Dynamic Industrial Interface
V 3.0

A Universal
Application Programming Interface
To
Data Acquisition Products

Users Manual

Design & Implementation by
Decision Computer International Company

Contents

1. Introduction

This document provides the Dynamic Industrial Interface Specifications, including all function calls, installation requirements, 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.

2. Features

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

- Platform-independent
  The library is compatible under Windows 98, windowsME, Windows 2000 and windows XP. 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.
  This also means that the ISA 16 Relay card is to be programmed exactly like the ADDIO 16 Relay output card. All details of the ADDIO implementation are hidden from the user.

- Multiple Card Support
  You could access device by its name or by its information (device type, index).

- Device Naming
  Each device can be given it's own name, specified by the function "Device Naming" on DII Control Panel Applet. This allows easier access of devices and cards in situations, where multiple cards of the same type are installed.

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

3. Distribution Contents

The Dynamic Industrial Interface Library Distribution consists of the following components:

1. A Control Panel Applet to view the device information and for device naming/renaming.

2. WDM Device Driver for Windows 98, ME, 2000, XP.

3. A Win32 DLL (file Dii.dll) for accessing the driver. In order to access the DLL, a C-Header file and a visual Basic file containing the function declarations are included.

4. The Visual C++ sample applications
   The Sample applications demonstrate the use of the driver API by graphically.The applications runs unchanged under Windows 98/ME/2k/XP. They were developed using Visual C++ 6.0. The full source codes are included.

   After installation, you can find a folder “Dii Test Programs” under your installation directory, and all sample applications are in it (includes VC++ and VB samples).

5. The Visual Basic sample applications
   The Visual Basic samples are similar to the Visual C++ sample, except it’s written using Visual Basic Version 6.0.

   After installation, you can find a folder “Dii Test Programs” under your installation directory, and all sample applications are in it (includes VC++ and VB samples).

4.Install and Uninstall

This chapter specify installation and uninstalltion for DDI software and Decision device. DDI (Decision Device Interface) software contains two kinds of library, DII and DecLib, and users could choose each of them during installing DDI software (Notice that DII and DecLib can not been mix-use).

4.1 DDI (Decision Device Interface) Software Installation:

Step 1: Execute the DDI install program (DDI_130.exe)

Step 2: Following the specifications of DDI install program. During installtion, program would query which library you use. To choose your favorite library and continue to finish installing.

After installation, you could find folder “Decision Device Interface 3.0” on [Start] -> [Programs].

4.2 DDI Software Uninstallation:

You could uninstall by following two methods:

Method 1: click [Start] -> [Programs] -> [Decision Device Interface 3.0] -> Remove Decision Device Interface 3.0

Method 2: On the “Add/Remove Programs” on Control Panel, You could find “Decision Device Interface 3.0”, and remove it.

4.3 Device Installation:

Step1: Plug your card on empty PCI or ISA slot, and boot.

Step2: When entering operation system, hardware wizard would find your card automatically

Step3: following the hardware wizard to continue installation. During installing, you might specify your driver location automatically.

Step4: After installing driver, computer may be reboot. And you would see youir device on “Device Manager”.

4.4 Device Unnstallation:

Step1: On [Control Panel] -> [System] -> [Hardware] -> [Device Manager], choose the device you would like to uninstall and click right mouse button --> uninstall.

Step2: Under the folder “C:\Windows\inf” (2k/XP is on “C:\WinNt\inf”), you could find some oem#.inf files (# represent digits). To delete these oem#.inf files that contains your card information (open these .inf as test program).

Step3: Under the folder “C:\Windows\system32\drivers” (2k/XP is on “C:\WINNT\system32\drivers”), you could find driver for your card, delete them. (file name: XXXX.sys, XXXX represent your device name).

5. Device Naming

The Dynamic Industrial Interface (DII) has a powerful feature: Device Naming.

After the installation of devices, each device is assign a default unique name. And you could view or fix device's name on the DII Control Panel Applet. When creating your application program, you can then access that device using the name, or you may prompt the user for the name, or even let the user graphically browse for the correct device for your application.

This technique has one important feature:

- More clarity when using multiple devices
  Using device names, you can now access more easily and clearly multiple devices of the same type in your machine. For example, if you have 3 pieces of 8 Relay 8 Photo Isolator cards in your machine, you could access these cards not only by their type and index, which often lead to confusion. You can now just assign a name to each card on DII Control Panel Applet, especially a meaningful name.

There is something to care when using DII Control Panel Applet for naming/renaming:

- Please reboot your computer after you change device's name or the device can't work.
- You might name/rename your device only using characters, digits, space and underscore. Others might cause problem.
- Don't name/rename the same name to different devices or it would cause problem.
- The maximum length of device name is 100 bytes (100 characters).

6. Backward compatibility

This new version of Dynamic Industrial Interface (DII 3.0) would good backward compatibility feature. You won't change any piece of your old program codes (but something would be concerned, list below), because the interface of the new-version DII is the same to that of the old-version one. You would only replace new file DII.DLL and DII.h with old ones, and then your programs could work well as before.

Note.. there are some original DII function not support on this version of DII yet. Like notification, block, pulse counter, and some parts of AD/DA, because they are usually used on ISA card (of course, all ISA cards don't support yet). But all basic functions like I/O access, AD/DA convertion, timer handle, are implemented and fix some bug before. On later update, we would add them.

Some differences to old version of DII:

- Internal implementation of some functions is changed, but the interfaces of all functions are the same to old-version DII.
- The meaning of device index is not the same to the past. The older one represents its bus position, but now is another. You don't need to know what it means. The only thing you needs to know is that device index of any installed device is unique and you could check it on Device Information of DII Control Panel Applet.
- Dii Control Panel Applet has some change.

7. Device Type Definition

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

// ISA card (not support yet)
RD_SMARTLAB_16CHANNEL 0x001
RD_SMARTLAB_8CHANNEL 0x002
RD_ICC_8RELAY8ISOLATOR 0x003
RD_ICC_16RELAY 0x004
RD_ICC_16ISOLATOR 0x005
RD_ICC_8SSR8LOGIC 0x006
RD_TTL 0x007
RD_IBC_32ISOLATOR 0x009
RD_ADVANCE_ADDA 0x100
RD_SUPER_12BIT_ADDA 0x101
RD_12BIT_ADDA 0x102
RD_8CHANNEL_DA 0x103
RD_SUPER_14BIT_ADDA 0x104

// PCI card
RD_PCI_IND_4CHANNEL 0x200
RD_PCI_4RELAY4ISOLATOR 0x201
RD_PCI_8RELAY8ISOLATOR 0x202
RD_PCI_16RELAY16ISOLATOR 0x203
RD_PCI_8255 0x204
RD_PCI_MULTI_8255 0x205
RD_PCI_12BIT_ADDA 0x206
RD_PCI_14BIT_ADDA 0x207

8. Dynamic Industrial Interface function calls

Since the DII 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 Dlls as well and is compatible with Visual Basic.

8.1. Functions to open and close Devices

DiiOpenDevice

This function opens a device for further access.

Declaration

```
HANDLE DiiOpenDevice (      DWORD dwDeviceType,
                            DWORD dwIndex,
                            BOOL bExclusive
                     );
```

Parameters

dwDeviceType The type of the device to open. For more information, please see the chapter “Device Type Definition”.

dwIndex device's index. For more information, please see the chapter “Backward Compatibility”. You could find device's index on Decision Control Panel Applet.

bExclusive Determines, whether to open the device exclusively (TRUE), or allow other applications to use the device, as well.

Return value

A valid handle representing the device, or INVALID_HANDLE_VALUE (-1) if an error occurred.

Example

```
HANDLE hDevice = DiiOpenDevice (Device Type, Device Index, TRUE);

if (hDevice == INVALID_HANDLE_VALUE)
{
        MessageBox (NULL,“Open Failed!“,“Error“,MB_OK);
}
```

DiiOpenNamedDevice

This function opens a device for further access.

Declaration

```
HANDLE DiiOpenNamedDevice (     LPCTSTR lpszDeviceName,
                                BOOL bExclusive
                           );
```

Parameters

lpszDeviceName The name of the device to open. For further information, please see the chapter „Device Naming“

bExclusive Determines, whether to open the device exclusively (TRUE), or allow other applications to use the device, as well.

Return value

A valid handle representing the device, or INVALID_HANDLE_VALUE (-1) if an error occurred.

Example

```
HANDLE hDevice = DiiOpenNamedDevice (“DeviceName”, TRUE);

if (hDevice == INVALID_HANDLE_VALUE)
{
        MessageBox (NULL,“Open Failed!“,“Error“,MB_OK);
}
```

Remarks

This function is the preferred way of opening a device. The function scans the configuration information provided, and returns a handle to the device, if found.

DiiCloseDevice

This function closes a device.

Declaration

```
BOOL DiiCloseDevice ( HANDLE hDevice );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

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.

Example

```
DiiCloseDevice (hDevice);
```

8.2. Functions to enumerate or browse devices

DiiSelectDevice

This function displays a dialog box on the screen and allows the user to select one of the installed devices. The name of the device is returned, and can then easily be passed to DiiOpenNamedDevice.

Declaration

```
BOOL DiiSelectDevice (      HWND        hParent,
                            LPTSTR      lpszDeviceName,
                            DWORD       dwMaxDeviceName
                      );
```

Parameters

hParent A valid handle representing the parent window for the dialog box to be displayed or NULL.

lpszDeviceName A pointer to a buffer receiving the name of the device selected.

dwMaxDeviceName The maximum size of the buffer pointed to by lpszDeviceName.

Return value

TRUE if successful and the user has confirmed his/her selection, FALSE otherwise.

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

ERROR_MORE_DATA - The buffer passed by lpszDeviceName was too small.

Example

```
char szDeviceName[101];

if (DiiSelectDevice (NULL, szDeviceName,100))
{
        HANDLE hDevice = DiiOpenNamedDevice (szDeviceName, TRUE);

        DiiCloseDevice (hDevice);
}
```

Remarks

This powerful function allows the programmer to keep his/her application independent of a specific data acquisition device, as it allows to dynamically select a device by the user. Of course, the devices list on the window are all installed on the computer before, so they doesn't represent the active devices on board.

DiiGetInstalledDevice

This function returns a name of a device installed. It can be called multiple times to enumerate all devices installed in a computer.

Declaration

```
BOOL DiiGetInstalledDevice (   DWORD      dwIndex,
                               LPTSTR     lpszDeviceName,
                               DWORD      dwMaxDeviceName
                           );
```

Parameters

dwIndex The sequence of device, whose name is to be retrieved. It is not the same to device's index.

lpszDeviceName A pointer to a buffer receiving the name of the device selected.

dwMaxDeviceName The maximum size of the buffer pointed to by lpszDeviceName.

Return value

TRUE if successful and the user has confirmed his/her selection, FALSE otherwise.

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

ERROR_MORE_DATA - The buffer passed by lpszDeviceName was too small.

ERROR_NO_MORE_ITEMS - There are no more devices to be returned

Example

```
//
// print all devices installed on screen:
//
char szDeviceName[201];

for (int counter = 0;;counter++)
{
        if (!DiiGetInstalledDevice (counter, szDeviceName, 200))
                break;

        printf ("%d . %s\n",counter,szDeviceName);
}
```

8.3. Functions to retrieve information about an Device

DiiGetNumberOfDigitalChannels

This function returns the number of digital channels a device offers (if any)

Declaration

```
DWORD DiiGetNumberOfDigitalChannels ( HANDLE hDevice );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

Return value

The number of digital channels the device supports

Example

```
HANDLE hDevice = DiiOpenNamedDevice (“DeviceName“,TRUE);
DWORD dwChannels = DiiGetNumberOfDigitalChannels( hDevice );
DiiCloseDevice (hDevice);
```

DiiGetNumberOfDigitalInOutChannels

This function returns the number of digital channels a device offers (if any), differentiated by input and output channels.

Declaration

```
BOOL DiiGetNumberOfDigitalInOutChannels (
                                    HANDLE hDevice,
                                    LPDWORD lpdwInputChannels,
                                    LPDWORD lpdwOutputChannels);
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

lpdwInputChannels
A pointer to a DWORD variable receiving the number of input channels for the device.

lpdwOutputChannels
A pointer to a DWORD variable receiving the number of output channels for the device.

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 line number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DWORD dwInputChannels;
DWORD dwOutputChannels;

DiiGetNumberOfDigitalChannels( hDevice, &dwInputChannels, &dwOutputChannels );

DiiCloseDevice (hDevice);
```

DiiGetNumberOfAnalogChannels

This function returns the number of analog channels a device offers (if any)

Declaration

DWORD DiiGetNumberOfAnalogChannels(HANDLE hDevice);

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

Return value

The number of analog channels the device supports

Example

```
HANDLE hDevice = DiiOpenNamedDevice (“DeviceName“,TRUE);
DWORD dwChannels = DiiGetNumberOfAnalogChannels( hDevice );
DiiCloseDevice (hDevice);
```

DiiGetNumberOfAnalogInOutChannels

This function returns the number of analog channels a device offers (if any), differentiated by input and output channels.

Declaration

```
BOOL DiiGetNumberOfAnalogInOutChannels (  HANDLE hDevice,
                                          LPDWORD lpdwInputChannels,
                                          LPDWORD lpdwOutputChannels);
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

lpdwInputChannels
A pointer to a DWORD variable receiving the number of input channels for the device.

lpdwOutputChannels
A pointer to a DWORD variable receiving the number of output channels for the device.

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 line number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DWORD dwInputChannels;
DWORD dwOutputChannels;

DiiGetNumberOfAnalogChannels( hDevice, &dwInputChannels, &dwOutputChannels );

DiiCloseDevice (hDevice);
```

DiiGetResolution

This function returns the analog resolution a device supports.

Declaration

```
DWORD DiiGetResolution( HANDLE hDevice );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

Return value

A number representing the number of bits available for analog input/output, for example 12 or 14.

If the device does not have analog input/output capabilities, zero is returned.

Example

```
HANDLE hDevice = DiiOpenNamedDevice (“DeviceName“,TRUE);
DWORD dwResolution = DiiGetResolution( hDevice );
DiiCloseDevice (hDevice);
```

8.4. Functions for digital input/output

DiiSetDigitalBit

This function sets or clears a single bit on a digital output line.

Declaration

```
BOOL DiiSetDigitalBit ( HANDLE hDevice,
                        DWORD dwLine,
                        BOOL    bState
                      );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwLine The index of the bit on the card to manipulate. The first bit has index 0.

bState The new state of the bit, either set (1/TRUE) or cleared (0/FALSE)

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 line number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiSetDigitalBit ( hDevice, 0, 1);          // set's first bit to one.
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

This function automatically configures the specified port for output, if the chip was not configured before.

Since the 8255 chip's lines can be reconfigured for input/output, the bit line index starts counting on the very first line of the first chip, regardless of whether it was configured as input or output at the time the function was called.

The 8255 chip's control port is disregarded for standard input/output. So line 24 will be the first line of Port A of the second 8255 chip on the device (if any)

AD/DA Devices (DIO parts):

AD/DA digital I/O is 2-byte, so please use function DiiSetDigitalWord.

PCI 4 Channel Industrial Card:

PCI IND digital I/O is 2-byte, so please use function DiiSetDigitalWord.

DiiSetDigitalByte

This function outputs a complete byte to a digital output port of a device.

Declaration

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

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwPort The index of the port on the card to manipulate. The first port has index 0.

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 = DiiOpenNamedDevice ("DeviceName",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiSetDigitalByte( hDevice, 0, 0xFF);          // set's all bits on the first port
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

This function automatically configures the specified port for output, if the chip was not configured before.

Since the 8255 chip's lines can be reconfigured for input/output, the port index starts counting on the very first line of the first chip, regardless of whether it was configured as input or output at the time the function was called.

The 8255 chip's control port is disregarded for standard input/output index calculation. So port 3 will be Port A of the second 8255 chip on the device (if any)

AD/DA Devices (DIO parts):

AD/DA digital I/O is 2-byte, so please use function DiiSetDigitalWord.

PCI 4 Channel Industrial Card:

PCI IND digital I/O is 2-byte, so please use function DiiSetDigitalWord.

DiiSetDigitalWord

This function outputs a complete word (2-byte) to a digital output port of a device.

Declaration

```
BOOL DiiSetDigitalWord ( HANDLE hDevice,
                         DWORD dwPort,
                         Word  wData
                        );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwPort The index of the port on the card to manipulate. The first port has index 0.

wData 2-byte data to write.

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 = DiiOpenNamedDevice ("DeviceName",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiSetDigitalWord( hDevice, 0, 0xFFFF);
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

8255 digital I/O is 1-byte, so please use function DiiSetDigitalBit and DiiSetDigitalByte.

AD/DA Devices (DIO parts):

This function only show digital I/O port for user, so index 0 represent real digital I/O port 6.

PCI 4 Channel Industrial Card:

index 0 represent channel 0, index 1 represent channel 1, and etc. You would not know the real port address mapping to each channel.

DiiGetDigitalBit

This function returns the state of a single bit on an input port of a device

Declaration

```
BOOL DiiGetDigitalBit (   HANDLE   hDevice,
                          DWORD    dwLine,
                          LPBOOL   lpbState
                      );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwLine The index of the bit on the card to manipulate. The first bit has index 0.

lpbState A pointer to a variable receiving the new state of the bit, either set (1/TRUE) or cleared (0/FALSE)

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 line number was out of range for the device selected.

Example

```
BOOL bState;

HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiGetDigitalBit ( hDevice, 0, &bState);        // reads the state of the first bit
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

This function automatically configures the specified port for input, if the chip was not configured before.

Since the 8255 chip's lines can be reconfigured for input/output, the bit line index starts counting on the very first line of the first chip, regardless of whether it was configured as input or output at the time the function was called.

The 8255 chip's control port is disregarded for standard input/output. So line 24 will be the first line of Port A of the second 8255 chip on the device (if any).

AD/DA Devices (DIO parts):

AD/DA digital I/O is 2-byte, so please use function DiiGetDigitalWord.

PCI 4 Channel Industrial Card:

PCI IND digital I/O is 2-byte, so please use function DiiGetDigitalWord.

DiiGetDigitalByte

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

Declaration

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

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwPort The index of the port on the card to read. The first port has index 0.

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

```
BYTE byState;

HANDLE hDevice = DiiOpenNamedDevice ("*",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiGetDigitalByte( hDevice, 0, &byState);          // reads the state of the first input port
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

This function automatically configures the specified port for input, if the chip was not configured before.

Since the 8255 chip's lines can be reconfigured for input/output, the port index starts counting on the very first line of the first chip, regardless of whether it was configured as input or output at the time the function was called.

The 8255 chip's control port is disregarded for standard input/output index calculation. So port 3 will be Port A of the second 8255 chip on the device (if any)

AD/DA Devices (DIO parts):

AD/DA digital I/O is 2-byte, so please use function DiiGetDigitalWord.

PCI 4 Channel Industrial Card:

PCI IND digital I/O is 2-byte, so please use function DiiGetDigitalWord.

DiiGetDigitalWord

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

Declaration

```
BOOL DiiGetDigitalWord  (   HANDLE    hDevice,
                            DWORD     dwPort,
                            LPWORD    wDataBuf
                        );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwPort The index of the port on the card to read. The first port has index 0.

wDataBuf A pointer to a variable of type Word receiving the data from channel.

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

```
Word wData;

HANDLE hDevice = DiiOpenNamedDevice ("*",TRUE);
if (hDevice != INVALID_HANDLE_VALUE)
{
        DiiGetDigitalWord( hDevice, 0, &wData);          // reads the state of the first input port
        DiiCloseDevice (hDevice);
}
```

Remarks

8255 Devices:

8255 digital I/O is 1-byte, so please use function DiiSetDigitalBit and DiiSetDigitalByte.

AD/DA Devices (DIO parts):

This function only show digital I/O port for user, so index 0 represent real digital I/O port 6.

PCI 4 Channel Industrial Card:

index 0 represent channel 0, index 1 represent channel 1, and etc. You would not know the real port address mapping to each channel.

DiiGetOutputPort

This function reads the state of an output port.

Declaration

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

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwPort The index of the *output* port on the card to read. The first port has index 0.

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 = DiiOpenNamedDevice ("DeviceName",TRUE);

DiiSetDigitalByte ( hDevice, 0, 0x12 );

BYTE byState;
DiiGetOutputPort( hDevice, 0, &byState);          // reads the state of the first output port

DiiCloseDevice (hDevice);
```

Remarks

On the past, because some hardware adapters, like photo relay series, don't offer the ability to read the state of an output port from the hardware, this function is designed for them. But from now on, all PCI devices could use normal digital I/O functions, like DiiGetDigitalByte, and this function has no necessary to use (its works the same as DiiGetDigitalByte). It is for capacibility now.

8.5. Functions to configure digital channels

DiiSet8255Config

This function configures a 8255 chip on the device.

Declaration

```
BOOL DiiSet8255Config ( HANDLE   hDevice,
                        DWORD    dwChip,
                        BYTE     byConfiguration
                      );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwChip The index of the chip on the card to manipulate. The first chip has index 0.

byConfiguration A byte containing the new configuration for the 8255 chip. This byte must conform to the configuration byte specified in the 8255 chip's data sheet.

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 chip number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DiiSet8255Config( hDevice, 0, 0x80);     // configures all ports as output, mode 0

DiiCloseDevice (hDevice);
```

Remarks

This function sets the device configuration for a 8255 chip on the card.

After specifically setting the device configuration, the DII library does no longer reconfigure the ports in response to DiiGetDigitalByte/DiiSetDigitalByte calls.

You may also directly use the chip's bit set/reset feature using this call, as it outputs directly to the control port of the specified 8255 chip.

8.6. Functions for analog input / output

DiiSetAnalogChannel

This function sets an analog channel on the device to a specific value. It takes a 32-bit integer value as a RAW value to be written to the channel.

Declaration

```
BOOL DiiSetAnalogChannel(   HANDLE hDevice,
                            DWORD dwChannel,
                            DWORD dwValue
                        );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwChannel The index of the channel on the card to manipulate. The first channel has index 0.

dwValue A 32-bit integer containing the value to output. The value is truncated according to the appropriate resolution of the device

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 channel number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DiiSetAnalogChannel( hDevice, 0,  0);    // sets the first analog channel to zero

DiiCloseDevice (hDevice);
```

DiiGetAnalogChannel

This function sets an analog channel on the device to a specific value. It returns the raw integral value from the card.

Declaration

```
BOOL DiiGetAnalogChannel(   HANDLE hDevice,
                            DWORD dwChannel,
                            LPDWORD lpdwValue
                         );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwChannel The index of the channel on the card to read. The first channel has index 0.

lpdwValue A pointer to a 32-bit integer receiving the current value of the analog channel. The value is truncated according to the appropriate resolution of the device

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 channel number was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("*",TRUE);

DWORD dwValue;
DiiGetAnalogChannel( hDevice, 0, &dwValue);   // reads the first analog channel on the device

DiiCloseDevice (hDevice);
```

Remarks

This function returns the raw analog channel value as given from the card. It does not perform any transformation into the value range setup by the user and supported by the device.

8.7. Functions to access timers on cards

DiiSetTimerConfig

This function configures timer chips on the device.

Declaration

```
BOOL DiiSetTimerConfig (    HANDLE hDevice,
                            DWORD dwTimer,
                            DWORD dwConfig
                        );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwTimer The index of the timer on the card to access. The first timer has index 0.

dwConfig A 32-bit integer containing the configuration for the timer. On 8253 timers, only the lowest byte is being used.

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 timer index was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DiiSetTimerConfig( hDevice, 0, 0x6);     // set square wave generation for timer

DiiCloseDevice (hDevice);
```

Remarks

8253 Chips:
This function sets the configuration byte for a timer on the 8253 chip. The configuration is not actually output, until DiiLoadTimer is called.

DiiLoadTimer

This function loads a value into a timer.

Declaration

```
BOOL DiiLoadTimer(      HANDLE hDevice,
                        DWORD dwTimer,
                        DWORD dwValue
                  );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwTimer The index of the timer on the card to access. The first timer has index 0.

dwValue A 32-bit integer containing the timer value to write.

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 timer index was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DiiLoadTimer( hDevice, 0, 40000);       // load the value into the first timer

DiiCloseDevice (hDevice);
```

Remarks

8253 Chips:
This function actually writes out the timer configuration and loads the timer according to the 8253 specifications.

DiiGetTimer

This function loads a value into a timer.

Declaration

```
BOOL DiiGetTimer(       HANDLE hDevice,
                        DWORD dwTimer,
                        LPDWORD lpdwValue
                );
```

Parameters

hDevice A valid device handle, previously obtained from DiiOpenDevice or DiiOpenNamedDevice

dwTimer The index of the timer on the card to access. The first timer has index 0.

lpdwValue A pointer to a 32-bit integer receiving the current timer value.

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 timer index was out of range for the device selected.

Example

```
HANDLE hDevice = DiiOpenNamedDevice ("DeviceName",TRUE);

DWORD dwValue;
DiiGetTimer( hDevice, 0, &dwValue);     // read the current value of the timer

DiiCloseDevice (hDevice);
```

Remarks

8253 Chips:
A counter latching operation is actually performed, so reading the timer does not interfere with the actual timing.

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 web-site.

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 ("Dii.h") as well as an import library ("Dii.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 "Dii.h" include file, then you can use any of the functions provided by the DII DLL. Be sure to include the import library "Dii.lib" during the linking step of your application, so your applications successfully references the actual interface DLL.

9.2. Visual Basic

Note: 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 Dynamic Industrial Interface (DII) , you can call the Dii 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 Dynamic Industrial Interface (DII).

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 DII.