Dynamic Industrial Interface
V 1.8

Die universelle Benutzerschnittstelle zur Meßdatenerfassung

Handbuch

Design & Entwicklung von
NT-ware Systemprogrammierungs GmbH

Teile dieser Dokumentation dürfen ohne Genehmigung der NT-ware System-Programmierungs GmbH weder vervielfältigt noch auf irgendeine andere Art, sei es auf elektronischem Wege oder durch Aufnahmen, übertragen werden.

Inhalt

1. Einleitung

Dieses Handbuch stellt Ihnen das Dynamic Industrial Interface vor, einschließlich aller Funktionsaufrufe, Installationsanforderungen und Handhabungen.

Garantie/Ausschluß der Garantie

Die NT-ware Systemprogrammierungs GmbH (NT-ware) haftet nicht für Schäden, die durch den Gebrauch der Software verursacht wurden. In keinem Fall ist die NT-ware für Schäden verantwortlich, die aus dem Gebrauch der Sache herrühren, wie z.B. entgangener Geschäftsgewinn, Geschäfts-unterbrechung oder dem Verlust von Geschäftsdaten. Ebenso kann nicht für Schäden gehaftet werden, die entstehen, weil das Produkt nicht sachgemäß angewendet wird. Dies gilt auch dann, wenn wir über die Möglichkeit des Schadens informiert worden sind.

Geschützte Markennamen

Visual Basic, Visual C++, OCX and ActiveX sind eingetragene Markennamen der Microsoft Corporation, Redmond.
Delphi ist ein eingetragener Markenname der Borland, Inc.

2. Beschreibung

Das Dynamic Industrial Interface (DII) ermöglicht Ihnen den standardisierten Zugriff auf alle Geräte zur Meßdatenerfassung. Mit dem DII verfügen Sie über eine Treiber-Schnittstelle, die nicht nur zu PC-Karten oder Pocket I/O Adaptern paßt. Sie ist auch für zukünftige Entwicklungen gerüstet, denn sie ist so ausgestattet, daß sie auch Geräte unterstützen wird, die jetzt noch in der Entwicklung sind.

Im besonderen zeichnet sich das DII durch folgende Eigenschaften aus:

Plattform-unabhängig
Die Schnittstelle ist binär kompatibel mit Windows NT and Windows 95. Dadurch können Programme, die auf einem dieser Betriebssysteme geschrieben wurden, auch unverändert auf dem anderen laufen, sogar ohne Rekompilieren.

- Abstrahiert die Funktionalität eines Gerätes
  Die Funktionalität der Karte wird durch die Schnittstelle sichergestellt. Für den Nutzer sind daher spezielle Kenntnisse über das jeweilige Karten-Design nicht notwendig. So muß er z. B. nicht darüber nachdenken, welcher Port für welche Funktion benötigt wird.
  Das bedeutet auch, daß die ISA 16 Relay-Karte auf die gleiche Art programmiert wird wie die ADDIO 16 Relay Output-Karte. Über die Details der ADDIO-Implementation, wie z. B. den parallelen Port-Zugriff, muß der Nutzer keine Kenntnisse haben.

- Geräte-Kennung
  Ein besonderes Leistungsmerkmal des DII ist die Geräte-Kennung. Sie ermöglicht es Ihnen, bei der Installation jedem Gerät einen eigenen Namen zu geben. Dieses erleichtert Ihnen den Zugang zu Geräten und Karten in den Fällen, wo mehrere Karten gleichen Typs installiert werden. Darüberhinaus vereinfacht es die Überragbarkeit von Anwendungen zwischen den Produkten. Beispiel: Der Nutzer möchte an einer Maschine das Pocket I/O Module einsetzen, das mit einem Sensor verbunden ist, an einer anderen Maschine aber mit einer ISA-Karte arbeiten. Indem er nun beiden Geräten den gleichen Namen gibt, kann er seine Anwendung von einer Maschine zur anderen übertragen, ohne irgendeinen Quellcode ändern zu müssen,.

- Standardisierte Benachrichtigung
  Mit der DII verfügen Sie über eine Möglichkeit, Benachrichtigungen und Interrupts, die von Industrial I/O-Karten gesendet werden, bequem und auf standardisierte Weise handhaben zu können. Oftmals gestaltet es sich für die Nutzer schwer, mit Interrupts im Rahmen von Windows 95/NT umzugehen, erfordern Interrupts doch, daß ein Kernel-Mode-Geräte-Treiber geschrieben werden muß. Die Verwendung von Interrupt-basierten Karten wird die Leistung und den Nutzen von Industrial I/O – Karten unter Windows 95/NT nun dramatisch erhöhen. Das Polling dagegen verlangsamt das gesamte Betriebssystem und ist in vielen Situationen nicht praktikabel.
  Mit dem DII kann der Nutzer die Vorzüge von Interrupt-getriebenen I/O-Karten voll ausnutzen, denn er erhält jedesmal eine Nachricht, wenn die Industrial I/0-Karte einen Interrupt sendet.
  Die Funktion der Benachrichtigung vom DII wird durch OCX vergrößert, weil die Nachricht als OCX-Ereignis empfangen wird und auf diese Weise bei Anwendungen wie Visual Basic leicht gehandhabt werden kann.

- Unabhängig von Programmier-Sprachen
  Die DII ermöglicht einen sprachunabhängigen Zugriff auf Industrial I/O card, weil sie über eine Dynamic-Link-Library Architektur verfügt.

- Volle 32-bit-Unterstützung
  Die DII unterstützt die 32-bit-Anwendungen von Windows NT/95. Das bedeutet, daß Visual Basic 4.0/5.0, Delphi, Visual C++, etc. als Zugang zu I/O Adaptersbenutzt werden können. Mit dem Einsatz von OCX/ActiveX control, können sogar JAVA-Anwendungen Industrial I/O-fähig werden.

- Volle Multithreading- Unterstützung
  Das DII unterstützt Multithreading, so daß die gleiche Anwendung gleichzeitig den Zugang zur Hardware eröffnet. Die 0CX-Komponente unterstützt das ”apartment threading model”.

- Übertragbarkeit und Stabilität
  Der Nutzer kann seine Anwendung leicht von seinem Desk-Top-Computer auf ein Laptop übertragen, ohne die Anwendung wechseln zu müssen und auch ohne Rekompilierung. Der Desk-Top-Computer muß dazu über eine ISA ADDA-Karte, das Laptop über eine Pocket A/D-Box verfügen.

3. Der Inhalt des Dynamic Industrial Interface

Das Dynamic Industrial Interface Library Distribution besteht aus den folgenden Komponenten:

1. Windows NT Kernel -Treibern

2. Windows 95 VxD - Treibern

3. Eine Systemsteuerung, um die Geräte-Konfiguration zu bearbeiten und anzuzeigen.

4. Ein Win32 DLL als Zugang zum Treiber. Dieser Treiber ermöglicht sowohl den Zugang zu Windows 95 VxD als auch zu dem Windows NT-Treiber.
   Um auf DLL zugreifen zu können, sind eine C-Header-Datei und eine Visual Basic-Datei beigefügt, die eine Beschreibung der Funktionen enthalten.

5. Eine OCX/ActiveX - Komponente

6. Eine Visual C++ Muster-Anwendung.
   Die Muster-Anwendung zeigt den Gebrauch und Nutzen des API-Treibers. Er stellt den Status der Input und Output Ports graphisch da und ermöglicht es mit der Leichtigkeit eines Mausklicks, die Output-Ports anzusteuern oder zu verlassen.
   Darüberhinaus werden noch der Status von analogen Kanälen für AD/DA-Geräte und einige Tests für 8253-Timerchips angezeigt.

Die Anwendung läuft unverändert sowohl unter Windows 95 als auch unter Windows NT.

Bei der Entwicklung wurde Visual C++ 5.0. verwendet. Der gesamte Quellcode ist beigefügft.

Nach der Installation können sie die Visual C++ Demo im Unterverzeichnis “DiiDemo” unter Ihrem Installations-Verzeichnis finden.

7. Eine Visual Basic Beispiel-Anwendung
   Das Visual Basic-Beispiel gleicht dem Visual C++ Beispiel. Es wurde allerdings mit der Visual Basic Version 5.0 geschrieben und verwendet dieOCX/ActiveX – Komponente, die mit dem DII mitgeliefert wird.

   Nach der Installation finden Sie die Visual Basic Demo im Unterverzeichnis “DiiDemoVB” unter Ihrem Installations-Verzeichnis.

8. Eine Delphi-Beispiel-Anwendung
   Die Delphi-Beispiel gleicht dem Visual C++ Beispiel. Es wurde allerdings mit der Delphi Version 2.0 geschrieben und verwendet die OCX/ActiveX – Komponente, die mit dem DII mitgliefert wird.

   Nach der Installation finden Sie das Delphi-Beispiel im Unterverzeichnis “DiiDemoDelphi” unter Ihrem Installations-Verzeichnis.

4. Installation

Die Installation von Windows NT – Treiber-Dateien und Beispiel-Anwendungen:

1. Legen Sie die Treiber-Diskette in das Disketten-Laufwerk ein und starten Sie das Programm “Setup.Exe“ in Ihrem Haufptverzeichnis.
   Diese Anwendungen wird Sie durch den Installations-Prozeß leiten.

2. Wenn Sie die Installation beendet haben, können Sie Geräte mit Hilfe der Systemsteuerung hinzufügen. Für Windows NT 4.0 wählen Sie bitte ”Start/Einstellungen/Systemsteuerung” und dann das Symbol für das Dynamic Industrial Interface. Für Windows NT 3.51, starten Sie bitte die Systemsteuerung aus dem Program Manager.

3. Mit der der System-Steuerung können Sie nun Geräte hinzufügen. Dazu müssen Sie zunächst die passenden Produkt-Typen (PC –Karten oder Pocket I/O - Karten) zufügen.

4. Klicken Sie auf die Zeile ”Hinzufügen”, um ein neues Gerät in die Liste der bereits installierten Geräte aufzunehmen. Eine Dialog-Box mit der Liste der unterstützten Geräte wird angezeigt. Bitte wählen Sie nun das Gerät aus, das Sie installieren wollen oder eines, das mit ihm kompatibel ist.

5. Je nachdem welchen Geräte-Typ Sie ausgewählt haben, müssen Sie nun die Port-Base-Address und den Gerätenamen bestimmen. Die ausführliche Beschreibung finden Sie im Kapitel ”Geräte-Kennung”.

6. Bitte wiederholen Sie die Schritte 3. – 5. für alle Geräte, die Sie installieren möchten. Die ausführliche Beschreibung für den Installationsprozeß finden Sie in den Kapiteln 4.1 und 4.2..

7. Beenden Sie die Installation, indem Sie ”Ok” in der Systemsteuerung drücken.

Die Installation von Windows 95 – Treiber-Dateien und Beispiel-Anwendungen

1. Legen Sie Ihre Treiber-Diskette in das Laufwerk ein und starten Sie das Programm “Setup.Exe“ im Hauptverzeichnis.
   Dieses Anwendung wird Sie durch den Installationsprozeß leiten.

2. Nutzen Sie den Windows 95 Hardware – Assistenten, den Sie in der Systemsteuerung finden, um neue DII-unterstützte Geräte hinzuzufügen. Wählen Sie “Industrial I/O Devices” um die Geräteklasse hinzuzufügen. Der DII Device Selector wird erscheinen.

3. Wählen Sie das Gerät aus, das Sie dem System hinzufügen möchten. Windows 95 wird Ihrem Gerät nun den passenden I/O – Port und die IRQ`s zuweisen. Wenn Sie diese Einstellungen ändern möchten, gehen Sie bitte in den Geräte-Manager, den Sie in der Systemsteuerung finden.

4. Wenn Sie diese Einstellungen bestätigt haben, müssen Sie einen Namen für das Gerät eingeben. Mehr Informationen zur Geräte-Benennung im Kapitel ”Geräte-Kennung”.

5. Wiederholen Sie die Schritte 2.-5. Für alle Geräte, die Sie installieren möchten. Die ausführliche Beschreibung für den Installationsprozeß finden Sie in den Kapiteln 4.1 und 4.2.

4.1. Installation / Entfernen von ISA I/O Cards

Für ISA-Karten ist der Installationsprozeß nur geringfügig anders als bei Windows NT and Windows 95. Die DII-Geräte-Treiber sind nämlich voll in die Plug And Play-Architektur von Windows 95 integriert, die zur Zeit natürlich nicht für Windows NT erhältlich ist.

Installation unter Windows NT:

1. Für Windows NT (3.51 and 4.0), starten Sie bitte das “Industrial I/O Devices” Setup in Ihrer Systemsteuerung.
2. Klicken Sie auf die Zeile ”Hinzufügen”, um ein neues Gerät in die Liste der bereits installierten Geräte aufzunehmen. Eine Dialog-Box mit der Liste der unterstützten Geräte wird angezeigt. Bitte wählen Sie nun das Gerät aus, das Sie installieren wollen oder eines, das mit ihm kompatibel ist.
3. Je nachdem,welchen Geräte-Typ Sie ausgewählt haben, müssen Sie nun die Port-Base-Adress und den Gerätenamen bestimmen. Die ausführliche Beschreibung finden Sie im Kapitel ”Geräte-Kennung”.
   Beachten Sie, daß die Installations-Anwendung noch keine Port-Konflikte mit der I/O-Adress anzeigt, die Sie gewählt haben.
4. Wiederholen Sie diese Schritte für jedes Gerät, das Sie diesem System hinzufügen möchten.

Entfernen unter Windows NT:

1. Für Windows NT (3.51 and 4.0), starten Sie bitte das “Industrial I/O Devices” Setup in Ihrer Systemsteuerung.
2. Wählen Sie aus dem “ISA/PCI I/O Cards” – Fenster das Gerät, das Sie entfernen wollen.
3. Klicken Sie auf das Feld ”Entfernen”, um das Gerät aus Ihrem System zu beseitigen.
4. Wiederholen Sie diese Schritte für jedes Gerät, das sie aus Ihrem System entfernen möchten.

Installation under Windows 95

1. Aktivieren Sie den Hardware-Assistenten der Systemsteuerung von Windows 95.
2. Wählen Sie ”Manuelle Geräte hinzufügen”. Starten Sie bitte nicht die automatische Hardware-Erkennung.
3. Wählen Sie die Geräte-Kategorie ”IndustrialI/O Devices”.
4. Wählen Sie aus dem “device selection dialog”, der auf dem Monitor erscheint die I/O-Karte, die Sie möchten und klicken Sie auf ”Ok”.
5. Windows weist automatisch I/O Base address zu und gegebenenfalls auch eine IRQ, die für Ihre Karte verfügbar ist.
   Bitte versichern Sie sich, daß die Einstellungen auf Ihrer Karte zu denen von Windows 95 passen. Später können Sie die von Windows 95 zugewiesenen Einstellungen in der Systemsteuerung ändern.
6. Nachdem Sie Ihre I/O-Port-Einstellungen kontrolliert haben, werden Sie aufgefordert, einen Namen für das neue Gerät einzugeben oder den vorgegebenen Namen zu akzeptieren.
7 Wiederholen Sie diese Schritte für jedes Gerät, das Sie Ihrem System zufügen wollen.
8. Es ist erforderlich, daß Sie Ihren PC neu starten, wenn Sie Ihre Karten installiert haben.

Entfernen unter Windows 95

1. Starten Sie das System-Applet aus der Systemsteuerung von Windows 95 und wechseln Sie zu dem Fenster ”Geräte-Manager”.
2. Hier finden Sie alle Geräte, die Sie der Kategorie ”Industrial I/O Geräte” hinzugefügt haben. Wählen Sie die Geräte aus, die Sie entfernen wollen und klicken Sie auf das Feld ”entfernen”.
3. Wiederholen Sie diese Schritte für jedes Gerät, das Sie aus Ihrem System entfernen wollen.
4. Es ist erforderlich, daß Sie Ihren PC neu starten, wenn Sie Ihre Karten entfernt haben.

4.2. Installation von Pocket I/O Series Modulen

Die Installation von Pocket I/O Modulen in Ihrem System ist für Windows 95 und Windows NT gleich.

Das Einrichten von Pocket I/O Modulen

1. Für Windows NT and Windows 95 starten Sie bitte das “Industrial I/O Devices” Setup in Ihrer Systemsteuerung.
2. Wechseln Sie zu der Seite “Pocket I/O Module”.
3. Fügen Sie nun als erstes einen Port hinzu, mit dem ein oder mehrere Pocket I/O Module verbunden sind. Klicken Sie dann auf ”Port hinzufügen”.
4. Bestimmen Sie, welchen parallelen Port Sie für die Verbindung zu Pocket I/O wählen oder welchen seriellen Port Sie für die Kommunikation mit dem Pocket COM Modul gebrauchen wollen. .Bei dem Com-Modul versichern Sie sich, daß die Baud-Rate und die BOX-ID-Einstellungen mit den DIP Switch-Einstellungen des COM Modules korrespondieren.
5. Wenn Sie einen Port hinzugefügt haben, erscheint er im Konfigurationsfenster. Um ein Pocket I/O Modul dem Port hinzuzufügen, wählen Sie den Port bitte mit Ihrer Maus aus und klicken dann auf das Feld ”Hinzufügen”.
6. Bestimmen Sie den Typ des Pocket I/O Moduls, das Sie Ihrem System hinzufügen möchten.
7. Geben Sie dem Pocket I/O Modul einen Namen, mit dem Sie später auf das Modul zugreifen können, oder akzeptieren Sie den vorgegebenen Namen.
8. Passen Sie die Box-ID den DIP-Switch-Einstellungen Ihres Pocket I/O Moduls an. Für weitere Informationen schlagen Sie bitte in Ihrem Pocket ADDIO-Handbuch nach.
9. Geben Sie die ungefähre Länge Ihres Kabels zwischen dem parallelen Port und dem Pocket I/O-Modul ein. Die DII Kernel Treiber benötigen seine Länge, um die Verzögerung für den Zugriff auf die Geräte berechnen zu können. Je länger das Kabel, desto länger die Verzögerung.
10. Wiederholen Sie diese Schritte für alle Pocket I/O-Module, die Sie Ihrem System hinzufügen möchten. Wenn Sie multiple Pocket I/O-Module mit demselben Port verbinden, achten sie darauf, daß die Box-ID`s für jede Box verschieden sind. Sonst werden Sie Schwierigkeiten bekommen, wenn Sie mit allen Boxen gleichzeit kommunizieren wollen.

Bitte denken Sie daran, daß Sie beliebig viele Ports mit beliebig vielen Pocket I/O-Modulen konfigurieren können, die mit jedem Port verknüpft sind. Die Begrenzung ergibt sich lediglich aus der Anzahl der auf Ihrem Computer installierten Ports und der Zahl der Pocket I/O-Module, die mit einem Port zu verbinden sind.

Entfernen von Pocket I/O Modulen

1. Für Windows NT und Windows 95 starten Sie bitte das ”Industrial I/O Devices” Setup in Ihrer Systemsteuerung.
2. Wechseln Sie zu der Seite “Parallel Port Pocket I/O Module”.
3. Wenn Sie ein einziges Gerät entfernen wollen, wählen Sie das Gerät mit Ihrer Maus aus und klicken dann auf ”Entfernen”.
4. Wenn Sie einen ganzen Port mit all jenen Geräten löschen möchten, die mit ihm verbunden sind, dann wählen Sie den Port bitte mit Ihrer Maus aus und klicken auf ”Port entfernen”.
5. Wiederholen Sie diese Schritte für alle Pocket I/O-Module, die Sie aus Ihrem System entfernen möchten.

5. De-Installation

Falls Sie beispielsweise einmal zu einer anderen Maschine wechseln wollen, kann das Dynamic Industrial Interface inclusive aller Dateien, Register-Einstellungen, Konfigurations-Dateien etc. automatisch und leicht entfernt werden.

Bei Windows 95 and Windows NT 4.0, nutzen Sie bitte “Add/Remove Software“ der Systemsteuerung und wählen dann “Dynamic Industrial Interface“. Klicken Sie dann auf “Entfernen “ und das Dynamic Industrial Interface ist von Ihrer Maschine gelöscht worden.

Bei Windows NT 3.51 wählen Sie bitte ”Dynamic Industrial Interface entfernen” aus der Programm-Managergruppe, die während des Installationsprozesses geschaffen wurde.

Bitte denken Sie daran, daß alle Einstellungen gelöscht werden. Wenn Sie die Schnittstelle wieder einsetzen wollen, dann müssen Sie das gesamte Programm, wie oben beschrieben, neu installieren. Ebenso muß jedes Gerät neu installiert werden. Gehen Sie dabei vor, wie oben beschrieben.

6. Die Geräte-Kennung

Das Dynamic Industrial Interface zeichnet sich durch eine besondere Eigenschaft aus: der Namens-Kennung. Bei der weiteren Arbeit mit dem DII werden sie die Leistungsfähigkeit uind den Nutzen dieser Neuerung besonders schätzen lernen.

Während der Installation der Geräte wird Ihnen aufgefallen sein, daß Sie jedem Gerät einen speziellen Namen zuweisen können. Wenn Sie Ihr Anwendungs-Programm erstellen, können Sie auf dieses Gerät mit dem Namen zugreifen, den Sie ihm gegeben haben. Sie können aber auch den Nutzer nach dem Namen fragen oder Sie lassen den Nutzer graphisch nach dem korrekten Gerät Ihrer Anwendung suchen.

Die Geräte-Kennung bringt Ihnen folgende Vorteile:

- Einfache Handhabung beim Benutzen mehrerer Gerätere
  Durch die Geräte-Kennung sind Sie jetzt in der Lage, leichter und schneller auf verschiedene Geräte des gleichen Typs zugreifen zu können. Wenn Sie z. B. 3 8 Relay 8 Photo Isolator–Karten auf Ihrer Maschine haben, müssen Sie diese Karten nicht länger über Typ und Index suchen, was ja oft zur Verwirrung führ. Jetzt sind Sie in der Lage, schon bei der Installation jeder Karte einen Namen zuzuweisen, insbesondere einen bedeutungsvollen Namen.

- Bessere Übertragbarkeit
  Sie werden bemerkt haben, daß es eines der Ziele des Dynamic Industrial Interface ist, den Zugriff auf die Geräte bezüglich aller Produkte zu erleichtern und zu vereinheitlichen. Der Anwender soll entlastet werden. Deshalb ist es auch nicht mehr nötig, daß er sich über den Unterschied zwischen einem Pocket I/O AD – Modul und einer ISA – Karte mit AD-Funktionalität informieren muß. Darüberhinaus ist Ihre Software auf verschiedene Produkte übertragbar, so daß Sie oder Ihr Endverbraucher das passende Gerät wählen können.
  Durch die Geräte-Kennung z. B. können Sie Ihrem Produkt ganz einfach einen Namen wie “AnalogIO“ zuweisen, indem Sie Analog Input and Output handhaben. Auf einer anderen Maschine können Sie den gleichen Namen einer anderen Karte oder einem anderen installierten Produkt zuweisen. Da ja nun Ihre Software mit einem Gerät namens “AnalogIO“ arbeitet und Sie sich um weitere Details Ihres Gerätes nicht kümmern müssen, kann Ihre Software ohne weitere Änderungen operieren, selbst auf verschiedenen Plattformen(Windows NT/ Windows 95).

7. Rückwärtskompatibilität

Um dem Nutzer die Übertragung des vorherigen RelDrv Relay/Photo–Treibers auf Windows NT and Windows 95zu erleichtern, wurde das the Dynamic Industrial Interface mit diesem Treiber kompatibel gemacht.

Durch Namenänderung der folgenden Funktionsaufrufe können Sie Ihre Sofware mit dem Dynamic Industrial Interface kompatibel machen:

RdOpenAdapter	->	DiiOpenDevice (better DiiOpenNamedDevice)
RdCloseAdapter	->	DiiCloseDevice
RdSetSingleRelay	->	DiiSetDigitalBit
RdSetRelaysByteWise	->	DiiSetDigitalByte
RdReadSingleIsolator	->	DiiGetDigitalBit
RdReadIsolatorsByteWise	->	DiiGetDigitalByte
RdReadAnalogChannel	->	DiiGetAnalogChannel
RdWriteAnalogChannel	->	DiiSetAnalogChannel

Weitere Unterschiede:

Unter Windows 95 unterstützt das Dynamic Industrial Interface nicht länger den Zugrifft auf Geräte ohne installiertem VxD–Treiber.

Um die Spreicherbenutzung reduzieren zu können, ist Windows 95 VxD jetzt vollkommen dynamisch. Es kann nach Bedarf geladen werden und geschlossen, wenn keine weiteren Anwenungen das DII gebrauchen.

Bitte beachten Sie: Um eine hochentwickelte Funktion wie die Geräte-Kennung voll auszunutzen, müssen Sie Ihren Aufruf “RdOpenAdater“ durch “DiiOpenNamedDevice” ersetzen.

Für weitere Fragen stehen wir Ihnen gerne zur Verfügung.

8. Die Funktions-Aufrufe des Dynamic Industrial Interface

Da das DII in der C++ Programmier-Sprache entwickelte wurde, kann es sein, daß einige der verwendeten Daten-Typen in der Programmier-Sprache fehlen, die Sie verwenden möchten.
Falls Sie in Ihrer Programmier-Sprache Schwierigkeiten mit DLL haben, können Sie auch die OCX-Komponente benutzen, die einen leichteren Zugriff auf viele Hochsprachen erlaubt.

Bitte orientieren Sie sich an der folgenden Tabelle:

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
LPCTSTR	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

Bitte beachten Sie: DLL arbeitet mit dem Standard Call (Pascal) Aufruf-Mechanismus. Dieser wird ebenso für alle DII-Systeme verwendet und ist mit Visual Basic kompatibel.

8.1. Funktionen zum Öffnen und Schließen von Geräten

DiiOpenDevice

Diese Funktion öffenet ein Gerät für den weiteren Zugriff.

Deklaration

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

Parameter

dwDeviceType Der Gerätetyp, der geöffnet werden soll. Näheres siehe unter ”Bemerkungen”.

dwIndex Der Index des Gerätes, das geöffnet werden soll. (0 ist das erste Gerät)

bExclusive Bestimmt, ob das Gerät allein geöffnet werden soll (TRUE), oder ob andere Anwendungen das Gerät auch benutzen dürfen.

Rückgabewert

Ein korrekter Handle oder INVALID_HANDLE_VALUE (-1) falls ein Fehler unterlaufen ist.

Bemerkungen

Diese Funkion ermöglicht Rückwärtskompatibilität. Bitte benutzen Sie für neue Anwendungen die Funktion DiiOpenNamedDevice

Der Parameter *dwDeviceType* kann einen der folgenden Werte enthalten:

RD_SMARTLAB_16CHANNEL	*(1)*
RD_SMARTLAB_8CHANNEL	*(2)*
RD_ICC_8RELAY8ISOLATOR	*(3)*
RD_ICC_16RELAY	*(4)*
RD_ICC_16ISOLATOR	*(5)*
RD_ICC_8SSR8LOGIC	*(6)*
RD_TTL	*(7)*
RD_IBC_32ISOLATOR	*(9)*
RD_ADVANCE_ADDA	*(0x100)*
RD_SUPER_12BIT_ADDA	*(0x101)*
RD_12BIT_ADDA	*(0x102)*
RD_8CHANNEL_DA	*(0x103)*
RD_SUPER_14BIT_ADDA	*(0x104)*

Beachten Sie: Die Windows 95 and Windows NT- Gerätetreiber sind dynamisch, d. h. sie werden nicht zur Startzeit in das System geladen. Wenn Sie einen Adapter das erste Mal öffenen möchten, kann das die Treiber aufladen und Photo-Isolator-Adapter können initialisiert werden.

DiiOpenNamedDevice

Diese Funktion öffnet ein Gerät für den weiteren Zugriff.

Deklaration

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

Parameter

LpszDeviceName Der Name des Gerätes,das geöffnet werden soll. Näheres siehe unter ”Geräte-Kennung

bExclusive Bestimmt, ob das Gerät allein geöffnet werden soll (TRUE), oder ob andere Anwendungen das Gerät auch benutzen dürfen.

Rückgabewert

Ein korrekter Handle, der das Gerät repräsentiert oder INVALID_HANDLE_VALUE (-1) falls ein Fehler unterlaufen ist.

Beispiel

```
HANDLE hDevice = DiiOpenNamedDevice (“Device*“, TRUE);

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

Bemerkungen

Diese Funktion ist der einfachste Weg, um ein Gerät für den späteren Gebrauch zu öffnen. Diese Funktion scant die Konfiguration, welche die Information liefert: Sie gibt den Handle zu dem Gerät zurück, falls er gefunden wurde.

Der *lpszDeviceName* Parameter kann auch ein Wildcard-Zeichen (‘*’) enthalten. Der gelieferte Name wird dann mit all den Gerätenamen verglichen, die gefunden wurden. Das erste passende Gerät wird zurückgeben.
Beispie.: DiiOpenNamedDevice (“Device*“, TRUE); öffnet ein Gerät namens “Device0“ or “Device12“, etc.

Note: Die Windows 95 and Windows NT-Geräte-Treiber sind dynamisch, d. h. sie werden nicht zur Startzeit in das System geladen. Wenn Sie einen Adapter das erste Mal öffnen möchten, kann das die Teiber aufladen und Photo-Isolator-Adapters können initialisiert werden.

DiiCloseDevice

Diese Funktion schließt ein Gerät wieder.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher von DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, ansonsten FALSE.

Im Falle eines Fehler, bringt GetLastError() die folgenden Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war nicht korrekt.

Beispiel

```
DiiCloseDevice (hDevice);
```

Bemerkungen

8.2. Funktionen zum Aufzählen und Suchen von Geräten

DiiSelectDevice

Diese Funktion zeigt eine Dialog-Box auf dem Bildschirm und gestattet es dem Nutzer, eines der installierten Geräte auszuwählen. Der Name des Gerätes wird zurückgegeben und kann dann leicht an DiiOpenNamedDevice übergeben werden.

Deklaration

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

Parameter

hParent Einn korrekter Handle, der das Parent-Window for the Dialog- Box anzeigt, die NULL anzeigen soll.

lpszDeviceName Ein Zeiger auf einem Puffer, der den Namen des gewählten Gerätes erhält.

dwMaxDeviceName Die maximale Größe des Puffers, auf den mit lpszDeviceName gezeigt wird.

Rückgabewert

TRUE falls erfolgreich und der Nutzer seine Wahl bestätigt hat, andernfalls FALSE

Im Falle eines Fehlers, bringt GetLastError() die folgenden Werte zurück:

ERROR_MORE_DATA – Der Puffer, der von lpszDeviceName übergeben wurde, war zu klein.

Beispiel

```
char szDeviceName[201];

if (DiiSelectDevice (NULL, szDeviceName,200))
{
        HANDLE hDevice = DiiOpenNamedDevice (szDeviceName, TRUE);

        DiiCloseDevice (hDevice);
}
```

Bemerkungen

Diese wirkungsvolle Funktion ermöglicht es dem Nutzer, seine Anwendung unabhängig von einem speziellen Datenmeßgerät zu betreiben.Ebenso gestattet es dem Nutzer, ein Gerät dynamisch auszuwählen.

DiiGetInstalledDevice

Diese Funktion gibt den Namen eines installierten Gerätes zurück. Es kann immer aufgerufen werden, um alle installierten Geräte aufzuzählen.

Deklaration

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

Parameter

dwIndex Der Index des Gerätes, dessen Namen gelesen werden soll.

lpszDeviceName Ein Zeiger auf einen Pufer, der den Namen des gewählten Gerätes zugewiesen bekommt.

dwMaxDeviceName Die maximale Größe des Puffers, auf den mit lpszDeviceNamegezeigt wird.

Rückgabewert

TRUE falls erfolgreich und der Nutzer seine Auswahl bestätigt hat, ansonsten FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgendes Werte zurück:

ERROR_MORE_DATA – Der Puffer, der durch lpszDeviceName übergeben wurde, war zu klein.

ERROR_NO_MORE_ITEMS – Es können keine weiteren Geräte zurückgeliefert werden.

Beispiel

```
        //
        // gebe alle installierten Geräte auf dem Bildschirm an:
        //
char szDeviceName[201];

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

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

Beispiele

8.3. Funktionen, die Informationen über ein Gerät zurückliefern

DiiGetNumberOfDigitalChannels

Diese Funktion gibt die Anzahl der digitalen Kanäle zurück, die ein Gerät anbietet (falls es welche anbietet).

Deklaration

DWORD DiiGetNumberOfDigitalChannels (HANDLE hDevice);

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

Die Zahl der digitalen Kanäle, die das Gerät unterstützt.

Beispiel

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

DWORD dwChannels = DiiGetNumberOfDigitalChannels( hDevice );

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetNumberOfDigitalInOutChannels

Diese Funktion liefert die Anzahl der digitalen Kanäle, die ein Gerät anbietet (falls es welche anbietet). Dabei wird zwischen Input- und Output-Kanälen unterschieden.

Deklaration

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

Parameter

hDevice Ein korekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

lpdwInputChannels
Ein Zeiger auf eine DWORD-Variable, die die Anzahl der Input-Kanäle für das Gerät erhält.

lpdwOutputChannels
Ein Zeiger auf eine DWORD-Variable, die die Anzahl der Output-Kanäle für das Gerät erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war unkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DWORD dwInputChannels;
DWORD dwOutputChannels;

DiiGetNumberOfDigitalChannels( hDevice, &dwInputChannels, &dwOutputChannels );

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetNumberOfAnalogChannels

Diese Funktion liefert die Anzahl der analogen Kanäle, die ein Gerät anbietet (falls es welche anbietet).

Deklaration

```
DWORD DiiGetNumberOfAnalogChannels( HANDLE hDevice );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice or DiiOpenNamedDevice erhalten wurde.

Rückgabewert

Die Anzahl der analogen Kanäle, die ein Gerät unterstützen.

Beispiel

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

DWORD dwChannels = DiiGetNumberOfAnalogChannels( hDevice );

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetNumberOfAnalogInOutChannels

Diese Funktion gibt die Zahl der analogen Kanäle zurück, die ein Gerät anbietet (falls es welche anbietet). Dabei wird zwischen Input- und Output-Kanälen unterschieden.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher von DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

lpdwInputChannels
Ein Zeiger auf eine DWORD-Variable, die die Nummer des Input-Kanals des Gerätes erhält.

lpdwOutputChannels
Ein Zeiger auf eine DWORD-Variable, die die Numme des Output-Kanals des Gerätes erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war unkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DWORD dwInputChannels;
DWORD dwOutputChannels;

DiiGetNumberOfAnalogChannels( hDevice, &dwInputChannels, &dwOutputChannels );

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetResolution

Diese Funktion liefert die analoge Auflösung, die ein Gerät unterstützt.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher von DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

Eine Zahl, die die Anzahl der Bits repräsentiert, die für dien analogen Input/Output erhältlich sind, z. B. 12 oder 14.

Falls das Gerät nicht über einen analogen Input/Output verfügt, wird NULL zurückgegeben.

Beispiel

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

DWORD dwResolution = DiiGetResolution( hDevice );

DiiCloseDevice (hDevice);
```

Bemerkungen

8.4. Funktionen für den digitalen Input/Output

DiiSetDigitalBit

Diese Funkion setzt einen einzelnen Bit auf einen digitalen Output-Kanal oder entfernt ihn davon.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, er vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwLine Der Index des Bits auf der Karte, das Sie ändern können. Das erste Bit hat den Index 0.

bState Der neue Status des Bits, entweder gesetzt (1/TRUE) oder entfernt (0/FALSE).

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war unkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSetDigitalBit ( hDevice, 0, 1);          // setzt den ersten Bit auf eins.

DiiCloseDevice (hDevice);
```

Bemerkungen

8255 Devices:

Wenn der Chip zuvor noch nicht konfiguriert war, konfiguriert diese Funktion diesen Port automatisch für den Output. Falls auf einem 8255-Chip dieser Kanal schon existierte, wird die Set/Reset-Funktionalität des Chips übernommen.
Weil die Kanäle des 8255-Chips für den Input/Output rekonfiguriert werden können, zählt der Bit-Line-Index mit dem ersten Kanal des ersten Chips aufwärts. Dabei spielt es keine Rolle, ob er zum Zeitpunkt des Funktionsaufrufes als Input oder Output konfiguriert war.
Der Kontroll-Port des 8255-Chips ist ungeeignet für standardisierten Input/Output. So wird Kanal 24 der erste Kanal für Port A auf dem zweiten 8255-Chip des Gerätes sein (falls es ihn gibt).

DiiSetDigitalByte

Diese Funktion liefert ein vollständiges Byte als Output zu einem digitalen Output-Port des Gerätes.

Deklaration

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

Parameter

hDevice Ein korrketer Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwPort Der Index des Ports auf der Karte, den Sie ändern können. Der erste Port hat den Index 0.

byPortState Der neue Statuts des Ports

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgenden Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Ports war außerhalb des Bereiches des ausgewählten Gerätes.

Beispiel

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

DiiSetDigitalByte( hDevice, 0, 0xFF);          // setzt alle Bits auf den ersten Port

DiiCloseDevice (hDevice);
```

Bemerkungen

8255 Devices:

Wenn der Chip zuvor noch nicht konfiguriert war, konfiguriert diese Funktion diesen Port automatisch für den Output.
Weil die Kanäle des 8255-Chips für den Input/Output rekonfiguriert werden können, zählt der Port-Index mit dem ersten Kanal des ersten Chips aufwärts. Dabei spielt es keine Rolle, ob er zum Zeitpunkt des Funktionsaufrufes als Input oder Output konfiguriert war.
Der Kontroll-Port des 8255-Chips ist für die standardisierte Input/Output-Index-Kalkulation nicht geeignet. So wird Port 3 gleich Port A auf dem zweiten 8255-Chip des Gerätes sein (falls es ihn gibt).

DiiGetDigitalBit

Diese Funktion gibt den Status eines einzelnen Bits an einen Input-Port des Gerätes zurück

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwLine Der Index des Bits auf der Karte, das Sie ändern möchten. Das erste Bit hat den Index 0.

lpbState Ein Zeiger auf eine Variable, die den neuen Status des Bit erhält, entweder als ”set” (1/TRUE) oder ”cleared” (0/FALSE)

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war unkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches desgewählte Gerät.

Beispiel

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

BOOL bState;

DiiGetDigitalBit ( hDevice, 0, &bState);          // liest den Status des ersten Bits

DiiCloseDevice (hDevice);
```

Bemerkungen

8255 Devices:

Wenn der Chip zuvor noch nicht konfiguriert war, konfiguriert diese Funktion diesen Port automatisch für den Input. Falls auf einem 8255-Chip dieser Kanal schon existierte, wird die Set/Reset-Funktionalität des Chips übernommen.
Weil die Kanäle des 8255-Chips für den Input/Output rekonfiguriert werden können, beginnt der Bit-Line-Index mit dem ersten Kanal des ersten Chips aufwärts zu zählen. Dabei spielt es keine Rolle, ob er zum Zeitpunkt des Funktionsaufrufes als Input oder Output konfiguriert war.
Der Kontroll-Port des 8255-Chips ist ungeeignet für standardisierten Input/Output. So wird Kanal 24 der erste Kanal für Port A auf dem zweiten 8255-Chip des Gerätes sein (falls vorhanden).

DiiGetDigitalByte

Diese Funktiong gibt von einem digitalen Input-Port aus ein vollständiges Byte ein.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwPort Der Index des Ports auf der, den Sie lesen möchten. Der erste Port hat den Index 0.

lpbyPortState Ein Zeiger auf eine Variablen des Typs BYTE, die den neuen Status des Ports erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers kann GetLastError() folgende Werte zurückgeben:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer de Ports war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

BYTE byState;
DiiGetDigitalByte( hDevice, 0, &byState);        // liest den Status des ersten Input-Ports

DiiCloseDevice (hDevice);
```

Bemerkungen

8255 Devices:

Wenn der Chip zuvor noch nicht konfiguriert war, konfiguriert diese Funktion diesen Port automatisch für den Input.
Weil die Kanäle des 8255-Chips für den Input/Output rekonfiguriert werden können, beginnt der Port-Index mit dem ersten Kanal des ersten Chips aufwärts zu zählen. Dabei spielt es keine Rolle, ob er zum Zeitpunkt des Funktionsaufrufes als Input oder Output konfiguriert war.
Der Kontroll-Port des 8255-Chips ist für die standardisierte Input/Output-Index-Kalkulation nicht geeignet. So wird Port 3 gleich Port A auf dem zweiten 8255-Chip des Gerätes sein (falls es ihn gibt).

DiiGetOutputPort

Diese Funktion liest den Status eines Output-Ports, i.e. a bank of relays.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwPort Der Index des Output-Ports auf der Karte, die Sie lesen möchten. Der erste Port hat den Index 0.

lpbyPortState Ein Zeiger auf eine Variable des Typs BYTE, die den neuen Status des Ports erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Ports war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSetDigitalByte ( hDevice, 0, 0x12 );

BYTE byState;
DiiGetOutputPort( hDevice, 0, &byState);        // liest den Status des ersten Output-Ports

DiiCloseDevice (hDevice);
```

Bemerkungen

Einige Hardware-Adapter können den Status eine Output-Ports nicht von der Hardware lesen. In diesem Falle gibt das DII den Status des internen Port Cache`s zurück, der von den Kernel-Treibern auf Windows NT und Windows 95 gebraucht wird. Wenn das Gerät in der Lage ist, den Status eines Output-Ports zu lesen, führt das DII den Port-Input selber aus.

8.5. Funktionen zum Konfigurieren digitaler Kanäle

DiiSet8255Config

Diese Funktion konfiguriert einen 8255-Chip auf dem Gerät.

Deklaration

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

Parameter

HDevice Ein korrekter Geräte-Handle, der zuvor durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

DwChip Der Index des Chips auf der Karte, die Sie ändern möchten. Der erste Chip hat den Index O

byConfiguration Ein Byte, das die neue Konfiguration für den 8255-Chip enthält. Dieses Byte muß zu dem konfigurierten Byte passen, wie er in dem Datenblatt des 8255-Chips beschrieben wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgenden Werte zurück.

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Chips war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSet8255Config( hDevice, 0, 0x80);    // konfiguriert alle Ports als Output, Modus O
DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion konfiguriert den 8255-Chip auf der Karte.
Nachdem die Gerätekonfiguration mit dieser Funktion explizit eingestellt wurde, rekonfiguriert das DII die Ports nicht länger als Antwort auf die DiiGetDigitalByte/DiiSetDigitalByte-Funktionsaufrufe. (oder: wird der 8255 Chip von der DII nicht mehr dynamisch nachkonfiguriert?)

Mit diesem Funktionsaufruf können Sie auch direkt die Set/Reset-Eigenschaft des Chips`s ausnutzen, denn er wird direkt zum Kontroll-Port dieses 8255-Chip`s weitergeleitet.

8.6. Funktionen für den analogen Input/Output

DiiSetAnalogChannel

Diese Funktion setzt einen analogen Kanal auf einen bestimmten Wert.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index des Kanals auf den Karte, den Sie ändern möchten. Der erste Kanal hat den Index 0.

dwValue Ein 32-bit integer, der den Wert für die Ausgabe enthält. Der Wert wird gemäß der Auflösung des Gerätes reduziert.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgenden Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalbe des Bereiches des Gerätes.

Beispiel

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

DiiSetAnalogChannel( hDevice, 0,  0);   // setzt den ersten analogen Kanal auf null

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion sollten Sie benutzen, wenn Sie manuell eine Umkehrung des Spannungs-Output zum Rohwert vornehmen möchten. Einen analogen Kanal können Sie leicht mit der Funktion *DiiSetRealAnalogChannel* einstellen.

DiiSetRealAnalogChannel

Diese Funktion stellt einen analogen Kanal auf einen bestimmten Wert ein. Dazu bedarf es eines Double-Wertes in dem Bereich, der von dem Gerät unterstützt wird und/oder eines Setups durch den Nutzer.

Deklaration

```
BOOL DiiSetRealAnalogChannel(    HANDLE hDevice,
                                 DWORD dwChannel,
                                 double dValue
                             );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index des Kanals auf der Karte, den Sie ändern möchten. Der erste Kanal hat den Index 0.

dValue Ein Double-Wert (Fließkomma-Zahl), der den Wert enthält, der gesetzt werden soll. Er wird zu einem Rohwert umgesetzt, der von dem eingestellten Bereich und der Auflösung des Gerätes abhängt. Diese Umsetzung vollzieht dasDII automatisch.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgenden Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSetAnalogChannel( hDevice, 0,  3.5);// setzt den ersten Kanal auf 3.5 V

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion ermöglicht die leichte, verläßliche und geräte-unabhängige Einstellung von analogen Kanal-Werten.

DiiGetAnalogChannel

Diese Funktion stellt einen Kanal auf einen bestimmten Wert ein. Sie liefert den Rohwert von der Karte.

Deklaration

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

Parameter

hDevice Ein korrekter Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index auf der Karte, den Sie lesen möchten. Der erste Kanal hat den Index 0.

lpdwValue Ein Zeiger auf ein 32-bit integer, das den aktuellen Wert des analogen Kanals erhält. Der Wert ist gemäß der Auflösung des Gerätes reduziert.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers kann GetLastError() folgende Werte zurückbringen:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DWORD dwValue;
DiiGetAnalogChannel( hDevice, 0, &dwValue);  // liest den ersten analogen Kanal auf dem Gerät

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion gibt den Rohwert des analogen Kanals zurück, wie er von der Karte geliefert wird Sie leistet nicht die Übertragung zu dem von dem Benutzer eingestellten und von dem Gerät unerstützten Bereich. Um einen echten Wert von der Karte zu bekommen (z. B. ein Voltage), betätigen Sie bitte die Funktion DiiGetRealAnalogChannel.

DiiGetRealAnalogChannel

Diese Funktion stellt einen analogen Kanal auf einen bestimmten Wert ein. Es liefert von der Karte einen interpretierten Wert.

Deklaration

```
BOOL DiiGetRealAnalogChannel(    HANDLE hDevice,
                                 DWORD dwChannel,
                                 double * lpdValue
                             );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index des Kanals auf der Karte, den Sie lesen möchten. Der erste Kanal hat den Index 0.

lpdValue Ein Zeiger auf eine Double-Variable (Fließkomma-Zahl), die den aktuellen Wert des analogen Kanals erhält.
Dieser Wert wird entsprechend dem eingestellten Bereich des Kanals interpretiert, und zwar so, wie er vom Nutzer vorgesehen und vom Gerät unterstützt wird.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE

Im Falle eines Fehlers kann GetLastError() folgende Werte zurückbringen:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

double dValue;
DiiGetRealAnalogChannel( hDevice, 0, &dValue); // liest den ersten analogen Kanal des Gerätes

                // dValue beinhaltet nun den Wert in Volt oder den ersten analogen Kanal.

DiiCloseDevice (hDevice);
```

Bemerkungen

Mit dieser Funktion können Sie leicht und sicher den Wert von einem analogen Gerätes zurückbringen.

8.7. Funktionen zum Konfigurieren analoger Input/Output-Kanäle

Mit V1.7.0 gestartet, unterstützt das DII leistungsfähigere Konfigurationen von analogen Input- und Output-Kanälen. Es ist ein besonderes Leistungskennzeichen des DII, daß es die automatische Umkehrung von Rohwerten , wie sie von der Hardware geliefert werden, zu bedeutungsvollen Nutzer-Werten unterstützt, wie z. B. Voltages.

Darüberhinaus kann das DII automatisch alle Werte in einen besonderen, vom Benutzer eingestellten Bereich konvertieren. Wenn z. B. ein Gerät zur Messung von Strom in dem Bereich von 0 20 mA (was 0 – 10 V entspricht) eingesetzt wird, kann das DII den Bereich von 0 – 20 mA direkt zurückholen, indem esden Bereich durch die Funktion DiiSetChannelRange konfiguriert.

Bitte beachten Sie, daß die Funktion DiiSetChannelRange lediglich eine Kalkulation innerhalb des DII durchführt. Sie hat keinerlei Auswirkung auf die Hardware.

Funktionen, die die Konfiguration der Hardware ändern können, sind DiiSetChannelGain und DiiSetChannelBipolar. Sie wahren den unterstützten Werte-Bereich.

DiiSetChannelGain

Diese Funktion konfiguriert einen programmierbaren Vorverstärker auf einem analogen Gerät.

Deklaration

```
BOOL DiiSetChannelGain(     HANDLE hDevice,
                            BOOL bInput,
                            DWORD dwChannel,
                            double dGain
                       );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Vorverstärker auf einen Input-Kanal (TRUE) oder einen Output-Kanal gesetzt wird (FALSE).

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0.

dGain Eine Fließkomma-Zahl-Variable, die den Wert der Verstärkung des Kanals anzeigt.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgenden Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewünschten Gerätes.

Beispiel

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

DiiSetChannelGain (hDevice,TRUE,0,10.0);
                    // Configuriert die 10-fache Verstärkung des ersten Kanals
                    // Der Wertbereich ändert sich von 0-10 V auf 0-1 V.

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetChannelGain

Diese Funktion liefert die aktuelle Konfiguration eines programmierbaren Vorverstärkers für ein analoges Gerät.

Deklaration

```
BOOL DiiGetChannelGain(     HANDLE hDevice,
                            BOOL bInput,
                            DWORD dwChannel,
                            double * lpdGain
                       );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Vorverstärker von einem Input-Kanal (TRUE) oder einem Output-Kanal (FALSE) gelesen werden soll.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0.

lpdGain Ein Zeiger auf eine Fließkomma-Zahl-Variable, die den Wert der Vorverstärkung für den analogen Kanal erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE

Im Falle eines Fehlers kann GetLastError() folgende Werte zurückbringen:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

double dCurrentGain;
DiiGetChannelGain (hDevice,TRUE,0,&dCurrentGain);

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiSetChannelBipolar

Diese Funktion konfiguriert einen bipolaren Modus auf dem analogen Kanal.

Deklaration

```
BOOL DiiSetChannelBipolar(    HANDLE hDevice,
                              BOOL bInput,
                              DWORD dwChannel,
                              BOOL bIsBipolar
                          );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Kanal ein Input-Kanal (TRUE) oder ein Output-Kanal (FALSE) ist.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0.

bIsBipolar Ein Boolean, der anzeigt, daß der Kanal bipolar (TRUE) oder nicht bipolar (FALSE) ist.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSetChannelBipolar (hDevice,TRUE,0,TRUE);
                    // Konfiguriert den ersten Kanal für den bipolaren Modus

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetChannelBipolar

Diese Funktion liefert die aktuelle Konfiguration eines programmierbaren Vorverstärkers für ein analoges Gerät.

Deklaration

```
BOOL DiiGetChannelBipolar(   HANDLE hDevice,
                             BOOL bInput,
                             DWORD dwChannel,
                             LPBOOL lpbBipolar
                         );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Kanal ein Input-Kanal (TRUE) oder ein Output-Kanal (FALSE) ist.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0.

lpbBipolar Ein Zeiger auf eine Boolean-Variable, die anzeigt, ob der Kanal für den bipolaren oder nicht bipolaren Modus konfiguriert ist.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

BOOL bIsBipolar;
DiiGetChannelBipolar (hDevice,TRUE,0,&bIsBipolar);

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiSetChannelRange

Diese Funktion konfiguriert einen nutzerdefinierten Kanalbreich für ein analoger Gerät. Alle Werte , die von DiiGetRealAnalogChannel gelesen oder durch DiiSetRealAnalogChannel eingestellt werden, bleiben inhalb des Bereiches.

Deklaration

```
BOOL DiiSetChannelRange (   HANDLE hDevice,
                            BOOL bInput,
                            DWORD dwChannel,
                            double dMinimum
                            double dMaximum
                        );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Kanal ein Input-Kanal (TRUE) oder ein Output-Kanal (FALSE) ist.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0.

dMinimum Bestimmt den Minimum-Wert des Kanals.

dMaximum Bestimmt den Maximum-Wert des Kanals.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DiiSetChannelRange (hDevice,TRUE,0,0,0.020);
                    // ändert den Bereich des Rückgabewertes auf zum Beispiel 0-20 mA.

DiiCloseDevice (hDevice);
```

Bemerkungen

Mit dieser Funktion wird ein Konversionfilter eingerichtet. Das DII konvertiert nun automatisch alle Werte von und zu diesem Kanal aus einem nutzerdefinierten Bereich in einen Bereich, der von dem Gerät unterstützt wird.So kann der Bereich von z. B. 0-10 V auf 0-0.020 A geändert werden.

DiiGetChannelRange

Diese Funktion liefert einen nutzerdefinierten Kanalbereich für ein analoges Gerät. Alle Werte, die von DiiGetRealAnalogChannel gelesen oder durch DiiSetRealAnalogChannel eingestellt werden, bleiben innerhalb des Bereiches.

Deklaration

```
BOOL DiiGetChannelRange (   HANDLE hDevice,
                            BOOL bInput,
                            DWORD dwChannel,
                            double * lpdMinimum
                            double * lpdMaximum
                        );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Kanal ein Input-Kanal (TRUE) oder ein Output-Kanal (FALSE) ist.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index O

dMinimum Eine Variable, die den Minimum-Wert des Kanals erhält.

dMaximum Eine Variable, die den Maximum-Wert des Kanals erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

double dMinRange;
double dMaxRange;
DiiGetChannelRange (hDevice,TRUE,0,&dMinRange,&dMaxRange);

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetDeviceChannelRange

Diese Funktion liefert den geräteabhängigen Wertebereich zurück. Sie berücksichtigt die physischen Grenzen der Hardware und die aktuelle Konfiguration des Vorverstärkers.

Deklaration

```
BOOL DiiGetDeviceChannelRange (   HANDLE hDevice,
                                  BOOL bInput,
                                  DWORD dwChannel,
                                  double * lpdMinimum
                                  double * lpdMaximum
                               );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

bInput Bestimmt, ob der Kanal ein Input-Kanal (TRUE) oder ein Output-Kanal (FALSE) ist.

dwChannel Der Index des Kanals auf der Karte. Der erste Kanal hat den Index 0

dMinimum Eine Variable, die den Minimum-Wert des Kanals erhält.

dMaximum Eine Variable, die den Maximum-Wert des Kanals erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder die Nummer des Kanals war außerhalb des Bereichs des gewählten Gerätes.

Beispiel

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

double dMinRange;
double dMaxRange;
DiiGetDeviceChannelRange (hDevice,TRUE,0,&dMinRange,&dMaxRange);

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion bringt den Kanalbereich des Gerätes zurück. Er kann vom Nutzer nicht verändert werden, außer, er stellt den Vorverstärker eines Kanals ein. Die meisten Geräte unterstützen einen Kanalbereich von 0-10 V ohne Vorverstärkung. Für Geräte, die einen programmierbaren Vorverstärker unterstützen, kann sich dieser Bereich ändern.

8.8. Funktionen zum Einrichten des Timers auf Karten

DiiSetTimerConfig

Diese Funktion konfiguriert Timer-Chips auf dem Gerät.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwTimer Der Index des Timers auf der Karte. Der erste Timer hat den Index 0.

dwConfig Ein 32-bit-Integer, der die Konfiguration des Timers enthält. Bei 8253-Timern wird nur das niedrigste Byte benutzt.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder der Timer-Index war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

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

DiiCloseDevice (hDevice);
```

Bemerkungen

8253 Chips:
Diese Funktion setzt das zu konfigurierende Byte eines Timers auf einen 8253-Chip. Diese Konfiguration wird solange nicht eingestellt bis DiiLoadTimer aufgerufen ist.

DiiLoadTimer

Diese Funktion lädt einen Wert in einen Timer

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwTimer Der Index des Timers auf der Karte. Der erste Timer hat den Index 0.

dwValue Ein 32-bit-Integer, der den Timer-Wert enthält, der geschrieben werden soll.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder der Timer-Index war außerhalb des Bereichs des gewählten Gerätes

Beispiel

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

DiiLoadTimer( hDevice, 0, 40000);       // lädt den Wert in den ersten Timer

DiiCloseDevice (hDevice);
```

Bemerkungen

8253 Chips:
Diese Funktion stellt die Konfiguration des Timers ein und lädt den Timer gemäß der 8253-Spezifikationen.

DiiGetTimer

Diese Funktion lädt einen Wert in einen Timer.

Deklaration

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

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwTimer Der Index des Timers auf der Karte. Der erste Timer hat den Index 0.

lpdwValue Ein Zeiger auf ein 32-bit-Integer, der den aktuellen Timer-Wert erhält.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER – Der übergebene Handle war inkorrekt, oder der Timer-Index war außerhalb des Bereiches des gewählten Gerätes.

Beispiel

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

DWORD dwValue;
DiiGetTimer( hDevice, 0, &dwValue);     // liest den aktuellen Wert des Timers

DiiCloseDevice (hDevice);
```

Bemerkungen

8253 Chips:
Eine Counter-Latching-Operation wird durchgeführt, so daß das Ablesen des Timers nicht die laufende Zeitmessung stört.

8.9. Funktionen zur Handhabung standardisierter Nachrichten

Die folgenden Strukturen und Konstanten werden für den Nachrichtenmechanismus gebraucht. Bitte beachten Sie, daß diese für die C-Sprache gelten. Andere Sprachen können für den einfacheren Zugriff auf Nachrichten-Funktionen den OCX benutzen. Für die einfachere Handhabung sind in OCX Nachrichten als standardisierte OCX-Ereignisse implementiert.

Bitte beachten Sie: Nachrichten wie auch OCX-Ereignisse werden nur von I/O-Geräte unterstützt, die Hardware-Interrupts unterstützen. Wir arbeiten zur Zeit daran, die Nachrichtenunterstützung auch für ältere Karten möglich zu machen. Wir empfehlen Ihnen, den Boolean-Rückgabewert mit **DiiRequestNotification** zu überprüfen, um sicher zu gehen, daß das Gerät die Benachrichtigung unterstützt.

Im Folgenden werden die Nachrichtentypen definiert, die angefordert werden können. Die verschiedenen Benachrichtigungen können durch ein bit-weises OR kombiniert werden.

```
DII_TIMER_EXPIRED  0x000000001
        //              Kann aufgerufen werden, wenn ein Timer abgelaufen ist.
DII_SIGNAL_CHANGED 0x00000002
        //              Kann aufgerufen werden, wenn sich die Signale auf der Karte verändert haben.
DII_DATA_BLOCK_READ 0x00000004
        //              Kann aufgerufen werden, wenn Hintergrund-Daten verfügbar sind.
DII_DATA_BLOCK_SENT 0x00000008
        //              Kann aufgerufen werden, sobald ein Datenblock gesendet wurde.
DII_VALUE_OUT_OF_RANGE 0x00000010
        //              (für den zukünftigen Gebrauch reserviert)
DII_COUNTER_ZERO 0x00000020
        //              Kann aufgerufen werden, wenn ein Rückzähler Null erreicht hat.

        //
        // Die folgende Datenstruktur wird mit der Benachrichtigung
        // an die Nutzer-Anwendung gesendet:
        //

typedef struct _tagDII_NOTIFICATION_DATA
{
        HANDLE      hDevice;            // Identifiziert das Gerät, das die Nachricht ausgelöst hat.

        DWORD       dwNotificationMask; // Eine Bitmaske, die die Nachricht identifiziert
                                                    // fragt an, wofür die Daten sind

        DWORD       dwNotificationParam;
                                        // Ein 32-Bit-Parameter, der mit der Nachricht übergeben
                                        // wird.
                                        // Dieser kann eine Bitmaske für die Signale enthalten, die
                                        // die Nachricht geändert haben
                                        // oder den Counter-Inex enthalten, der Null erreicht hat.

        LPVOID      lpBuffer;           // Der Puffer, der Ereignisdaten enthält ode NULL
        DWORD       dwBufferSize;       // Die Größe des Puffer, auf die mit lpBuffer gezeigt wird.

        DWORD       dwUserData;         // Ein 32-bit-integer für Nutzerdaten, die in
                                        // DiiRequestNotification übergeben werden.

        DWORD       dwAllocationScheme; // Intern: Das Zuteilungs-Schema für diesen Nachrichten-
                                        // Datenblock. Bitte ändern Sie ihn nicht. DO NOT
                                        MODIFY!
```

} DII_NOTIFICATION_DATA, * LPDII_NOTIFICATION_DATA;

typedef void (*PDII_NOTIFY_CALLBACK)(LPDII_NOTIFICATION_DATA);

Eine Tabelle für die Interpretation der Parameter und ihre Abhängigkeit vom Nachrichtentyp:

Nachrichten-Typ:	dwNotificationParam	LpBuffer
DII_TIMER_EXPIRED	Ein null-basierte Index des Timers, der abgelaufen ist oder Null erreicht hat.	Nicht gebraucht.
DII_SIGNAL_CHANGED	Eine Bitmaske, die alle Signale enthält, die sich geändert haben. Zur Zeit auf 32 Signale begrenzt.	Nicht gebraucht.
DII_DATA_BLOCK_READ	<noch nicht implementiert>	<noch nicht implementiert>
DII_DATA_BLOCK_SENT	<noch nicht implementiert>	<noch nicht implementiert>
DII_VALUE_OUT_OF_RANGE	<noch nicht implementiert>	<noch nicht implementiert>
DII_COUNTER_ZERO	Ein null-basierter Index des Zählers, der auf Null zurückgezählt hat.	Nicht gebraucht.

DiiSetNotificationMethodHwnd

Diese Funktion registriert eine Window-Message mit dem Nachrichten-Mechanismus.

Deklaration

```
BOOL DiiSetNotificationMethodHwnd (  HANDLE hDevice,
                                     HWND hWnd,
                                     UINT    msg,
                                     WPARAM wParam
                                     );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

hWnd Ein Handle zu einem Fenster, an das Nachrichten-Messages gesandt werden.

msg Die zu sendende Nachricht. Der WPARAM-Parameter der Message ist gegeben. Der LPARAM-Parameter der Message zeigt auf eine DII_NOTIFICATION_DATA-Struktur, die weitere Informationen über die Nachricht enthält.

wParam Der Parameter, der mit der Message übergeben wird.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Einer der übergebenen Handle war inkorrekt, oder das Gerät unterstützt die Nachricht generell nicht.

Beispiel

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

DiiSetNotificationMethodHwnd (hDevice,
                              AfxGetMainWnd()->m_hWnd,
                                    WM_USER+1,
                              0);

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion registriert eine Message, die jedesmal dann gesendet wird, wenn ein Gerät die Benutzer-Anwendung über ein Ereignis benachrichtigen soll. Bevor eine Anwendung Ereignisse erhält, muß **DiiRequestNotification** nach dieser Funktion aufgerufen werden. Diese Funktion bestimmt lediglich, wie Nachrichten gesendet werden.

Bitte beachten Sie: Die Kernel-Treiber verfügen über ein eigenes intelligentes Puffer-Schema, um Hintergrund-Daten lesen zu können. Nachdem Sie die Window-Message bearbeitet haben, müssen Sie die DII_NOTIFICATION_DATA-Struktur mit dem Funktionsaufruf **DiiReleaseNotificationData** aus dem Speicher freigeben. Berücksichtigen Sie bitte, daß es zu einer Verzögerung zwischen der Nachricht und dem Empfang der Window Message kommen kann. Dies liegt daran, daß die Message nicht synchron, sondern asynchron zu dem Zielfenster geschickt wird.

DiiSetNotificationMethodCallBack

Diese Funktion registriert eine Callback-Funktion mit dem Nachrichtenmechanismus

Deklaration

```
BOOL DiiSetNotificationMethodCallBack(      HANDLE hDevice,
                                            PDII_NOTIFY_CALLBACK pfnCallBack
                                            );
```

Parameters

hDevice Ein korrekter Geräte-Handle, der vorher duch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

pfnCallBack Ein Zeiger auf eine Funktion, die jedesmal dann aufgerufen wird, wenn eine Benachrichtigung aussteht.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Einer der übergebenen Handle war inkorrekt oder das Gerät unterstützt die Nachricht generell nicht.

Beispiel

```
        // Vorwärts-Deklaration
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);

...

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

DiiSetNotificationMethodCallBack (hDevice,
                                  &ProcessDeviceNotification
                                  );

DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion registriert eine Callback-Funktion, die jedesmal dann aufgerufen wird, wenn das Gerät die Benutzer-Anwendung über ein Ereignis benachrichtigt. Bevor eine Anwendung Ereignisse erhält, muß **DiiRequestNotification** nach dieser Funktion aufgerufen werden. Diese Funktion bestimmt lediglich, wie Nachrichten versandt werden.

Note: Diese Funktion wird ein einem anderen Thread aufgerufen. Deshalb müssen Sie eine Synchronisation mit dem Hauptcode herstellen.

DiiRequestNotification

Diese Funktion fordert vom Gerät Benachrichtigungen über spezielle Ereignisse, die geschehen können.

Deklaration

```
BOOL DiiRequestNotification    (    HANDLE hDevice,
                                    DWORD dwNotificationMask,
                                    DWORD dwUserValue
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice übergeben wurde.

dwNotificationMask
Ein 32-bit-Integer, der die Benachrichtigungen spezifiziert, die von dem Gerät gefordert werden.

dwUserValue Ein 32-bit-Integer, der innerhalb der DII_NOTIFICATION_DATA Daten-Struktur zurückgeben wird.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Einer der übergebenen Handle war inkorrekt oder dasGerät unterstützt generell keine der geforderten Benachrichtigungen.

Beispiel

```
        // Vorwärts-Deklaration...
    void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);

    ...

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

    DiiSetNotificationMethodCallBack (hDevice,
                                      &ProcessDeviceNotification
                                      );

    DiiRequestNotification (hDevice, DII_SIGNAL_CHANGED, 0);

    DiiCloseDevice (hDevice);
```

Bemerkungen

Diese Funktion aktiviert den Benachrichtigungs-Mechanismus. Bevor Sie diese Funktion aufrufen, achten Sie darauf, daß entweder **DiiSetNotificationMethodHwnd** oder **DiiSetNotificationMethodCallBack** aufgerufen werden muß, um das DII darüber zu informieren, wie Nachrichten an die Benutzer-Anwendung zurückgesandt werden können.

DiiCancelNotification

Diese Funktion bricht vorher geforderte Benachrichtigungen ab. D. h. Benachrichtigungen mit den abgebrochenen Ereignissen werden nicht mehr an die Benutzer-Anwendung gesandt.

Deklaration

```
BOOL DiiCancelNotification     (     HANDLE hDevice,
                                     DWORD dwNotificationMask
                                     );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwNotificationMask
Ein 32-bit-Intger, das die Benachrichtigungen spezifiziert, die von dem Gerät abgebrochen werden.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt.

Beispiel

```
        /Vorwärts-Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);

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

if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       )) return;        // Das Gerät unterstützt keine Benachrichtigungen

if (!DiiRequestNotification (hDevice, DII_SIGNAL_CHANGED, 0)) return;
                    // Fordert immer dann eine Benachrichtigung, wenn sich ein Signal verändert hat.
getch();            // Warten Sie, bis ein Key gedrückt wurde, Benachrichtigungen werden im Hinter-
                      grund erhalten.
DiiCancelNotification (hDevice, 0);       // bricht alles ab.

DiiCloseDevice (hDevice);
```

Bemerkungen
Diese Funktion aktiviert den Benachrichtigungs-Mechanismus. Bevor Sie diese Funktion aufrufen, achten Sie darauf, daß entweder **DiiSetNotificationMethodHwnd** oder **DiiSetNotificationMethodCallBack** aufgerufen werden muß, um die DII darüber zu informieren, wie Nachrichten an die Benutzer-Anwendung zurückgesandt werden können.

Beachten Sie: Der *dwNotificationMask-P*arameter wird ignoriert. Durch den Aufruf von **DiiCancelNotification** werden alle vorher geforderten Nachrichten abgebrochen.

DiiReleaseNotificationData

Diese Funktion muß aufgerufen werden, wenn Nachrichten zu einer Window Message geschickt werden sollen, um die vorhandene Nachrichten-Struktur freizugeben. Es ist nämlich nutzlos, sie über die Datenstruktur aufzurufen, die zu DII_NOTIFY_CALLBACK übergeben wurde.

Deklaration

```
BOOL DiiReleaseNotificationData (       HANDLE hDevice,
                                        LPDII_NOTIFICATION_DATA lpNotifData
                                );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

lpNotifData Ein Zeiger auf die DII_NOTIFICATION_DATA –Struktur, die durch eine Message erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Beispiel

```
// Einen Message-Handler in der MFC vorausgesetzt:

void CSampleClass::OnMessage (WPARAM,LPARAM lParam)
{
        LPDII_NOTIFICATION_DATA lpNotifData = LPDII_NOTIFICATION_DATA (lParam);

        // arbeitet auf irgendeine Weise mit der Nachricht

        // befreit die Benachrichtigungs-Struktur:
        DiiReleaseNotificationData (    lpNotifData->hDevice,
                                        lpNotifData );
}
```

Bemerkungen

Diese Funktion wird aufgerufen, um die Benachrichtigungs-Struktur freizugeben, die in eine Windows Message übergeben wurde. Sie ist nur dann notwendig, wenn Benachrichtigungen über Windows Messages angefordert werden, denn diese Messages werden asynchron bearbeitet.

Bitte beachten Sie: Wird diese Funktion nicht aufgerufen, so hat das einen konstant ansteigenden Speichergebrauch durch das DII zur Folge. Möglich ist auch ein Datenverlust, da Puffer nicht wieder benutzt werden können.

8.10. Funktionen für den Zugriff auf Kernel-Modus-Impuls-Zähler

Das Pulse Counter Interface mißt digitale Impulse von Hochsprachen aus. Vorher konnten die Impulse nur gemessen werden, indem die I/O-Ports direkt von Hochsprachen aus gepollt wurden, oder, falls das Gerät Interupts unterstützte, indem diese Interrupts innerhalb der Hochsprache abgearbeitet wurden.
Beide Vorgehensweisen haben Nachteile. Das Polling ist nicht verläßlich bei Multi-Tasking-Betriebssystemen wie Windows NT und Windows 95, und das Abarbeiten von Interrupts in einer Hochsprache führt zu einer Verlangsamung.

Theoretisch können Impulse gezählt werden, indem man Benachrichtigungen des Typs "DII_SIGNAL_CHANGED" von der DII anfordert. Das würde dazu führen, daß eine Nachricht jedesmal dann gesendet würde, wenn sich ein Signal auf einer interrupt-betriebenen I/O-Karte geändert hätte. Bitte beachten Sie, daß diese Art der Impuls-Zählung eine verlangsamte Reaktionszeit zur Folge hätte. Es ist so, als gingen Impule verloren.

Um diese Probleme zu lösen, implementiert das DII nun ein Pulse Counter Interface. Es arbeitet sowohl mit den älteren I/O-Karten als auch mit den neueren interrupt-getriebenen Karten. Bei älteren Karten führt die DII Link Link Library selbst das Polling aus. Das ist vorteilhafter als es in einer Hochsprache wie z. B. Visual Basic zu tun, welche aber immer noch Gegenstand der Prozessor-Einteilung ist. Bei interrupt-getriebenen Karten wird die Impuls-Zählung direkt von den Kernel-Treibern für Windows 95 und Windows NT vorgenommen.

Das Pulse Counting Interface zeichnet sich durch folgende Leistungen aus:

1. Jeder Kanal auf einer digitalen I/O-Karte kann Individuell als ein Vor- und Zurück-Zähler konfiguriert werden. Er kann individuell zu jeder beliebigen Zeit vor- und zurückgesetzt werden.
2. Mit der tatsächlichen Anzahl der Impulse wird die minimale und maximale Zeit zwischen zwei Impulsen zurückgegeben.
3. Bei interrupt-getriebenen Karten kann eine "DII_COUNTER_ZERO" –Nachricht gesendet werden, wenn ein Zurückzähler Null erreicht hat.
4. Zurückzähler können individuell so eingestellt werden, daß sie nach dem Zurückzahlen bis auf Null sich selbst wieder auf den ursprünglichen Wert setzen.

Das Pulse Counting Interface ist mit den folgenden Funktionen implementiert :

DiiEnablePulseCounting

Diese Funktion ermöglicht die Impuls-Zählung für eine bestimmte Karte

Deklaration

```
BOOL DiiEnablePulseCounting (   HANDLE hDevice,
                                DWORD dwStartPort,
                                DWORD dwNumberOfPorts,
                                DWORD dwReserved
                                );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwStartPort Der Start I/O-Port, der überwacht werden soll. Um alle Ports zu überwachen, übergeben Sie bitte 0. Dieser Parameter wird ignoriert, wenn die Karte Interrupts unterstützt.

DwNumberOfPorts
Die Anzahl der zu scanenden oder zu unterstützenden Ports. Um alle Ports zu überwachen, übergeben sie bitte 0. Dieser Parameter wird ignoriert, wenn die Karte Interrupts unterstützt.

dwReserved Reserviert für zukünftige Verwendung

Rückgabewert

TRUE falls erfolgreich, anderfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt

Beispiel

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

if (!DiiEnablePulseCounting (hDevice,        // ermöglicht die Impuls-Zählung auf allen Ports
                             0,              // zurück, falls die Impuls-Zählung nicht
                                             unterstützt wird.
                             0,
                             0)) return;

getch();                     // warten Sie auf einen Tastendruck, die Impuls-Zählung wird im
                             Hintergrund vorgenommern

DiiDisablePulseCounting (hDevice);    // stoppt die Impuls-Zählung wieder

DiiCloseDevice (hDevice);
```

Bemerkungen

Mit dieser Funktion können Sie die Impuls-Zählung auf den vorgesehenen Ports starten.Bei nicht Interrupt fähigen Geräten wird die Impuls-Zählung mit 10Hz vorgenommen. Bei interrupt-getriebenen Karten wird das Gerät auf den Interrupt-Modus eingestellt. Im Falle eines Fehlers werden alle Zähler als Aufwärtszähler konfiguriert.

DiiDisablePulseCounting

Diese Funktion stoppt die Impuls-Zählung für eine bestimmte Karte

Deklaration

```
BOOL DiiDisablePulseCounting(       HANDLE hDevice
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt

Beispiel

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

if (!DiiEnablePulseCounting (hDevice,          // stoppt die Impuls-Zählung auf allen Ports.
                             0                 // zurück, falls die ImpulsZählung nicht
                                               unterstützt wird.
                             0,
                             0)) return;

getch();                  // warten Sie auf einen Tastendruck. Die Impuls-Zählung vollzieht sich im
                          Hintergrund.

DiiDisablePulseCounting (hDevice);      // stoppt die Impuls-Zählung wieder.

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiSetDownPulseCounter

Diese Funktion wird aufgerufen, um einen Zähler als Aufwärts- oder Abwärts-Impulszähler zu rekonfigurieren.

Deklaration

```
BOOL DiiSetDownPulseCounter     (     HANDLE hDevice,
                                      DWORD dwChannel,
                                      DWORD dwInitialValue,
                                      BOOL bAutoReset
                                );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der zu rekonfigurierende Kanal. Der Kanal korrespondiert mit einem einzelnen digitalen I/O-Kanal. Der erste I/O-Kanal wird mit 0 übergeben.

dwInitialValue Der Ausgangswert für den Zähler. Falls der nicht Null ist, wird der Zähler als Zurückzähler konfiguriert. Ist der Wert Null, wird der Zähler als Aufwährts-Zähler konfiguriert. Der Zähler stellt sich selbst auf diesen Wert ein.

bAutoReset Bestimmt, ob sich ein Zurück-Zähler selbst automatisch auf den Ausgangswert zurückstellt, wenn er bis Null zurückzählt. Wenn der Zähler auf Aufwärts-Zähler konfiguriert ist (dwInitialValue ist zero), wird dieser Parameter ignoriert.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE..

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der Kanal liegt außerhalb des Bereiches.

Beispiel

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

if (!DiiEnablePulseCounting (hDevice,        // stoppt die Impuls-Zählung auf allen Ports
                             0,              // zurück, falls die Impuls-Zählung nicht
                                             unterstützt wird
                             0,
                             0)) return;

DiiSetDownPulseCounter (hDevice,0,1000,TRUE);
        // konfiguriert den ersten Kanal, um von 1000 zurückzuzählen und sich
    automatisch zurückzustellen.
getch();            // warten sie auf einen Tastendruck. Impuls-Zählung vollzieht sich im
                    Hintergrund

DiiDisablePulseCounting (hDevice);      // stoppt Impuls-Zählung wieder

DiiCloseDevice (hDevice);
```

Bemerkungen

DiiGetPulseCounterValue

Diese Funktion wird aufgerufen, um einen Zähler als Aufwärts- oder Abwärtszähler zu rekonfigurieren.

Deklaration

```
BOOL DiiGetPulseCounterValue    (    HANDLE hDevice,
                                     DWORD dwChannel,
                                     LPDWORD lpdwValue,
                                     LPDWORD lpdwMinTimeBetweenPulses,
                                     LPDWORD lpdwMaxTimeBetweenPulses,
                                     BOOL bResetCounter
                                );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der zu rekonfigurierende Kanal. Der Kanal korrespondiert mit einem einzelnen digitalen I/O-Kanal. Der erste I/O-Kanal wird mit Null übergeben.

lpdwValue Ein Zeiger auf eine Variable, die den aktuellen Stand des Zählers anzeigt.

lpdwMinTimeBetweenPulses
Ein Zeiger auf eine Variable, die die geringste Zeit zwischen zwei Impulsen anzeigt. Die Zeit wird in Mikrosekunden angezeigt, obwohl die tatsächliche Auflösung des Timers niedriger ist.

lpdwMinTimeBetweenPulses
Ein Zeiger auf eine Variable, die die maximale Zeit zwischen zwei Impulsen anzeigt. Die Zeit wird in Mikrosekunden angezeigt, obwohl die tatsächliche Auflösung des Timers niedriger ist.

bResetCounter Bestimmt, ob ein Zähler während des Lesens zurückgestellt wird. Falls er zurückgestellt wird, wird ein Aufwärts-Zähler auf Null gesetzt und ein Abwärtszähler wieder auf den Ausgangswert gestellt. Ist dieser Parameter FALSE, wird der Zähler nicht zurückgestellt.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() die folgenden Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der Kanal liegt außerhalb des Bereiches.

Beispiel

```
DWORD dwValue;
DWORD dwMinTimeBetweenPulses;
DWORD dwMaxTimeBetweenPulses;

if (!DiiGetPulseCounterValue (  hDevice, 0,
                                &dwValue, &dwMinTimeBetweenPulses,
                                &dwMaxTimeBetweenPulses,
                                TRUE)) return;
        // liest einen Zähler und stellt ihn zurück
```

Bemerkungen

8.11. Funktionen für die kontinuierliche Hintergrund-Meßdatenerfassung

Zusätzlich zum standardisierten polling-basiertem Lesen von analogen oder digitalen Werten bzw. dem Schreiben dieser Werte auf eine Karte unterstützt das DII auch das block-basierte Lesen bzw Schreiben von großen Mengen von Meßdaten auf und von Geräten zur Meßdatenerfassung. Das DII optimiert intern automatisch die Speicherverwaltung, so daß die kontinuierliche Meßdatenerfassung gesichert ist. Das DII erzeugt und löscht selbsttätig alle Puffer, die erforderlich sind, um einen kontinuierlichen Strom von Input- und Output-Daten zu und von der Anwendung zu erzeugen.

Das DII abstrahiert die Funktionalität der Hardware, indem es von sich aus den besten Weg zur Konfigurierung der zugrundliegenden Hardware unterstützt. Einige Geräte können den DMA-Transfer und Interrupts unterstützen, andere brauchen ein internes Polling durch das DII.

Die Blockschnittstelle ist eng mit der standardisierten Schnittstelle für Benachrichtigungen in dem DII verbunden (wie auf S. 45ff dargestellt), um der Benutzeranwendung die erhaltenen Puffer anzuzeigen. Diese Benachrichtigung ist asynchron, entweder durch Active X-Ereignisse oder durch Function Callbacks.

Die Meßdaten-Schnittstelle arbeitet sowohl mit digitalen wie mit analogen Werten. Allerdings wird nur ein Typ pro Meßgerät unterstützt. Beim Lesen von analogen Kanälen sind die zurückgegebenen Daten Rohwerte. Weitere Informationen zur Konfiguration von analogen Kanälen finden Sie in Kapitel 8.7. (S. 34 ff).

Das DII unterstützt zwei Auslösemechanismen: Einen internen über einen Timer auf der Hardware oder einen externen über einen externen Input-Kanal.

Ein besonderes Leistungsmerkmal der Blockschnittstelle besteht darin, daß die Abtastrate und die interne Puffergröße für jeden Kanal gewählt werden können. Diese Funktionalität können allerdings nur solche Karten bieten, die pro Kanal für diese Parameter konfigurierbar sind.

Für den Fall, daß das Gerät der Hardware die Daten schneller liefert als die Benutzeranwendung diese auswerten kann, unterstützt das DII die Speicherung auf Festplatte. Die Daten werden also zuerst auf der Festplatte gespeichert und dann später über die gleiche Schnittstelle in den Speicher zurückgeholt.

Die Blockschnittstelle bietet grundsätzlich zwei Möglichkeiten , wie der Benutzer die Datengröße so konfigurieren kann, daß die Anwendung die Daten verarbeitet. Für jeden Kanal können die Frequenz und Puffer-Größe konfiguriert werden. Die Puffer-Größe ist immer in Bytes angegeben. Um die Anzahl der Puffer zubestimmen, die die Anwendung von einem spezifischen Kanal pro Sekunde erhält, gehen Sie bitte in der folgenden Weise vor:

*Anzahl von Meßdaten pro Sekunde = Kanalfrequenz / (Puffergröße * Bytes pro Wert))*

Deshalb ist die Anzahl der Daten-Blocks, die von der Benutzer-Schnittstelle an die Benutzeranwendung übergeben werden, gleich der Frequenz, auf der die Meßwerte erhalten werden, geteilt durch die Größe der Daten-Puffer eines jeden Kanals. Bitte beachten Sie dabei, daß diese Größe in Bytes angegeben wird. Deshalb müssen Sie die Anzahl von Bytes bestätigen, die für einen Wert genommen werden. Für digitale I/0-Karten ist ein Datenwert in der Regel 8-Bit groß. Bei analogen I/O-Karten ist die Datengröße abhängig von der Auflösung des Gerätes. Bei Karten mit einer Auflösung von genau 16 Bit oder weniger beträgt die Datengröße 2 Bytes pro Wert, bei höherer Auflösung 4 Bytes pro Wert.

DiiPrepareBlockInput

Diese Funktion wird aufgerufen, um einen spezifischen Kanal für einen blockweisen Input vorzubereiten.

Deklaration

```
BOOL DiiPrepareBlockInput    (    HANDLE hDevice,
                                  DWORD dwChannel,
                                  DWORD dwTriggerSource,
                                  DWORD dwTriggerValue,
                                  DWORD dwSampleBufferSize,
                                  LPCTSTR lpszFileName
                             );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index des Input-Kanals, der automatisch gescannt werden soll. (0 ist der erste Kanal)

dwTriggerSource Enthält den Auslösemechanismus für die Messung.

Specify 0 – für einen software-emulierten Auslöser,
Specify 1 – für einen Auslöser, der auf einen ON-Board-Timer basiert,
Specify 2 – für einen Trigger, der auf einer externen Quelle basiert, mit der er durch einen digitalen Input-Kanal verbunden ist.

dwTriggerValue Enthält die Frequenz von Messungen pro Sekunde, falls ein interner Timer benutzt wird oder die Kanalnummer, falls ein externes Signal zur Ausführung der Messung dient.

dwSampleBufferSize Bestimmt die Größe der Puffer, die für diesen Kanal zurückgegeben werden. Das ist die binäre Größe des Datenpuffers, nicht die Anzahl der Meßwerte. Bei einer analogen I/O-Karte verbraucht eine Messung für gewöhnlich 2 Bytes. Wählen Sie deshalb eine Größe von 2000, um 1000 Werte pro Puffer zu erhalten.

lpszFileName Der frei zu wählende Name einer Datei, in die Daten geschrieben werden, um sie direkt auf der Festplatte zur Verfügung zu haben..

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
```

```
                                    &ProcessDeviceNotification
                                    ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_READ, 0))
{
        '' could not request notification about data blocks being read
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                0,              // mißt den ersten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                30000,          // versucht  30 kHz zu messen
                                60000,          // sendet 30000 AD Messungen pro Block
                                NULL            // keine I/O-Datei
                ))
{
        '' Coulnd not initialize the channel for block input.
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                1,              // mißt den zweiten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                10000,          // versucht 10 kHz zu messen
                                10000,          // sendet 5000 AD Messungen pro Block
                                NULL            // keine I/O-Datei
                ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}
```

Bemerkungen

Benutzen Sie diese Funktion, wenn Sie einen einzelnen Kanal für einen Block-Input konfigurieren möchten. Diese Funktion kann sofort für andere Kanäle aufgerufen werden, bevor Sie DiiStartBlockInput eingeben.

DiiStartBlockInput

Diese Funktion wird aufgerufen, um den zuvor konfigurierten Block-Input zu starten.

Deklaration

```
BOOL DiiPrepareBlockInput    (      HANDLE hDevice,
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_READ, 0))
{
        '' could not request notification about data blocks being read
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                0,              // mißt den ersten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                30000,          // versucht 30 kHz zu messen
                                60000,          // sendet 30000 AD Messungen per Block
                                NULL            // keine I/O-Datei
                        ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}
```

```
if (!DiiPrepareBlockInput (     hDevice,


                                1,              // mißt den zweiten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                10000,          // versucht 10 kHz zu messen
                                10000,          // sendet 5000 AD Messungn per Block
                                NULL            // keine I/O-Datei
                ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}

if (!DiiStartBlockInput  (      hDevice))
{
        '' cannot start block input
        return FALSE;
}
```

Bemerkungen

Diese Funktionen können aufgerufen werden, nachdem die individuellen Kanäle für den Block-Input vorbereitet worden sind.

DiiEndBlockInput

Diese Funktion stoppt den block-basierten Input von Datenmessungen

Deklaration

```
BOOL DiiEndBlockInput       (       HANDLE hDevice,
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Funktionen zurück

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_READ, 0))
{
        // could not request notification about data blocks being read
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                0,              // mißt den ersten Kanl
                                0,              // benutzt den internen Timer als Pacer
                                30000,          // versucht 30 kHz zu messen
                                60000,          // sendet 30000 AD Messungen pro Block
                                NULL            // keine I/O-Datei
                        ))
{
        // Could not initialize the channel for block input.
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                1,              // mißt den zweiten Kanal
                                0,              // benutzt den internen Timer als Pacer
```

```
                                        10000,          // versucht  10 kHz zu messen
                                        10000,          // sendet 5000 AD Messungen per Block
                                        NULL            // keine I/O-Datei
                        ))
        {
                // Could not initialize the channel for block input.
                return FALSE;
        }

        if (!DiiStartBlockInput    (        hDevice))
        {
                // cannot start block input
                return FALSE;
        }

        //
...     // Hier eventuell weiterer Code, z. B. die Bearbeitung von Benachrichtigungen des Gerätes.

        if (!DiiEndBlockInput (         hDevice))
        {
                return FALSE;
        }
```

Bemerkungen

Stoppt die im Hintergrund arbeitende Blockschnittstelle.

DiiGetBlockInputStatus

Diese Funktion bringt Status-Informationen über den Block-Input für einen spezifischen Kanal zurück.

Deklaration

```
BOOL DIIEXPORT DiiGetBlockInputStatus    (
                                         HANDLE hDevice,
                                         DWORD dwChannel,
                                         LPDWORD lpdwTotalSamples,
                                         LPDWORD lpdwSamplesPerSecond,
                                         LPDWORD lpdwBufferUnderruns
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal, für den Informationen zurückgebracht werden sollen.

lpdwTotalSamples
Die gesamte Anzahl der bisher gelesenen Daten-Meßwerte (*nicht Bytes!*).

lpdwSamplesPerSecond
Die durchschnittliche Anzahl von Daten-Messungen pro Sekunde.

lpdwBufferUnderruns
Die Anzahl der bisherigen Puffer-Unterläufe. Puffer-Unterläufe treten auf, wenn die Hardware Daten schneller liefert als das DII sie verarbeiten kann.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt, oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        //
        // Lesen Sie dazu bitte das Beispiel, wie Sie den Block-Input einrichten und starten
können.

    DWORD dwTotalSamples;
    DWORD dwSamplesPerSecond;
    DWORD dwBufferUnderruns;

    if (DiiGetBlockInputStatus (        hDevice,
                                        0,          // liefert ein Info für den ersten Kanal
                                        &dwTotalSamples,
                                        &dwSamplesPerSecond,
                                        &dwBufferUnderruns
                                ))
    {
```

```
        // evaluiert die statistische Information
}
```

Bemerkungen

Diese Funktion bringt statistische Informationen während des Block-Inputs zurück. Die Benutzer-Anwendung achtet selbsttätig darauf, daß alle Daten korrekt geliefert werden, indem sie die fehlerhaften Puffer-Unterläufe auswertet. Puffer-Unterläufe können aus zwei Gründen auftreten: Sie ereignen sich einmal, wenn die Hardware Daten schneller sendet als das DII sie verarbeiten kann und sie so der Benutzer- Anwendung zustellt. Zum anderen treten sie auf, wenn die Benutzer-Anwendung die Daten durch das DII nicht schnell genug bearbeitet, so daß dem DII keine Puffer mehr zur Verfügung stehen.

DiiOpenBlockInputFile

Diese Funktion wird aufgerufen, um eine vorher erstellte Block-Input-Datei wieder zu öffnen.

Deklaration

```
BOOL DiiOpenBlockInputFile    (       HANDLE hDevice,
                                      DWORD dwChannel,
                                      LPCTSTR lpszFileName
                              );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal, für den die Datei geöffnet wurde.

lpszFileName Der Datei-Name der Datei.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
BYTE byaDataBuffer[4096];
DWORD dwDataRead;

if (!DiiOpenBlockInputFile      (hDevice,
                                0,
                                “c:\\temp\\Channel0Dump.Bin”))
        return;         // Datei kann nicht geöffnet werden

if (DiiReadBlockInputFile       (hDevice,
                                0,
                                byaDataBuffer,
                                4096,
                                &dwDataRead))
        {       // erfolgreich, Daten können ausgewertet werden      }

DiiCloseBlockInputFile          (hDevice,0);
```

Bemerkungen

Diese Funktion öffnet eine Datei, die vorher durch die Block-Input-Funktion erstellt worden ist. Bei der Meßdatenerfassung sollte die Abspeicherung auf Festplatte immer dann vorgenommen werden, wenn die Menge der erhaltenen Daten durch die Anwendung nicht gleichzeitig ausgewertet und verarbeitet werden können. Die Anwendung kann dann die Daten mit einer hohen Abtastrate auf der Festplatte speichern und sie später über eine Online-Meßdatenerfassung zurücklesen, um die Datenzeit unkritisch zu verarbeiten.

DiiReadBlockInputFile

Diese Funktion wird aufgerufen, um Daten aus einer geöffneten Block-Input-Datei zu lesen. .

Deklaration

```
BOOL DiiReadBlockInputFile    (       HANDLE hDevice,
                                      DWORD dwChannel,
                                      LPVOID lpvBuffer,
                                      DWORD dwBufferSize,
                                      LPDWORD lpdwBufferSizeFilled
                              );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal, für den die Daten aus der Datei gelesen werden.

lpvBuffer Der Puffer, der die Daten-Meßwerte erhält.

dwBufferSize Die maximale Größe des Puffers, auf den 'lpvBuffer' zeigt.

lpdwBufferSizeFilled Ein Zeiger auf DWORD, der die tatsächliche Anzahl der Bytes erhält, die aus der Datei gelesen werden. Diese Anzahl ist geringer oder genauso groß wie d*wBufferSize*

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE .

Im Falle eines Irrtums bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
BYTE byaDataBuffer[4096];
DWORD dwDataRead;

if (!DiiOpenBlockInputFile      (hDevice,
                                 0,
                                 "c:\\temp\\Channel0Dump.Bin"))
        return;         // Datei kann nicht geöffnet werden

if (DiiReadBlockInputFile       (hDevice,
                                 0,
                                 byaDataBuffer,
                                 4096,
                                 &dwDataRead))
        {       // erfolgreich, Daten können ausgewertet werden      }

DiiCloseBlockInputFile          (hDevice,0);
```

Bemerkungen

DiiCloseBlockInputFile

Diese Funktion wird aufgerufen, um eine vorher geöffnete Block-Input-Datei zu schließen.

Deklaration

```
BOOL DiiCloseBlockInputFile   (       HANDLE hDevice,
                                      DWORD dwChannel,
                              );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde
dwChannel Der Kanal, für den die Datei geschlossen werden soll.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal war außerhalb des Bereiches.

Beispiel

```
BYTE byaDataBuffer[4096];
DWORD dwDataRead;

if (!DiiOpenBlockInputFile      (hDevice,
                                 0,
                                 “c:\\temp\\Channel0Dump.Bin”))
        return;         // Datei kann nicht geöffnet werden

if (DiiReadBlockInputFile       (hDevice,
                                 0,
                                 byaDataBuffer,
                                 4096,
                                 &dwDataRead))
        {       // erfolgreich, Daten können ausgewertet werden      }

DiiCloseBlockInputFile          (hDevice,0);
```

Bemerkungen

DiiPrepareBlockOutput

Mit dieser Funktion ist es möglich, einen einzelnen Kanal für den block-basierten Output vorzubereiten.

Deklaration

```
BOOL DiiPrepareBlockOutput   (      HANDLE hDevice,
                                    DWORD dwChannel,
                                    DWORD dwTriggerSource,
                                    DWORD dwTriggerValue
                             );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Index des Output-Kanals, der automatisch gescannt werden soll (0 ist der erste Kanal).

dwTriggerSource Enthält den Auslösemechanismus für die Messung.

Specify 0 – für einen software-emulierten Auslöser
Specify 1 – für einen Auslöser, der auf einen On-Board-Timer basiert.
Specify 2 – für einen Auslöser, der auf einer externen Quelle basiert, mit der er durch einen digitalen Input-Kanal verbunden ist.

dwTriggerValue Enthält sowohl die Frequenz von Messungen pro Sekunde, falls ein interner Timer benutzt wird oder die Kanalnummer, falls ein externes Signal zur Ausführung der Messung dient.

Rückgabewert

TRUE falls erfolgreich,
Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_SENT, 0))
{
        // Benachrichtigung über geschriebene Blocks kann nicht geliefert werden
        return FALSE;
```

```
}

if (!DiiPrepareBlockOutput (      hDevice,



                                  0,                // mißt den ersten Kanal
                                  0,                // benutzt den internen Timer als Pacer
                                  30000             // versucht auf 30 kHz  zu senden
                  ))
{
        // Der Kanal für den Block-Output kann nicht geöffnet werden.
        return FALSE;
}

if (!DiiPrepareBlockOutput (      hDevice,
                                  1,                // mißt den zweiten Kanal
                                  0,                // benutzt den internen Timer als Pacer
                                  10000             // versucht auf 10 kHz zu senden.
                  ))
{
        // Der Kanal für den Block-Output kann nicht geöffnet werden.
        return FALSE;
}
```

Bemerkungen

Diese Funktion wird aufgerufen, um einen einzelnen Kanal auf den Block-Output vorzubereiten. Die Funktion selbst initiiert keinen Output, aber sie wird benutzt, um die Abtastrate und die Taktquelle für den Block-Output einzurichten.

DiiStartBlockOutput

Diese Funktion wird aufgerufen, um den vorher konfigurierten Block-Output zu starten.

Deklaration

```
BOOL DiiStartBlockOutput    (       HANDLE hDevice,
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE..

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal war außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_SENT, 0))
{
        '' could not request notification about data blocks being written
        return FALSE;
}

if (!DiiPrepareBlockOutput (    hDevice,
                                0,              // mißt den ersten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                30000           // versucht 30 kHz zu messen
                ))
{
        '' Could not initialize the channel for block output.
        return FALSE;
}

if (!DiiPrepareBlockOutput (    hDevice,
                                1,              // mißt den zweiten Kanal
```

```
                                        0,                  // benutzt den internen Timer als Pacer
                                        10000               // versucht 10 kHz zu messen


                        ))
{
        '' Could not initialize the channel for block output.
        return FALSE;
}

if (!DiiStartBlockOutput  (         hDevice))
{
        '' cannot start block output
        return FALSE;
}
```

Bemerkungen

Mit dieser Funktion können Sie die Blockausgabe starten. Die Funktion selbst initiiert zwar keinen Block-Output, aber sie bereitet die Hintergrund-Threats und die angeschlossene Hardware für den Block-Output vor.

DiiEndBlockOutput

Diese Funktion stoppt den block-basierten Output von Datenmessungen.

Deklaration

```
BOOL DiiEndBlockOutput      (       HANDLE hDevice,
                                    );
```

Parameter

HDevice Ein korrekter Geräte-Handle, der durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder er gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_SENT, 0))
{
        '' could not request notification about data blocks being written
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                0,              // mißt den ersten Kanal
                                0,              // benutzt den Timer als Pacer
                                30000           // versucht 30 kHz zu messen
                ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}

if (!DiiPrepareBlockOutput (    hDevice,
                                1,              // mißt den zweiten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                10000           // versucht 10 kHz zu messen
                ))
```

```
{
        '' Could not initialize the channel for block output
        return FALSE;
}

if (!DiiStartBlockOutput  (        hDevice))
{
        '' cannot start block output
        return FALSE;
}

BYTE byaDataBlock[4096];

DiiOutputBlock          (       hDevice,
                                byaDataBlock,
                                0,
                                4096,
                                TRUE); // gibt diesen Datenblock kontinuierlich aus

  //
...// Hier eventuell weiterer Code, z. B. die Bearbeitung von Benachrichtigungen des Gerätes.

if (!DiiEndBlockOutput (        hDevice))
{
        return FALSE;
}
```

Bemerkungen

Benutzen Sie diese Funktion, um eine im Hintergrund arbeitende Datenausgabe zu stoppen.

DiiOutputBlock

Diese Funktion wird aufgerufen, um einen Datenblock auszugeben.

Deklaration

```
BOOL DiiOutputBlock   (      HANDLE hDevice,
                             DWORD dwChannel,
                             LPVOID lpvBuffer,
                             DWORD dwBufferSize,
                             BOOL bContineous
                      );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal auf dem der Block ausgegeben werden soll.

lpvBuffer Der Puffer, der ausgegeben werden soll.

dwBufferSize Die Größe des Puffers, der ausgegeben werden soll.

bContineous Ein Zeichen, das anzeigt, ob die Puffer kontinuierlich ausgegeben werden.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
    void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


    if (!DiiSetNotificationMethodCallBack (hDevice,
                                &ProcessDeviceNotification
                                ))
    {
        return FALSE;
    }

    if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_SENT, 0))
    {
        '' could not request notification about data blocks being written
        return FALSE;
    }

    if (!DiiPrepareBlockInput (     hDevice,
                                    0,              // mißt den ersten Kanl
                                    0,              // benutzt den Timer als Pacer
```

```
                                30000                   // versucht 30 kHz zu messen
                ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}

if (!DiiPrepareBlockOutput (    hDevice,
                                1,                      // mißt den zweiten Kanal
                                0,                      // benutzt den internen Timer als Pacer
                                10000                   // versucht 10 kHz zu messen
                ))
{
        '' Could not initialize the channel for block output
        return FALSE;
}

if (!DiiStartBlockOutput  (     hDevice))
{
        '' cannot start block output
        return FALSE;
}

BYTE byaDataBlock[4096];

DiiOutputBlock          (       hDevice,
                                0,
                                byaDataBlock,
                                4096,
                                TRUE); // gibt diesen Datenblock kontinuierlich aus

  //
...// Hier eventuell weiterer Code, z. B. die Bearbeitung von Benachrichtigungen des Gerätes.

if (!DiiEndBlockOutput (        hDevice))
{
        return FALSE;
}
```

Bemerkungen

Diese Funktion dient dazu, einen Datenblock über einen Kanal auszugeben. Die Schnittstelle zur Blockausgabe muß, wie in dem Beispiel gezeigt, vorbereitet und gestartet werden. Der Block-Output vollzieht sich im Hintergrund, so daß auf den Daten-Puffer, der in die *DiiOutputBlock* - Funktion übergeben wurde, erst dann zugegriffen werden kann, wenn der Datenblock ausgegeben wurde.
Das DII sendet für jeden Datenblock-Output eine DII_DATA_BLOCK_SENT – Benachrichtigung. Durch die lpvBuffer-Parameter des LPDII_NOTIFICATION_DATA –Block kann die Benutzer-Anwendung entscheiden, welche Datenblocks erfolgreich ausgegeben werden.

DiiOutputBlockFromFile

Diese Funktion wird augerufen, um einen Datenblock von einer zuvor erhaltenen Datendatei auszugeben.

Deklaration

```
BOOL DiiOutputBlock   (      HANDLE hDevice,
                             DWORD dwChannel,
                             LPCTSTR lpszFileName,
                       );
```

Parameter

HDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal, auf dem der Block ausgegeben werden soll.

lpszFileName Der Dateiname, der die Datenmessungen enthält, die ausgegeben werden sollen.

Rückgabewert

TRUE falls erfolgreich, anderfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        // Deklaration...
void ProcessDeviceNotification (LPDII_NOTIFICATION_DATA);


if (!DiiSetNotificationMethodCallBack (hDevice,
                                       &ProcessDeviceNotification
                                       ))
{
        return FALSE;
}

if (!DiiRequestNotification (hDevice, DII_DATA_BLOCK_SENT, 0))
{
        '' could not request notification about data blocks being written
        return FALSE;
}

if (!DiiPrepareBlockInput (     hDevice,
                                0,              // mißt den ersten Kanal
                                0,              // benutzt den internen Timer als Pacer
                                30000           // tversucht 30 kHz zu messen
                        ))
{
        '' Could not initialize the channel for block input.
        return FALSE;
}
```

```
if (!DiiPrepareBlockOutput (      hDevice,
                                  1,                    // mißt den zweiten Kanal
                                  0,                    // benutzt den internen Timer als Pacer
                                  10000                 // versucht 10 kHz zu messen
                  ))
{
        '' Could not initialize the channel for block output
        return FALSE;
}

if (!DiiStartBlockOutput  (       hDevice))
{
        '' cannot start block output
        return FALSE;
}

DiiOutputBlockFromFile(           hDevice,
                                  0,
                                  "c:\\temp\\Channel0Dump.bin",
                                  TRUE); // gibt diesen Datenblock kontinuierlich aus

  //
...// Hier eventuell weiterer Code, z. B. die Bearbeitung von Benachrichtigungen des Gerätes.

if (!DiiEndBlockOutput (          hDevice))
{
        return FALSE;
}
```

Bemerkungen

Mit dieser Funktion können Sie einen Datenblock über einen Kanal ausgeben. Die Schnittstelle zur Blockausgabe muß, wie in dem Beispiel gezeigt, vorher vorbereitet und gestartet werden. Die Daten, die ausgegeben werden sollen, werden von der vorhandenen Datei gelesen. Diese Datei muß zuvor durch die Schnittstelle zur Blockausgabe erstellt werden. (Weiteres dazu unter *DiiPrepareBlockInput*).

Das DII sendet für jeden Datenblock-Output eine DII_DATA_BLOCK_SENT –Benachrichtigung zu der Anwendung.

DiiGetBlockOutputStatus

Diese Funktion bringt Status-Informationen über einen Block-Output für einen bestimmten Kanal zurück.

Deklaration

```
BOOL DIIEXPORT DiiGetBlockOutputStatus    (
                                          HANDLE hDevice,
                                          DWORD dwChannel,
                                          LPDWORD lpdwTotalSamples,
                                          LPDWORD lpdwSamplesPerSecond,
                                          LPDWORD lpdwBufferUnderruns
                                    );
```

Parameter

hDevice Ein korrekter Geräte-Handle, der vorher durch DiiOpenDevice oder DiiOpenNamedDevice erhalten wurde.

dwChannel Der Kanal, für den die Informationen zurückgeholt werden sollen.

lpdwTotalSamples
Die vollständige Anzahl von den schon gesendeten Daten-Meßungen (*nicht bytes!*).

lpdwSamplesPerSecond
Die durchschnittliche Anzahl von Daten-Meßungen pro Sekunde.

lpdwBufferUnderruns
Die Anzahl der bisherigen Puffer-Unterläufe. Puffer-Unterläufe treten auf, wenn die Hardware Daten schneller sendet als das DII sie verarbeiten kann. Das ist ein Fehler.

Rückgabewert

TRUE falls erfolgreich, andernfalls FALSE.

Im Falle eines Fehlers bringt GetLastError() folgende Werte zurück:

ERROR_INVALID_PARAMETER - Der übergebene Handle war inkorrekt oder der gewählte Kanal liegt außerhalb des Bereiches.

Beispiel

```
        //
        // Lesen Sie dazu das Beispiel, wie Sie den Block-Output einrichten und starten können.

    DWORD dwTotalSamples;
    DWORD dwSamplesPerSecond;
    DWORD dwBufferUnderruns;

    if (DiiGetBlockOutputStatus (       hDevice,
                                        0,          // bringt das Info für den ersten Kanal
                                        zurück
                                        &dwTotalSamples,
                                        &dwSamplesPerSecond,
                                        &dwBufferUnderruns
                            ))
    {
        // evaluiert die statistischen Informationen
```

```
}
```

Bemerkungen

Diese Funktion bringt statistische Informationen während eines laufenden Block-Outputs zurück. Die Benutzer-Anwendung achtet selbsttätig darauf, daß alle möglichen Daten korrekt gesendet werden. Ihr gelingt dieses durch die Auswertung der Fehler, die Pufferunterläufe verursachen.
Ein Pufferunterlauf tritt auf, wenn die Hardware Daten schneller sendet als das DII sie ihrerseits zu der Hardware liefern kann.

9. Das Dynamic Industrial Interface OCX/ActiveX Control

Die DII-OCX wird mit den Dynamic Industrial Interface-Treibern mitgeliefert und ist für den Gebrauch vieler Anwendungen, wie z. B. Visual Basic oder Delphi registriert. In vielen Fällen ist eine Anwendungsentwicklung, die eine DII-OCX gebraucht, schneller und effizienter, da viele technische Probleme, wie z. B. die Umkehrung von Parametern, von Programmierern nicht extra gelöst werden müssen. Die OCX bietet die gleichen Leistungen wie die DII-DLL. Wenn es Ihnen angenehmer ist, können Sie deshalb die DLL in Ihrer C/C++ Anwendung weiter nutzen.

Mit der OLE-Automation stellt die OCX Eigenschaften und Methoden zur Verfügung, die leicht verändert oder von Hochsprachen aufgerufen werden können. Die folgenden Seiten enthalten eine detaillierte Beschreibung von jeder Methode und Eigenschaft, die die OCX bietet.

9.1. Eigenschaften

Die folgenden Eigenschaften werden von der OCX-Control zur Verfügung gestellt:

Eigenschaft

DeviceName

Typ

String

Persistent

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

End If
```

Bemerkungen

Diese Eigenschaft enthält den Gerätenamen, der von einer Instanz der OCX geöffnet werden soll. Sobald diese Eigenschaft verändert wird, schließt die OCX das arbeitende Gerät und versucht, das Gerät mit dem neuen Namen zu öffnen.

Sie können auch die OCX.-Methode "SelectDevice" anwenden, um sich eine Dialog-Box anzeigen zu lassen, in der ein Benutzer den Gerätenamen aussuchen kann, mit dem er weiterhin arbeiten möchte.

Um festzustellen, ob ein Gerätenamen zu einem installierten Gerät in Ihrer Maschine paßt und ob es für den weiteren Zugriff bereit ist, müssen Sie die Eigenschaft "DeviceOpened" benutzen. Wenn Sie das Gerät erfolgreich geöffnet haben, erscheint TRUE, ansonsten FALSE.

Eigenschaft

Exclusive

Typ

Boolean

Persistent

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX Control enthält, "DiiDevice genannt".
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

DiiDevice.Exclusive = True

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

End If
```

Bemerkungen

Diese Eigenschaft bestimmt wie Sie einen Adapter öffnen, nämlich entweder exklusiv oder nicht-exklusiv. Wenn Sie diese Eigenschaft auf TRUE einstellen, hindert das andere Anwendungen oder andere OCXs daran, ein Gerät mit dem gleichen Namen zu öffnen.

Eigenschaft

DeviceOpened

Typ

Boolean

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX-Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

End If
```

Bemerkungen

Jedesmal wenn die OCX erfolgreich ein Gerät geöffnet hat, wird diese Eigenschaft auf TRUE eingestellt, andernfalls auf FALSE.

Eigenschaft

DigitalChannels

Typ

long

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX-Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DiiDevice.DigitalChannels = 0 Then

        MsgBox "This device cannot do digital I/O"

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummern von digitalen Kanälen des geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Bitte beachten Sie: Um zwischen Input- und Output-Kanälen unterscheiden zu können, stehen Ihnen auch die Eigenschaften *DigitalInputChannels* und *DigitalOutputChannels* nutzen zur Verfügung.

Eigenschaft

DigitalInputChannels

Typ

long

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX-Control enthält,  "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DiiDevice.DigitalInputChannels = 0 Then

        MsgBox "This device does not have digital input lines"

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummern der digitalen Input-Kanäle des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Eigenschaft

DigitalOutputChannels

Typ

long

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII-OCX-Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DiiDevice.DigitalOutputChannels = 0 Then

        MsgBox "This device does not have digital output lines"

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummer der digitalen Output-Kanäle des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigeschaft "DeviceName" spezifiziert ist.

Eigenschaft

AnalogChannels

Typ

long

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
‘’ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.”

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

    MsgBox “Unable To Open Device named Device0”

Else

    If DiiDevice.AnalogChannels = 0 Then

        MsgBox “This device cannot do analog I/O”

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummer der analogen Kanäle des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Bitte beachten Sie: Um zwischen Input- und Output-Kanälen unterscheiden zu können, stehen Ihnen auch die Eigenschaften *AnalogInputChannels* und *AnalogOutputChannels* zur Verfügung.

Eigenschaft

AnalogInputChannels

Typ

long

Read-Only

Parameter

Keine

Beispiel (Visual Basic)

```
‘’ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.”

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

    MsgBox “Unable To Open Device named Device0”

Else

    If DiiDevice.AnalogInputChannels = 0 Then

        MsgBox “This device does not have analog input channels”

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummer der analogen Input-Kanäle des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Eigenschaft

AnalogOutputChannels

Typ

long

Read-Only

Parameter

Keine

Muster (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

    MsgBox "Unable To Open Device named Device0"

Else

    If DiiDevice.AnalogOutputChannels = 0 Then

        MsgBox "This device does not have analog output channels"

    End If

End If
```

Bemerkungen

Diese Eigenschaft enthält die Nummer der analogen Output-Kanäle des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Eigenschaft

Resolution

Typ

long

Read-Only

Parameter

Keine

Muster (Visual Basic)

```
'' Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, "DiiDevice" genannt."
'' Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln."

DiiDevice.DeviceName = "Device0"

If Not DiiDevice.DeviceOpened Then

        MsgBox "Unable To Open Device named Device0"

Else

        MaximumResolution = DiiDevice.Resolution

End If
```

Bemerkungen

Diese Eigenschaft enthält in Bits die Auflösung des aktuell geöffneten Industrial I/O-Gerätes, wie es in der Eigenschaft "DeviceName" spezifiziert ist.

Eigenschaft

ChannelGain

Typ

double

Parameter

IsInputChannel (Boolean)

ChannelIndex (Long)

Muster (Visual Basic)

```
‘’ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.”

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

        MsgBox “Unable To Open Device named Device0”

Else

        Dim CurrentGain As Double
        CurrentGain = DiiDevice.ChannelGain (True,0)
                ‘’ Reads the channel gain from the first channel

        DiiDevice.ChannelGain(True,0) = 10.0
                ‘’ Sets the amplification for the first channel to 10.

End If
```

Bemerkungen

Diese Eigenschaft setzt die Vorverstärkung des Kanals oder gibt sie zurück, und zwar so, wie sie für einen bestimmten analogen Kanal konfiguriert ist. Für weitere Informationen lesen Sie bitte die Beschreibung der Funktionen DiiSetChannelGain und DiiGetChannelGain.

Eigenschaft

ChannelBipolar

Typ

double

Parameter

IsInputChannel (Boolean)

ChannelIndex (Long)

Muster (Visual Basic)

```
‘‘ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.”

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

        MsgBox “Unable To Open Device named Device0”

Else

        Dim IsChannelBipolar As Boolean
        IsChannelBipolar = DiiDevice.ChannelBipolar (True,0)
                ‘’ Reads the bipolar mode from the first channel

End If
```

Bemerkungen

Diese Eigenschaft setzt den bipolaren Modus, der für einen bestimmten analogen Kanal konfiguriert ist oder bringt ihn zurück. Für weitere Informationen lesen Sie bitte die Beschreibung der Funktionen DiiSetChannelBipolar and DiiGetChannelBipolar.

Eigenschaft

RealAnalogChannel

Typ

double

Parameter

ChannelIndex (Long)

Muster (Visual Basic)

```
‘‘ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

        MsgBox “Unable To Open Device named Device0”

Else

        Dim CurrentValue As Double

        CurrentValue = DiiDevice.RealAnalogChannel(0)
                ‘’ Reads the current value of the first analog channel

        DiiDevice.RealAnalogChannel(1) = 3.5
                ‘’ Sets the second analog output channel to 3.5 Volts.

End If
```

Bemerkungen

Diese Eigenschaft liest Echtwerte von Input-Kanälen bzw. schreibt Echtwerte auf Output-Kanäle. Falls diese Funktion fehlschlägt wird eine Ausnahme (OLE-Exception) ausgelöst. Für weitere Informationen gehen Sie bitte zu den Dii-Funkionen DiiSetRealAnalogChannel und DiiGetRealAnalogChannel.

Eigenschaft

NotificationMask

Typ

long

Parameter

Keine

Muster (Visual Basic)

```
‘‘ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

        MsgBox “Unable To Open Device named Device0”

Else

        DiiDevice.NotificationMask = &H2
                ‘’ Enables Signal Change Notifications

                ‘’ Check, if notifications were activated:
        If Not DiiDevice.NotificationsActive Then
                MsgBox “This device does not support notifications”
        End If

                ‘’ Disable notifications again.
        DiiDevice.NotificationMask = 0

End If
```

Bemerkungen

Sie werden die angeforderten Benachrichtigungen erhalten, wenn Sie diese Eigenschaft auf einen Wert setzen, der nicht Null ist. Dieser Vorgang ist gleich dem Aufruf “RequestNotification”. Wenn NotificationMask auf Null gestellt wird, werden die Benachrichtigungen gestoppt. Dieser Vorgang ist gleich dem Aufruf “CancelNotification”.

Eigenschaft

NotificationsActive

Typ

long

Read-Only

Parameter

Keine

Muster (Visual Basic)

```
‘‘ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den folgenden Kapiteln.”

DiiDevice.DeviceName = “Device0”

If Not DiiDevice.DeviceOpened Then

        MsgBox “Unable To Open Device named Device0”

Else

        DiiDevice.NotificationMask = &H2
                ‘’ Enables Signal Change Notifications

                ‘’ Check, if notifications were activated:
        If Not DiiDevice.NotificationsActive Then
                MsgBox “This device does not support notifications”
        End If

                ‘’ Disable notifications again.
        DiiDevice.NotificationMask = 0

End If
```

Bemerkungen

Diese Eigenschaft zeigt an, ob eine Benachrichtigung erfolgreich gesendet oder aber gestoppt wurde. Wenn die OCX auf True gesetzt wird, kann sie “Notify”-Ereignisse zu jeder Zeit zur Verfügung stellen. Wenn die OCX auf False gesetzt wird, kann sie “Notify”-Ereignisse nicht zur Verfügung stellen.

9.2. Methoden

Die Methoden, die von der OCX-Control zur Verfügung gestellt werden, entsprechen den jeweiligen DII-Funktionen des Dynamic Industrial Interface, wie es in den vorhergehenden Kapiteln dargestellt wurde. Bezüglich der folgenden Methoden lesen Sie bitte die entsprechende Beschreibung in dem Funktionen-Überblick der vorhergehenden Kapitel.

SetDigitalBit	->	DiiSetDigitalBit
GetDigitalBit	->	DiiGetDigitalBit
SetDigitalByte	->	DiiSetDigitalByte
GetDigitalByte	->	DiiGetDigitalByte
Set8255Config	->	DiiSet8255Config
SetAnalogChannel	->	DiiSetAnalogChannel
GetAnalogChannel	->	DiiGetAnalogChannel
SetRealAnalogChannel	->	DiiSetRealAnalogChannel
GetRealAnalogChannel	->	DiiGetRealAnalogChannel
SetChannelRange	->	DiiSetChannelRange
GetChannelRange	->	DiiGetChannelRange
GetDeviceChannelRange	->	DiiGetDeviceChannelRange
SetTimerConfig	->	DiiSetTimerConfig
LoadTimer	->	DiiLoadTimer

(etc)

Benutzen Sie die beigefügten Beispiel-Anwendungen, um Beispiele für den Gebrauch dieser Funktionen einsehen zu können.

9.3. Ereignisse

In der aktuellen Version unterstützt die DII-OCX das Senden von Ereignissen für interrupt-getriebene Karten. Diese Ereignisse korrespondieren mit Benachrichtigungen von der DII-DLL. Benachrichtigungen können immer dann gesendet werden, wenn sich ein Signal auf der Karte ändert oder wenn ein Zähler auf Null zurückfällt.

Die DII-OCX stellt mit “Notify” ein zentrales Ereignis zur Verfügung. Wenn ein “Notify”-Ereignis gesendet wird, sind folgende Parameter verfügbar:

```
Notify(ByVal NotificationMask As Long, _
       ByVal NotificationParam As Long, _
       ByVal DataBuffer As Variant, _
       ByVal BufferSize As Long)
```

Parameter

NotificationMask
Eine Bit-Maske, die die gesendeten Benachrichtigungen enthält. In dieser Maske korrespondieren ein oder mehrere Bits mit der vorher eingestellten Eigenschaft “NotificationMask”.

NotificationParam
Enthält die Parameter für die Benachrichtigung. Die Interpretation dieser Parameter hängt von der gesendeten Benachrichtigung ab. Für weitere Informationen sehen Sie bitte die DII_NOTIFICATION_DATA ein.

DataBuffer Eine Variante, die einen binären Puffer von Daten enthält, die das Ereignis begleiten. Sie wird für das Hintergrund-Messung von Daten genutzt.

BufferSize Die Größe der Puffer in Bytes. (Für Visual Basic nicht gebraucht).

Beispiel (Visual Basic)

```
‘‘‘ Das Beispiel geht von einer Form aus, die ein DII_OCX-Control enthält, “DiiDevice” genannt.”
‘’ Weitere Informationen zum Gebrauch der OCX von Visual Basic aus entnehmen Sie bitte den
folgenden Kapiteln.”

Private Sub Form_Load()

    DiiDevice.DeviceName = “Device0”

    If Not DiiDevice.DeviceOpened Then
        MsgBox “Unable To Open Device named Device0”
    Else
        DiiDevice.NotificationMask = &H2
                ‘’ Enables Signal Change Notifications

                ‘’ Check, if notifications were activated:
        If Not DiiDevice.NotificationsActive Then
            MsgBox “This device does not support notifications”
        End If
    End If
End Sub

Private Sub Form_Unload()

        ‘’ Disable notifications again.
```

```
        DiiDevice.NotificationMask = 0
End Sub

Private Sub DiiDevice_Notify(ByVal NotificationMask As Long, _
                             ByVal NotificationParam As Long, _
                             ByVal DataBuffer As Variant, _
                             ByVal BufferSize As Long)
                '
                '' Diese Funktion wird immer dann aufgerufen, wenn das Gerät eine
                '' Benachrichtigung sendet

                '' Prüfen Sie den Typ der Benachrichtigung:

        If ((NotificationMask And &H2) <> 0) Then

                If ((NotificationParam And &H1) <> 0) Then
                        '' Signal 1 changed.
                End If

                If ((NotificationParam And &H2) <> 0) Then
                        '' Signal 2 changed.
                End If

                If ((NotificationParam And &H4) <> 0) Then
                        '' Signal 3 changed.
                End If

                If ((NotificationParam And &H8) <> 0) Then
                        '' Signal 4 changed.
                End If

                        '' ...
        End If

        If ((NotificationMask And &H20) <> 0) Then
                '
                '' Eine Benachrichtigung, daß Rückwärtszähler
                Beep
        End If
End Sub
```

10. Der Gebrauch des Dynamic Industrial Interface mit anderen Programmier-Sprachen

Dieses Kapitel zeigt Ihnen, wie Sie das Dynamic Industrial Interface in den verschiedenen Programmier-Sprachen am besten gebrauchen können.

Wenn der Aufruf von Funktionen des Dynamic Industrial Interface Schwierigkeiten macht oder wenn Sie eine Programmier-Sprache verwenden, die nicht in diesem Handbuch berücksichtigt wurde, dann besuchen Sie doch unsere Web-Seite http://www.nt-ware.com an. Dort stellen wir Ihnen die neuesten Informationen über alle Fragen zur DII-Programmierung zur Verfügung. Über unsere Web-Seite können Sie auch unsere technische Unterstützung in Anspruch nehmen.

10.1. C++

Weil bei der Entwicklung des Dynamic Industrial Interface DLL auch C++ benutzt wurde, können Sie mit dieser Programmier-Sprache leicht auf Industrial I/O-Geräte zugreifen.
Zu diesem Zweck werden sowohl eine C++ Header-Datei ("Dii.h") wie auch eine Import Library ("Dii.lib") mitgeliefert. Vergewissern Sie sich, daß Sie die Entwickler-Version und nicht die Endkunden-Version installiert haben, da letztere nicht die Entwicklerdateien beinhaltet.

Bitte beziehen Sie die “Dii.h” Include Datei in Ihre C/C++ Quellcode-Dateien ein. Jetzt können Sie jede der Funktionen nutzen, die von dem Dynamic Industrial Interface DLL. angeboten werden. Vergewissern Sie sich, daß Sie die Import Library "Dii.lib" mit einbeziehen, während Sie das Programm linken. Auf diese Weise stimmen Sie Ihre Anwendungen erfolgreich auf die aktuelle DLL-Schnittstelle ab.

10.2. Visual Basic

Bitte beachten Sie: Weil das Dynamic Industrial Interface vollständig 32-Bit-fähig ist, werden nur 32-Bit-Versionen von Visual Basic unterstützt. Die Versionen 4.0 und Version 5.0 wurden getestet und werden unterstützt .

Wenn Sie Visual Basic für den Zugriff auf I/O-Geräte verwenden wollen, die von dem Dynamic Industrial Interface unterstützt werden, dann haben Sie grundsätzlich 2 Möglichkeiten:
Sie können entweder die Dii-DLL direkt aufrufen oder die OCX gebrauchen. Wegen des leichteren Zugriffs auf die unterstützten Meßdaten-Funktionen empfehlen wir den Weg mit der OCX.
Um das OCX-Control zu nutzen, versichern Sie sich, daß Sie das Dynamic Industrial Interface korrekt auf Ihrem Computer installiert haben. Dann fügen Sie das OCX-Control Ihrer Visual Basic-Tollbar hinzu, indem Sie aus dem Menü "Projects / Component..." auswählen (Visual Basic V5.0).

Um auf ein bestimmtes Gerät zugreifen zu können, ziehen Sie bitte das DII OCX-Control von Ihrer Visual Basic-Toolbar zu Ihrer Form. Wählen Sie jetzt den Gerätenamen aus den Eigenschaften aus. Das Programmieren von DII-unterstützten Karten vollzieht sich dann genauso wie das Programmieren jeder beliebigen anderen OCX. Genauere Informationen über OCX-Eigenschaften und –Methoden finden Sie in Kapitel 9.
Wenn Sie Genaueres darüber wissen möchten, wie Sie mit Visual Basic auf das Dynamic Industrial Interface zugreifen können, nutzen Sie bitte die Visual Basic-Beispiel-Anwendung. Sie finden sie in dem “DiiDemoVB” Unterverzeichnis Ihres Installationsverzeichnisses.

10.3. Borland Delphi

Bitte beachten Sie: Weil das Dynamic Industrial Interface 32-Bit-fähig ist, werden auch nur 32-Bit-Versionen von Delphi unterstützt. Insbesondere wurde die Version 2.0 getestet und wird unterstützt.

Das Dynamic Industrial Interface kann leicht mit der Programmiersprache Delphi genutzt werden. Setzen Sie dazu bitte das beigefügte OCX/ActiveX Conrol ein. Dafür müssen Sie zunächst das OCX-Control Ihrem Projekt hinzufügen, indem Sie “Component/Import ActiveX Control...” auswählen. Geben Sie dann an, daß Sie eine OCX-Komponente installieren möchten. Delphi schafft für Sie eine Unit(.pas), die eine Klasse, genannt “TDiiOcxCtrl” exportiert. Diese neue Klasse beinhaltet die DII OCX und Sie können damit auf jede Eigenschaft der OCX zugreifen.

Dann ziehen Sie bitte das OCX-Control zu Ihrer Form und bearbeiten graphisch alle Eigenschaften.

Wenn Sie Genaueres darüber wissen möchten, wie Sie mit Delphi auf das Dynamic Industrial Interface zugreifen können, nutzen Sie bitte die Delphi Beispiel-Anwendung. Sie finden sie in dem “DiiDemoDelphi” Unterverzeichnis Ihres Installations-Verzeichnisses.

11. Technische Unterstützung und Feedback

Ihre Rückmeldung ist für uns die wertvollste Quelle, um auch in Zukunft erfolgreiche, benutzerfreundliche Produkte zu entwickeln.

Wenn Sie Unterstützung beim Gebrauch der DII wünschen oder wenn Sie Verbesserungs-Vorschläge haben, nehmen Sie bitte Kontakt mit uns auf. Entweder über Email: support@nt-ware.com, oder über unsere Web-Seite http://www.nt-ware.com.

Wir arbeiten ständig daran, um das Dynamic Interface zu erweitern und zu verbessern. Wir möchten Ihnen hilfreiche Produkte rund um das DII zur Verfügung stellen. Wir planen, Ihnen neue Funktionalitäten und den Einsatz von speziellen Geräten und speziellen Anwendungen vorzustellen. Dabei stehen für uns immer Ihre Bedürfnisse im Mittelpunkt.