

**USB**  
**Dynamic Industrial Interface**  
**V 2.0.0.3**  
**A Universal**  
**Application Programming Interface**  
**To**  
**Data Acquisition Products**  
**Users Manual**

Design & Implementation by  
Decision Computer International Company

No parts of this documentation may be reproduced or transmitted in any form, by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of Decision Computer International Company.

## ***Contents***

1. Introduction .....	3
2. Features .....	4
3. Device Type definition .....	5
4. Data Types of Function calls.....	6
5. Functions to open and close Devices .....	7
6. Functions for digital input/output .....	10
7. Functions for reset hardware device .....	16
8. Functions for analog input/output .....	17
9. Using USB Dll with different programming language .....	18
9.1. C++.....	18
9.2 Visual Basic .....	18
10. Technical support and Feedback .....	18

## ***1. Introduction***

This document provides the USB Dynamic Industrial Interface Specifications, including all function calls, and operating procedures.

### **Disclaimer:**

Decision Computer International Company (DECISION) cannot take responsibility for consequential damages caused by using this software. In no event shall DECISION be liable for any damages whatsoever (including, without limitation, damages for loss of business profits, business interruption, loss of business information, or any other pecuniary loss) arising out of the use of or inability to use this product, even if we have been advised of the possibility of such damages.

### **Trademark Acknowledgments:**

Windows 98, Windows ME, Windows 2000, Windows XP, Windows 7, Visual Basic, Visual C++ are registered trademarks of Microsoft Corporation.

## **2. Features**

The USB Dynamic Industrial Interface (USBDII) was created to provide a standard way to access the functionality provided by all USB data acquisition products. Specifically, the USBDII provides the following features:

- Platform-independent

The library is compatible under Windows 98, Windows ME, Windows 2000, windows XP, Vista, and Win7. The compatibility under these operation systems guarantees that programs written for either operating system will work unchanged on the other, even without recompilation.

- Abstracts Card Functionality from Card Design

The interface concentrates on a card's functionality and hides the user from having to know specifics about the card design, for example, which port needs to be accessed in order to access specific functionality. All details of the card implementation are hidden from the user.

- Multiple Device Support

You could access device by its name or by its information (device type, id index).

- Programming Language Independent

The library provides a language independent way to access the USB industrial I/O cards, by using a Dynamic-Link-Library architecture.

### ***3. Device Type Definition***

Below are names for device types and its' corresponding defined value:

USB_16PIO	0x01	// USB 16 Channel Photo Input / 16 Channel Photo Output Board
USB_LABKIT	0x02	// USB LABKIT
USB_16PR	0x03	// USB 16 Channel Photo Input / 16 Channel Relay Output Board
USB_STARTER	0x04	// USB STARTER
USB_8PR	0x06	// USB 8 Channel Photo Input / 8 Channel Relay Output Board
USB_4PR	0x07	// USB 4 Channel Photo Input / 4 Channel Relay Output Board
USB_8PI	0x08	// USB 8 Channel Photo Input Board
USB_8RO	0x09	// USB 8 Channel Relay Output Board
USB_16PI	0x0A	// USB 16 Channel Photo Input Board
USB_16RO	0x0B	// USB 16 Channel Relay Output Board
USB_32PI	0x0C	// USB 32 Channel Photo Input Board
USB_32RO	0x0D	// USB 32 Channel Relay Output Board

#### ***4. Data Types of Function calls***

Since the USBDI was developed in the C++ language, some data types used may not be present in the programming language you want to use. Please find the following data type conversion table for your convenience:

HANDLE	An opaque 32-bit integer
BYTE	A 8-bit unsigned integer
BOOL	A 32-bit integer, either 0 (FALSE) or 1 (TRUE)
DWORD	A 32-bit unsigned integer
HWND	A 32-bit integer representing a valid handle to a Window
LPTSTR	A 32-bit flat pointer to a zero terminated string
LPBOOL	A 32-bit flat pointer to a variable of type BOOL
LPBYTE	A 32-bit flat pointer to a variable of type BYTE
LPDWORD	A 32-bit flat pointer to a variable of type DWORD

Also note that the DLL employs the Standard Call (Pascal) calling mechanism, which is used for all system. USBDI as well and is compatible with VB, VC, Delphi, .NET, and notice the variable with same type name may have different define in different program language. For example, in Visual Basic 6, the width of ***Integer*** is 16 bits and the width of ***Long*** is 32 bits, but in Visual Basic.Net, the width of ***Integer*** becomes 32 bits and the width of ***Long*** becomes 64 bits. If you declare variable with different width from our define, it may cause some run-time error.

## ***5. Functions to open and close Devices***

### **hid\_OpenDevice**

This function opens a device for further access by USB.

#### Declaration

```
HANDLE hid_OpenDevice ( DWORD device_type,  
                        DWORD device_id );
```

#### Parameters

*device\_type* The type of the device to open.

*device\_id* Device's id on the Board.

For more information, please see “Device Type Table & ID Table” following below.

#### Return value

A valid handle representing the device, or INVALID\_HANDLE\_VALUE (-1) if an error occurred.

For USB\_STARTER, there is no ID selection and device\_id = 0

#### Example

```
HANDLE hDevice = hid_OpenDevice(Device Type, Device Index);  
if (hDevice == INVALID_HANDLE_VALUE)  
{  
    MessageBox (NULL,“Open Failed!“,“Error“,MB_OK);  
}
```

### **hid\_CloseDevice**

This function closes a device by USB.

#### Declaration

```
BOOL hid_CloseDevice (HANDLE hDevice)
```

#### Parameters

*hDevice* A valid device handle.

#### Return value

TRUE if successful, FALSE otherwise.

#### Example

```
hid_CloseDevice(hDevice);
```

### **com\_OpenDevice**

This function opens a device for further access by Serial Port.

#### Declaration

```
HANDLE com_OpenDevice ( DWORD device_type,  
                        DWORD device_id,  
                        DWORD port_num );
```

#### Parameters

*device\_type* The type of the device to open.

*device\_id* Device's id on the board.

For more information, please see "Device Type Table & ID Table" following below.

*port\_num* Com Port Num to open.

#### Return value

A valid handle representing the device, or INVALID\_HANDLE\_VALUE (-1) if an error occurred.

#### Example

```
HANDLE hDevice = com_OpenDevice(Device Type, Device Index, 1);  
if (hDevice == INVALID_HANDLE_VALUE)  
    MessageBox (NULL, "Open Failed!", "Error", MB_OK);
```

### **com\_CloseDevice**

This function closes a device by Serial Port.

#### Declaration

```
BOOL com_CloseDevice(HANDLE hDevice)
```

#### Parameters

*hDevice* A valid device handle.

#### Return value

TRUE if successful, FALSE otherwise.

#### Example

```
com_CloseDevice(hDevice);
```

#### Remarks

Please see "Serial\_Communication.pdf" to set hardware for serial communication, and USB\_LABKIT, USB\_STARTER, USB\_8PR are not supported by serial communication.



**Device Type Table**

<i>Product</i>	<i>device_type</i>
USB_16PIO	0x01
USB_LABKIT	0x02
USB_16PR	0x03
USB_STARTER	0x04
USB_8PR	0x06
USB_4PR	0x07
USB_8PI	0x08
USB_8RO	0x09
USB_16PI	0x0A
USB_16RO	0x0B
USB_32PI	0x0C
USB_32RO	0x0D

**Device ID Table**

( Switch Setting on the Device Board )



<i>Switch Setting</i>	<i>device_id</i>
1, 2, 3, 4 OFF	0
2, 3, 4 OFF, 1 ON	1
1, 3, 4 OFF, 2 ON	2
3, 4 OFF, 1, 2 ON	3
1, 2, 4 OFF, 3 ON	4
2, 4 OFF, 1, 3 ON	5
1, 4 OFF, 2, 3 ON	6
4 OFF, 2, 3, 4 ON	7
1, 2, 3 OFF, 4 ON	8
2, 3 OFF, 1, 4 ON	9
1, 3 OFF, 2, 4 ON	10
3 OFF, 1, 2, 4 ON	11
1, 2 OFF, 3, 4 ON	12
2 OFF, 1, 3, 4 ON	13
1 OFF, 2, 3, 4 ON	14
1, 2, 3, 4 ON	Firmware update

## ***6. Functions for digital input/output***

### **hid\_SetDigitalByte**

This function sets or clears a byte on a digital output line by USB.

#### Declaration

```
BOOL  hid_SetDigitalByte ( HANDLE hDevice,
                          DWORD dwPort,
                          BYTE byPortState
                          );
```

#### Parameters

*hDevice*        A valid device handle, previously obtained from hid\_OpenDeviceDevice  
*dwPort*        The index of the port on the card to manipulate. The first port has index 0.  
                 For more information, please see “Write Address Table” following below.  
*byPortState*   The new state of the port

#### Return value

TRUE if successful, FALSE otherwise.

If an error occurred, GetLastError() may return the following values:

ERROR\_INVALID\_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

#### Example

```
HANDLE hDevice = hid_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
    hid_SetDigitalByte( hDevice, 0, 0xFF); // set's all bits on the first port
    hid_CloseDevice(hDevice);
}
```

## **com\_SetDigitalByte**

This function sets or clears a byte on a digital output line by Serial Port.

### Declaration

```
BOOL  com_SetDigitalByte ( HANDLE hDevice,
                           DWORD dwPort,
                           BYTE byPortState
                           );
```

### Parameters

*hDevice*        A valid device handle, previously obtained from com\_OpenDevice

*dwPort*        The index of the port on the card to manipulate. The first port has index 0.  
For more information, please see “Write Address Table” following below.

*byPortState*   The new state of the port

### Return value

TRUE if successful, FALSE otherwise.

If an error occurred, GetLastError() may return the following values:

ERROR\_INVALID\_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

### Example

```
HANDLE hDevice = com_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
    com_SetDigitalByte( hDevice, 0, 0xFF); // set's all bits on the first port
    com_CloseDevice(hDevice);
}
```

### Remarks

Please see “Serial\_Communication.pdf” to set hardware for serial communication, and USB\_LABKIT, USB\_STARTER, USB\_8PR are not supported by serial communication.

**Write Address Table**

<i>Product</i>	<i>dwPort</i>	<i>Content</i>
USB_16PIO	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_LABKIT	0x03	P1D07 to P1D00
USB_STARTER	0x03	P1D07 to P1D00
USB_16PR	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_8PR	0x01	OUT07 to OUT00
	0x02	DIO7 to DIO0
	0x03	DIO15 to DIO8
USB_4PR	0x02	OUT03 to OUT00
USB_8RO	0x02	OUT07 to OUT00
USB_16RO	0x02	OUT07 to OUT00
	0x03	OUT15 to OUT08
USB_32RO	0x00	OUT07 to OUT00
	0x01	OUT15 to OUT08
	0x02	OUT23 to OUT16
	0x03	OUT31 to OUT24

## hid\_GetDigitalByte

This function reads a complete byte from a digital input port of a device by USB.

### Declaration

```
BOOL  hid_GetDigitalByte ( HANDLE hDevice,
                          DWORD dwPort,
                          LPBYTE lpbyPortState
                          );
```

### Parameters

*hDevice*            A valid device handle, previously obtained from hid\_OpenDeviceDevice

*dwPort*            The index of the port on the card to manipulate. The first port has index 0.  
For more information, please see “Read Address Table” following below.

*lpbyPortState*    A pointer to a variable of type BYTE receiving the new state of the port

### Return value

TRUE if successful, FALSE otherwise.

If an error occurred, GetLastError() may return the following values:

ERROR\_INVALID\_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

### Example

```
HANDLE hDevice = hid_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
    hid_GetDigitalByte( hDevice, 0, &byState); // reads the state of the first input port
    hid_CloseDevice(hDevice);
}
```

## **com\_GetDigitalByte**

This function reads a complete byte from a digital input port of a device by Serial Port.

### Declaration

```
BOOL  com_GetDigitalByte ( HANDLE hDevice,
                          DWORD dwPort,
                          LPBYTE lpbyPortState
                          );
```

### Parameters

*hDevice*            A valid device handle, previously obtained from com\_OpenDevice

*dwPort*            The index of the port on the card to manipulate. The first port has index 0.  
For more information, please see “Read Address Table” following below.

*lpbyPortState*    A pointer to a variable of type BYTE receiving the new state of the port

### Return value

TRUE if successful, FALSE otherwise.

If an error occurred, GetLastError() may return the following values:

ERROR\_INVALID\_PARAMETER - The handle passed was invalid, or the port number was out of range for the device selected.

### Example

```
HANDLE hDevice = com_OpenDevice(0x01,0);
if (hDevice != INVALID_HANDLE_VALUE)
{
    com_GetDigitalByte( hDevice, 0, &byState); // reads the state of the first input port
    com_CloseDevice(hDevice);
}
```

### Remarks

Please see “Serial\_Communication.pdf” to set hardware for serial communication, and USB\_LABKIT, USB\_STARTER, USB\_8PR are not supported by serial communication.

**Read Address Table**

<i>Product</i>	<i>dwPort</i>	<i>Content</i>
USB_16PIO	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_LABKIT	0x02	P0D07 to P0D00
USB_STARTER	0x02	P0D07 to P0D00
USB_16PR	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_8PR	0x00	IN07 to IN00
	0x02	DIO7 to DIO0
	0x03	DIO15 to DIO8
	<b>0x10</b>	<b>JP9/JP10 Settings</b>
USB_4PR	0x00	IN03 to IN00
USB_8PI	0x00	IN07 to IN00
USB_16PI	0x00	IN07 to IN00
	0x01	IN15 to IN08
USB_32PI	0x00	IN07 to IN00
	0x01	IN15 to IN08
	0x02	IN23 to IN16
	0x03	IN31 to IN24

Remarks

In **USB\_8PR**, we provide 2 digital ports for user to define either as input or output. It can be defined by Jumper 10 and Jumper 11 on the board. And we can use hid\_GetDigitalByte / com\_GetDigitalByte function to read Jumper State to determine witch port is either input or output.

hid\_GetDigitalByte( hDevice, 0x10, &byState); // or use com\_GetDigitalByte for serial communication

When JP9 is closed, DIO7 - DIO0 is for Input.	The fifth bit of byState is 0
When JP9 is opened, DIO7 - DIO0 is for Output.	The fifth bit of byState is 1
When JP10 is closed, DIO15 – DIO8 is for Input.	The sixth bit of byState is 0
When JP10 is opened, DIO15 – DIO8 is for Output.	The sixth bit of byState is 1

## ***7. Functions for reset hardware device***

### **hid\_ResetHW**

This function directly resets the hardware device by USB. And all channels on the board will load default value. If you need to control the device again, please use hid\_open to get the handle again.

#### Declaration

BOOL      hid\_ResetHW(HANDLE hDevice)

#### Parameters

*hDevice*      A valid device handle.

#### Return value

TRUE if successful, FALSE otherwise.

#### Example

```
hid_ResetHW (hDevice);
```

### **com\_ResetHW**

This function directly resets the hardware device by Serial Port. And all channels on the board will load default value.

#### Declaration

BOOL      com\_ResetHW(HANDLE hDevice)

#### Parameters

*hDevice*      A valid device handle.

#### Return value

TRUE if successful, FALSE otherwise.

#### Example

```
com_ResetHW(hDevice);
```



## ***8. Functions for analog input/output***

### **hid\_GetAnalogChannel**

This function reads a complete word from an analog input port of a device by USB.

#### Declaration

```
BOOL  hid_GetAnalogChannel ( HANDLE hDevice,
                             DWORD dwPort,
                             LPDWORD lpdwPortState
                             );
```

#### Parameters

*hDevice*            A valid device handle, previously obtained from `hid_OpenDeviceDevice`  
*dwPort*            The index of the port on the card to manipulate. The first port has index 0.  
*lpdwPortState*    A pointer to a variable of type `DWORD` receiving the new state of the port

#### Return value

TRUE if successful, FALSE otherwise.

If an error occurred, `GetLastError()` may return the following values:

`ERROR_INVALID_PARAMETER` - The handle passed was invalid, or the port number was out of range for the device selected.

#### Example

```
HANDLE hDevice = hid_OpenDevice(0x02,0); // USB_LABKIT
if (hDevice != INVALID_HANDLE_VALUE)
{
    hid_hid_GetAnalogChannel ( hDevice, 0, &dwState); // reads the state of the first analog input port
    hid_CloseDevice (hDevice);
}
```

#### Remarks

This function now only enable in **USB\_LABKIT** and **USB\_STARTER** device. The range of *dwPort* is from 0~7.

## ***9. Using the Dynamic Industrial Interface with different programming languages***

This chapter provides an overview about how to best utilize the Dynamic Industrial Interface in various programming languages.

If you experience difficulties calling the Dynamic Industrial Interface functions from your programming language, or are using a programming language not covered in this documentation, please feel free to visit our web-site, to which we will post updated information regarding DII programming issues. You may also contact our technical support through our website: [www.decision.com.tw](http://www.decision.com.tw)

### ***9.1. C++***

Since the DII DLL was developed using C++, you may easily use it to access Industrial I/O devices. For this purpose, a C++ header file ("USBDII.h") as well as an import library ("USBDII.lib") are being shipped with the interface library. Make sure that you have installed the development release, not the retail release, which does not include support programming files. In your C/C++ source code files, just include the "USBDII.h" include file, then you can use any of the functions provided by the USBDII DLL. Be sure to include the import library "USBDII.lib" during the linking step of your application. So your applications successfully references the actual interface DLL.

### ***9.2. Visual Basic***

Since the Dynamic Industrial Interface is fully 32-bit compliant, only 32-bit versions of Visual Basic are supported. Specifically, Version 6.0 are tested and supported. If you are using Visual Basic to access any I/O Devices supported by the USB Dynamic Industrial Interface (USBDII), you can call the USBDII DLL directly. But before that, you should import them. You may also consult the Visual Basic sample application for more information about using Visual Basic to access the USB Dynamic Industrial Interface (USBDII).

## ***10. Technical Support and Feedback***

We believe that customer input is the most valuable source for creating successful products. We continuously update and extend the Dynamic Industrial Interface with new functionality, for specific devices, for specific applications, to meet your specific needs, and provide supportive products around the USBDII. You may also contact our technical support through our website: [www.decision.com.tw](http://www.decision.com.tw)