbio_do-setting.c | DO module setting. |
| | usbio_di.c | Read DI value. |
| | usbio_di-setting.c | DI module setting. |
| | usbio_ai.c | Read AI value. |
| | usbio_ai-setting.c | AI module setting. |
| | usbio_ao | Ser AO value |
| | usbio_ao-setting | AO module setting |

## 3.1 Set Demo variable BoardID

You have to set variable BoardID to match rotary switch number.



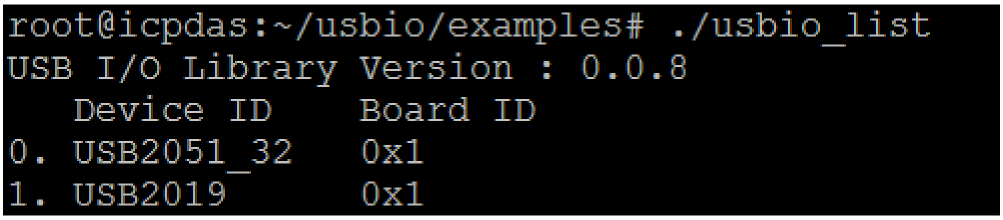


```
BYTE BoardID = 0x1;
printf("USB I/O Library Version : %s\n", USBIO_GetLibraryVersion());
res = USBIO_OpenDevice(BoardID, &DevNum);
USBIO_SetUserDefinedBoardID(DevNum, 0x16);
```

BoardID: If device rotary switch between 1 and F, then BoardID is between 0x1 and 0xF.
If device rotary switch is 0, the BoardID is user define.
User can use API USBIO_SetUserDefinedBoardID() to set BoardID, the BoardID range is between 0x10(default) to 0x7f.

## 3.2 Demo “usbio_list”

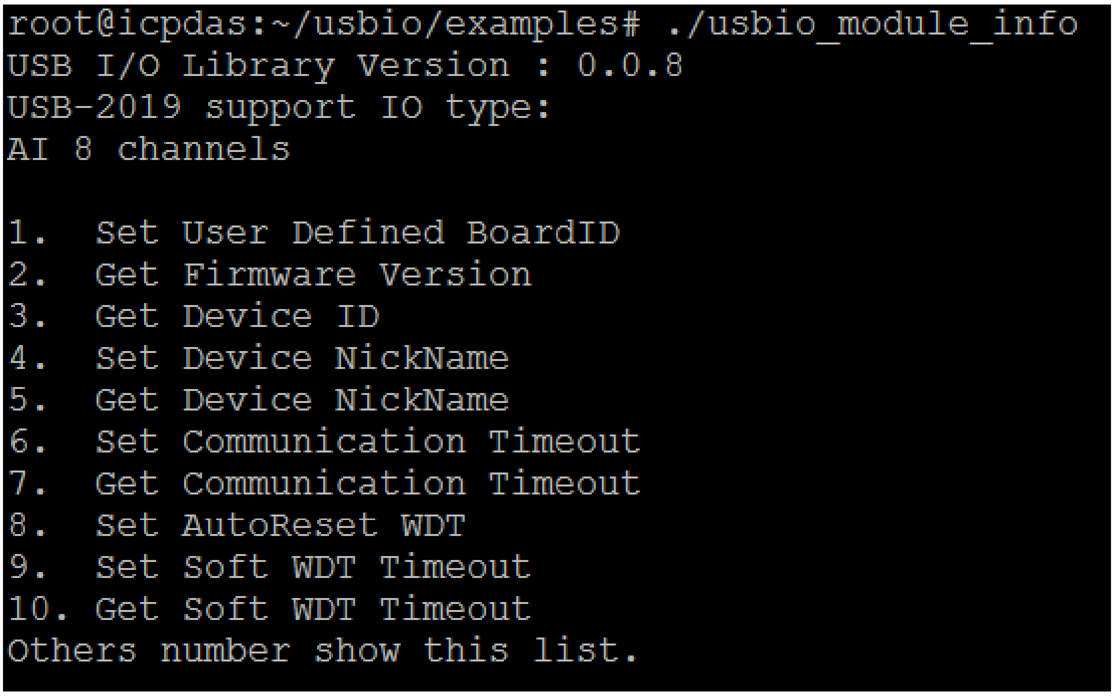
The usbio_list demo provides the module device ID and board ID, attached to the OS.



```
root@icpdas:~/usbio/examples# ./usbio_list
USB I/O Library Version : 0.0.8
   Device ID    Board ID
0. USB2051_32   0x1
1. USB2019      0x1
```

## 3.3 Demo “usbio_module_info”

The demo “usbio_module_info” can set/get generic module information.



```
root@icpdas:~/usbio/examples# ./usbio_module_info
USB I/O Library Version : 0.0.8
USB-2019 support IO type:
AI 8 channels

1.  Set User Defined BoardID
2.  Get Firmware Version
3.  Get Device ID
4.  Set Device NickName
5.  Get Device NickName
6.  Set Communication Timeout
7.  Get Communication Timeout
8.  Set AutoReset WDT
9.  Set Soft WDT Timeout
10. Get Soft WDT Timeout
Others number show this list.
```

**Set User Defined BoardID**

If you want to set Board ID with 16~127, please follow below steps

1. Turn rotary switch to 0, and check your default board id(usbio_list).

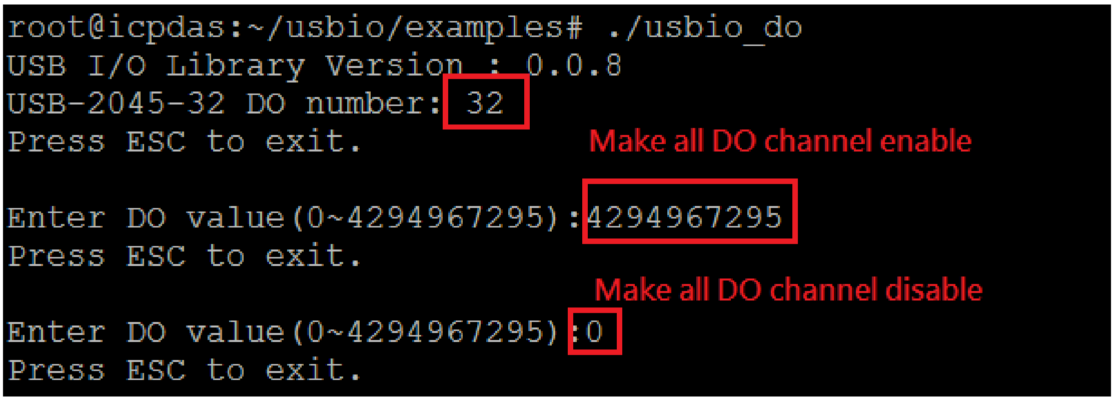
2. Open device and set new board id.

3. Power off then power on, now you used new board id.

## 3.4 Demo code “usbio_do”

The demo “usbio_do” can set DO value to control module’s DO function

This demo will show you module name and it’s DO channels
You can control all DO channel ones a time.



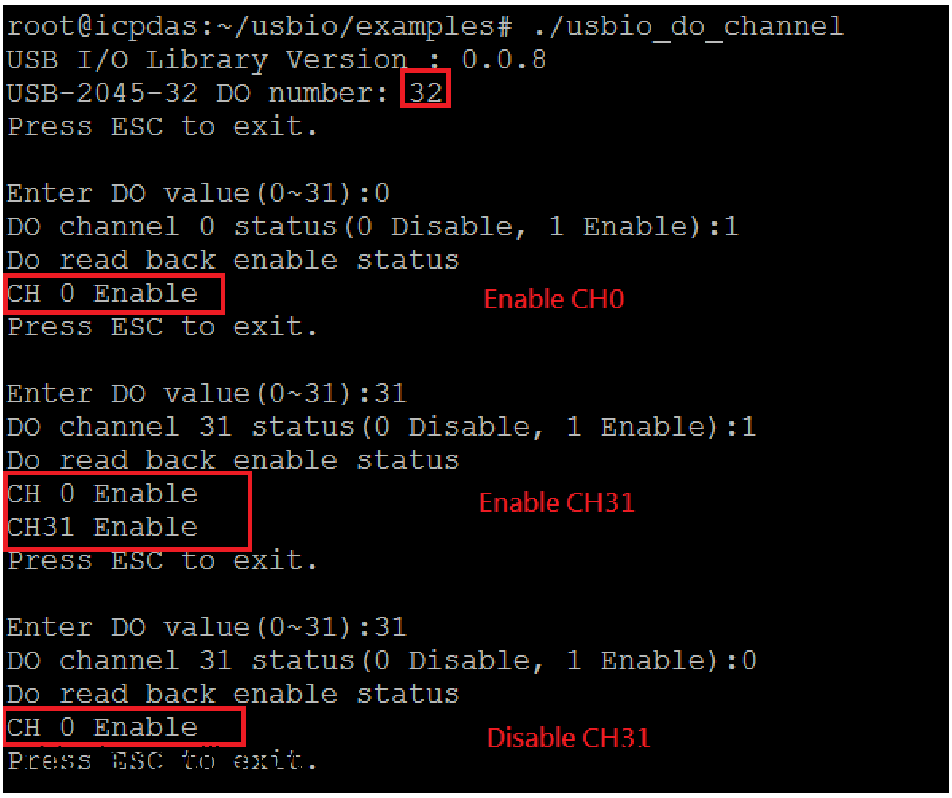
```
root@icpdas:~/usbio/examples# ./usbio_do
USB I/O Library Version : 0.0.8
USB-2045-32 DO number: 32
Press ESC to exit.

Enter DO value(0~4294967295):4294967295
Press ESC to exit.

Enter DO value(0~4294967295):0
Press ESC to exit.
```

Make all DO channel enable

Make all DO channel disable

## 3.5 Demo code “usbio_do_channel”

The demo “usbio_do_channel” can select DO channel to control module’s DO function



```
root@icpdas:~/usbio/examples# ./usbio_do_channel
USB I/O Library Version : 0.0.8
USB-2045-32 DO number: 32
Press ESC to exit.

Enter DO value(0~31):0
DO channel 0 status(0 Disable, 1 Enable):1
Do read back enable status
CH 0 Enable
Press ESC to exit.

Enter DO value(0~31):31
DO channel 31 status(0 Disable, 1 Enable):1
Do read back enable status
CH 0 Enable
CH31 Enable
Press ESC to exit.

Enter DO value(0~31):31
DO channel 31 status(0 Disable, 1 Enable):0
Do read back enable status
CH 0 Enable
Press ESC to exit.
```

Enable CH0

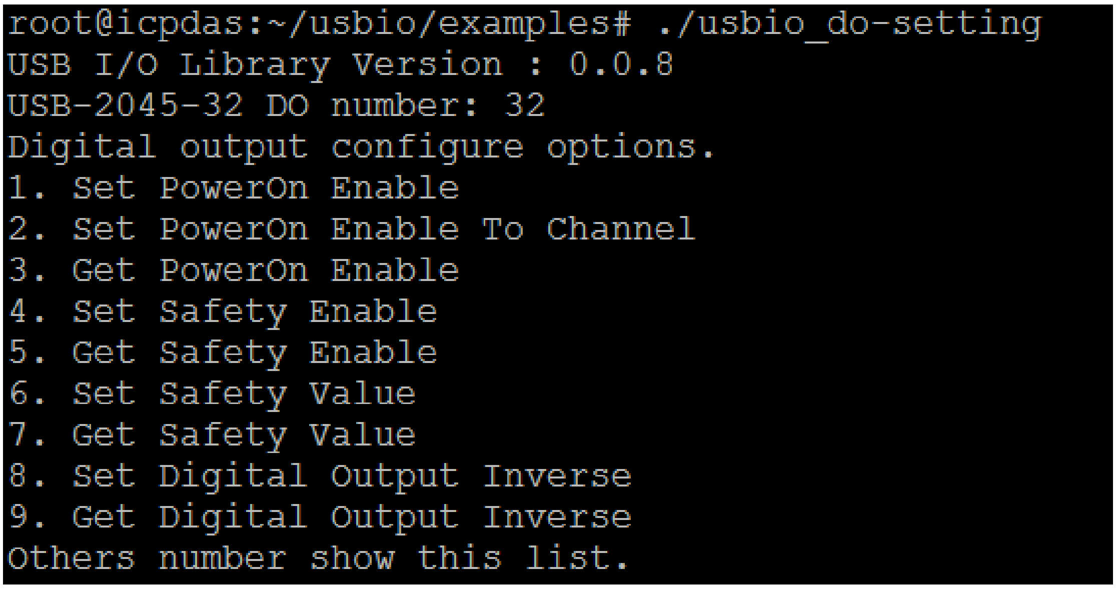
Enable CH31

Disable CH31

You can control one DO channel ones a time.

## 3.6 Demo code “usbio_do-setting”

The demo “usbio_do-setting” can set/get DO module information



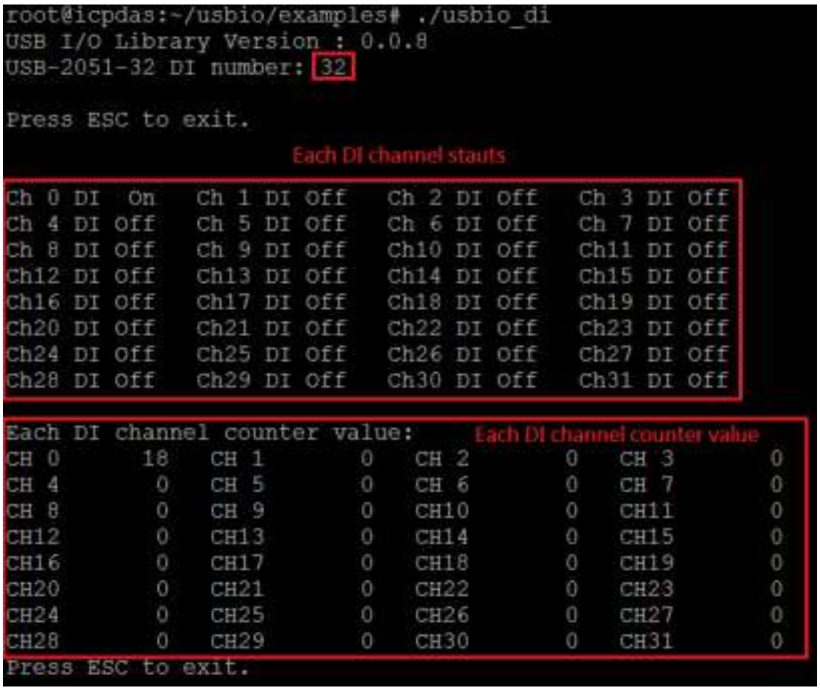
```
root@icpdas:~/usbio/examples# ./usbio_do-setting
USB I/O Library Version : 0.0.8
USB-2045-32 DO number: 32
Digital output configure options.
1. Set PowerOn Enable
2. Set PowerOn Enable To Channel
3. Get PowerOn Enable
4. Set Safety Enable
5. Get Safety Enable
6. Set Safety Value
7. Get Safety Value
8. Set Digital Output Inverse
9. Get Digital Output Inverse
Others number show this list.
```

## 3.7 Demo code “usbio_di”

The demo “usbio_di” can read DI value from module’s DI function

This demo will show you module name and all DI channels

You can check every DI channel status and counter value.



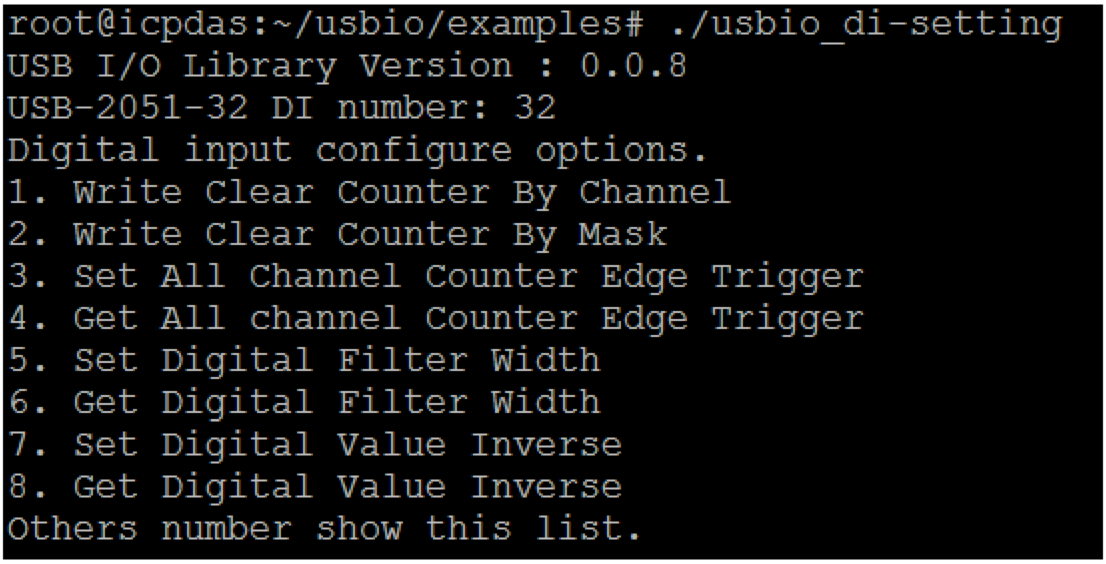
```
root@icpdas:~/usbio/examples# ./usbio_di
USB I/O Library Version : 0.0.8
USB-2051-32 DI number: 32

Press ESC to exit.

Ch 0 DI  On   Ch 1 DI Off   Ch 2 DI Off   Ch 3 DI Off
Ch 4 DI Off   Ch 5 DI Off   Ch 6 DI Off   Ch 7 DI Off
Ch 8 DI Off   Ch 9 DI Off   Ch10 DI Off   Ch11 DI Off
Ch12 DI Off   Ch13 DI Off   Ch14 DI Off   Ch15 DI Off
Ch16 DI Off   Ch17 DI Off   Ch18 DI Off   Ch19 DI Off
Ch20 DI Off   Ch21 DI Off   Ch22 DI Off   Ch23 DI Off
Ch24 DI Off   Ch25 DI Off   Ch26 DI Off   Ch27 DI Off
Ch28 DI Off   Ch29 DI Off   Ch30 DI Off   Ch31 DI Off

Each DI channel counter value:
CH 0      18   CH 1       0   CH 2       0   CH 3       0
CH 4       0   CH 5       0   CH 6       0   CH 7       0
CH 8       0   CH 9       0   CH10       0   CH11       0
CH12       0   CH13       0   CH14       0   CH15       0
CH16       0   CH17       0   CH18       0   CH19       0
CH20       0   CH21       0   CH22       0   CH23       0
CH24       0   CH25       0   CH26       0   CH27       0
CH28       0   CH29       0   CH30       0   CH31       0
Press ESC to exit.
```

Each DI channel stauts

Each DI channel counter value

## 3.8 Demo code” usbio_di-setting”

The demo “usbio_di-setting” can set/get DI module information



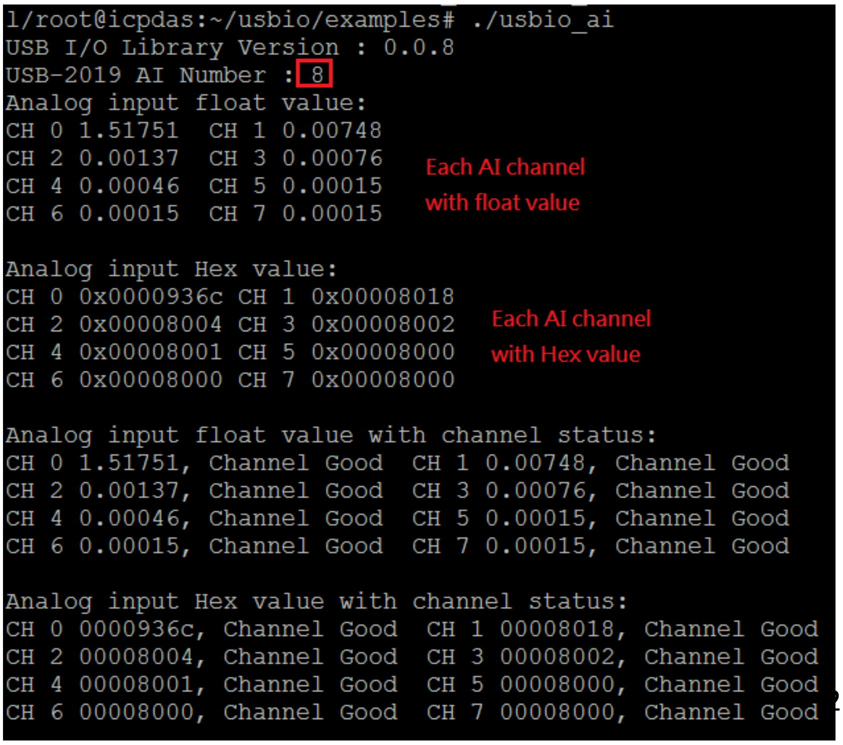
```
root@icpdas:~/usbio/examples# ./usbio_di-setting
USB I/O Library Version : 0.0.8
USB-2051-32 DI number: 32
Digital input configure options.
1. Write Clear Counter By Channel
2. Write Clear Counter By Mask
3. Set All Channel Counter Edge Trigger
4. Get All channel Counter Edge Trigger
5. Set Digital Filter Width
6. Get Digital Filter Width
7. Set Digital Value Inverse
8. Get Digital Value Inverse
Others number show this list.
```

## 3.9 Demo code “usbio_ai”

The demo “usbio_ai” can read AI value from module’s AI function

This demo will show you module name and it’s AI channels

You can read float value and hex value and channel status for each channel.



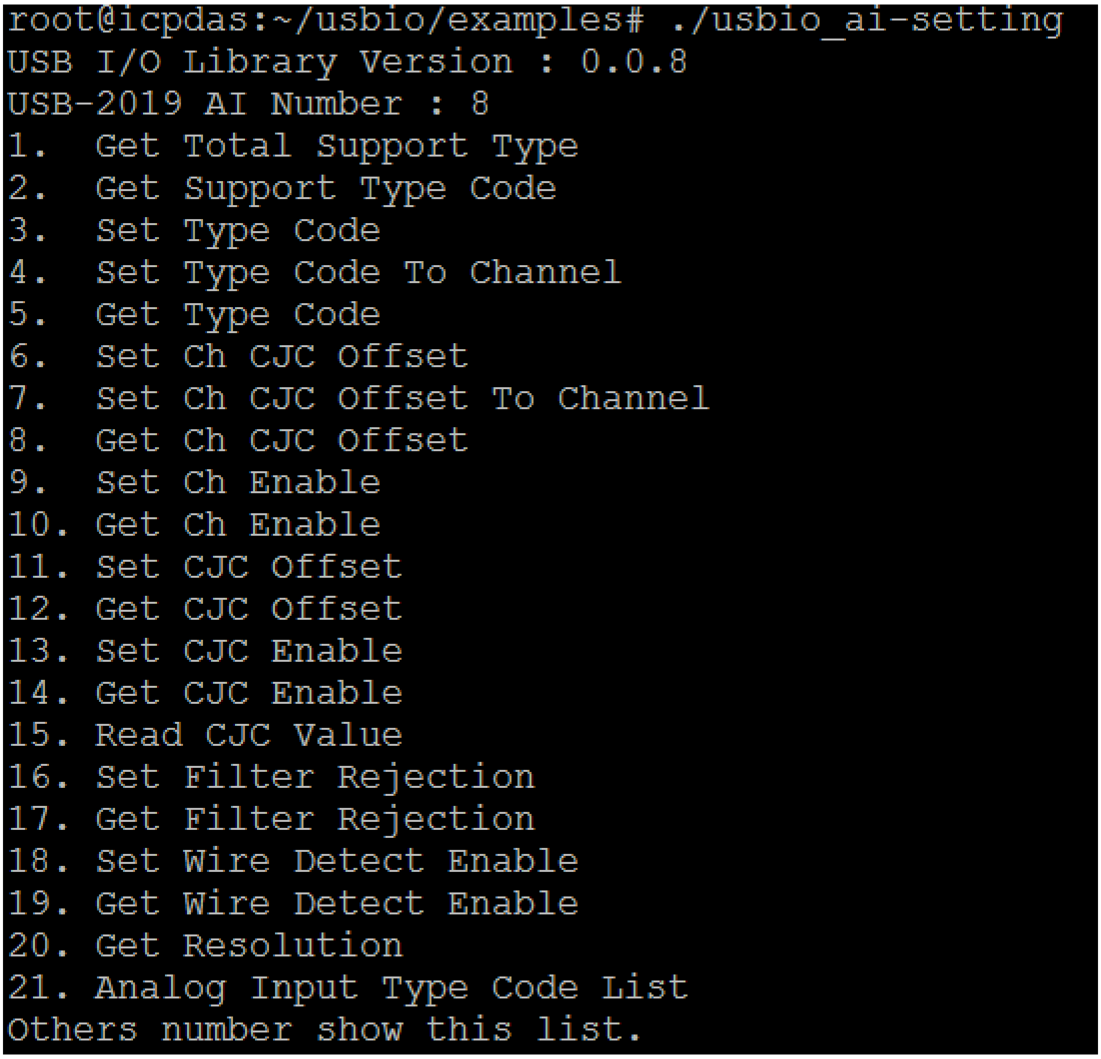
```
l/root@icpdas:~/usbio/examples# ./usbio_ai
USB I/O Library Version : 0.0.8
USB-2019 AI Number : 8
Analog input float value:
CH 0 1.51751  CH 1 0.00748
CH 2 0.00137  CH 3 0.00076
CH 4 0.00046  CH 5 0.00015
CH 6 0.00015  CH 7 0.00015

Analog input Hex value:
CH 0 0x0000936c CH 1 0x00008018
CH 2 0x00008004 CH 3 0x00008002
CH 4 0x00008001 CH 5 0x00008000
CH 6 0x00008000 CH 7 0x00008000

Analog input float value with channel status:
CH 0 1.51751, Channel Good  CH 1 0.00748, Channel Good
CH 2 0.00137, Channel Good  CH 3 0.00076, Channel Good
CH 4 0.00046, Channel Good  CH 5 0.00015, Channel Good
CH 6 0.00015, Channel Good  CH 7 0.00015, Channel Good

Analog input Hex value with channel status:
CH 0 0000936c, Channel Good  CH 1 00008018, Channel Good
CH 2 00008004, Channel Good  CH 3 00008002, Channel Good
CH 4 00008001, Channel Good  CH 5 00008000, Channel Good
CH 6 00008000, Channel Good  CH 7 00008000, Channel Good
```

Each AI channel with float value

Each AI channel with Hex value

## 3.10 Demo code “usbio_ai-setting”

The demo “usbio_ai-setting” can set/get AI module information

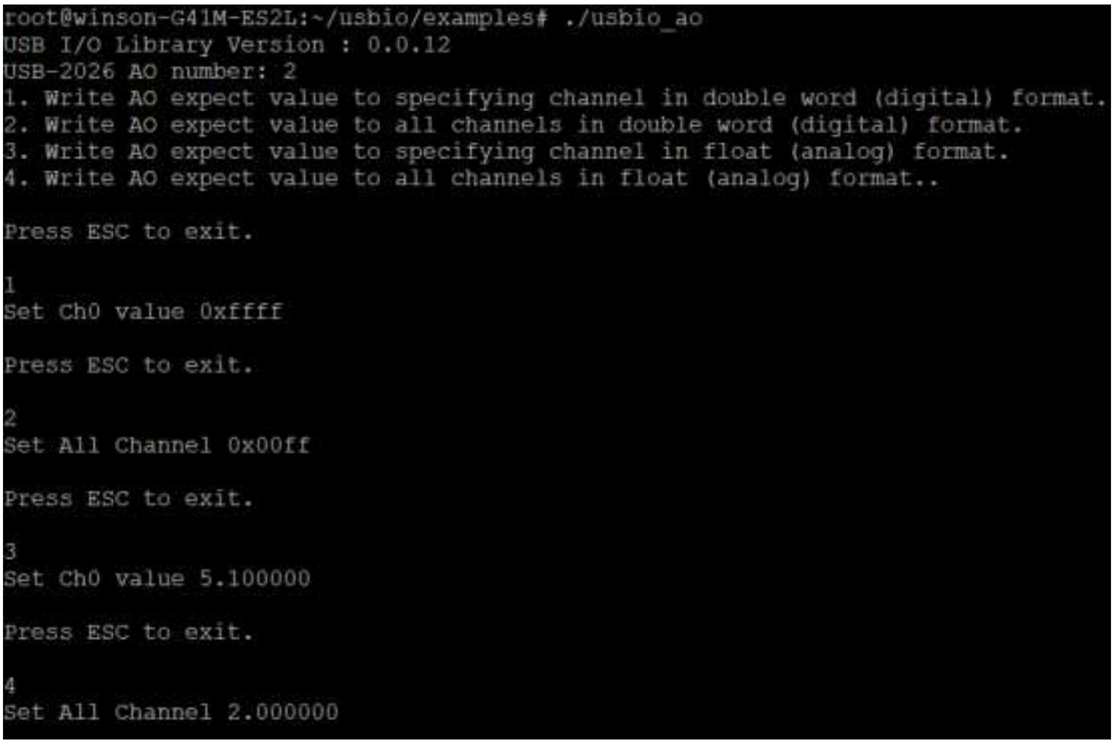


```
root@icpdas:~/usbio/examples# ./usbio_ai-setting
USB I/O Library Version : 0.0.8
USB-2019 AI Number : 8
1.  Get Total Support Type
2.  Get Support Type Code
3.  Set Type Code
4.  Set Type Code To Channel
5.  Get Type Code
6.  Set Ch CJC Offset
7.  Set Ch CJC Offset To Channel
8.  Get Ch CJC Offset
9.  Set Ch Enable
10. Get Ch Enable
11. Set CJC Offset
12. Get CJC Offset
13. Set CJC Enable
14. Get CJC Enable
15. Read CJC Value
16. Set Filter Rejection
17. Get Filter Rejection
18. Set Wire Detect Enable
19. Get Wire Detect Enable
20. Get Resolution
21. Analog Input Type Code List
Others number show this list.
```

## 3.11 Demo code “usbio_ao”

The demo “usbio_ao” can set AO value from module’s AO function

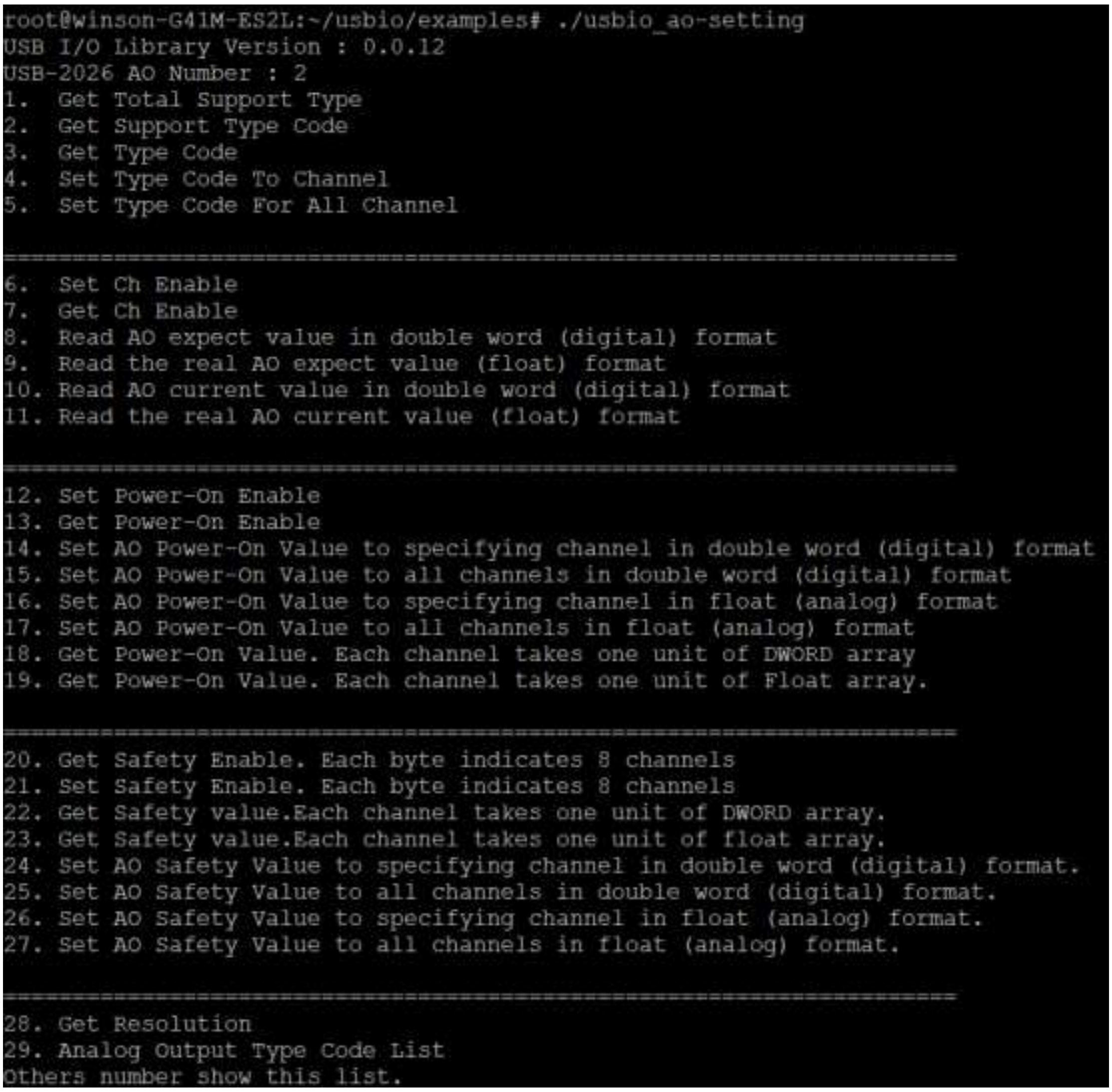
This demo will show you module name and set AO value sample.



```
root@winson-G41M-ES2L:~/usbio/examples# ./usbio_ao
USB I/O Library Version : 0.0.12
USB-2026 AO number: 2
1. Write AO expect value to specifying channel in double word (digital) format.
2. Write AO expect value to all channels in double word (digital) format.
3. Write AO expect value to specifying channel in float (analog) format.
4. Write AO expect value to all channels in float (analog) format..

Press ESC to exit.

1
Set Ch0 value 0xffff

Press ESC to exit.

2
Set All Channel 0x00ff

Press ESC to exit.

3
Set Ch0 value 5.100000

Press ESC to exit.

4
Set All Channel 2.000000
```

## 3.12 Demo code “usbio_ao-setting”

The demo “usbio_ao-setting” can set/get AO module information



```
root@winson-G41M-ES2L:~/usbio/examples# ./usbio_ao-setting
USB I/O Library Version : 0.0.12
USB-2026 AO Number : 2
1.  Get Total Support Type
2.  Get Support Type Code
3.  Get Type Code
4.  Set Type Code To Channel
5.  Set Type Code For All Channel

=====================================================================
6.  Set Ch Enable
7.  Get Ch Enable
8.  Read AO expect value in double word (digital) format
9.  Read the real AO expect value (float) format
10. Read AO current value in double word (digital) format
11. Read the real AO current value (float) format

=====================================================================
12. Set Power-On Enable
13. Get Power-On Enable
14. Set AO Power-On Value to specifying channel in double word (digital) format
15. Set AO Power-On Value to all channels in double word (digital) format
16. Set AO Power-On Value to specifying channel in float (analog) format
17. Set AO Power-On Value to all channels in float (analog) format
18. Get Power-On Value. Each channel takes one unit of DWORD array
19. Get Power-On Value. Each channel takes one unit of Float array.

=====================================================================
20. Get Safety Enable. Each byte indicates 8 channels
21. Set Safety Enable. Each byte indicates 8 channels
22. Get Safety value.Each channel takes one unit of DWORD array.
23. Get Safety value.Each channel takes one unit of float array.
24. Set AO Safety Value to specifying channel in double word (digital) format.
25. Set AO Safety Value to all channels in double word (digital) format.
26. Set AO Safety Value to specifying channel in float (analog) format.
27. Set AO Safety Value to all channels in float (analog) format.

=====================================================================
28. Get Resolution
29. Analog Output Type Code List
Others number show this list.
```

# Appendix

## 1.Analog Input Type Code

| 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 LDIN43710 TC, -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 |
| 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 | | |

## 2.Analog Output Type Code

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

## 3.Pulse Input Type Code

| 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 |

## 4.Channel Status

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