

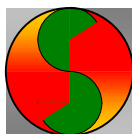
# I332.DLL

# HANDBUCH

ab Version 5.0

Stand 17.09.2009

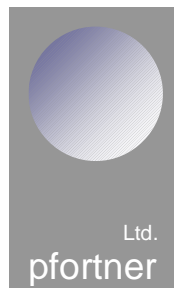
System-Software-Entwicklung  
Dipl.Inform.  
Thomas Schneider



Im Schieb 22 40668 Meerbusch

Tel. 02150 911 747  
Fax 02150 911 748

eMail: [thomas.schneider@sys-thos.de](mailto:thomas.schneider@sys-thos.de)



Fa. Pfortner Ltd.

Krautstückerweg 13  
76706 Dettenheim

Tel. +49 7247 906060  
Fax +49 7247 906063

[wopfo@t-online.de](mailto:wopfo@t-online.de)  
<http://www.pfortner.de>

# Inhaltsverzeichnis

## INHALTSVERZEICHNIS.....I

### 1 ÜBERSICHT..... 1-1

1.1	Dateien.....	1-2
1.1.1	Win9x_ME.....	1-2
1.1.2	WinNT.....	1-2
1.1.3	Win2k.....	1-2
1.1.4	Verzeichnis Dok.....	1-2
1.1.5	Verzeichnis Demo.....	1-2
1.2	Hinweise zur Installation.....	1-3
1.2.1	Windows 95/98/ME.....	1-3
1.2.2	Windows NT.....	1-3
1.2.3	Windows 2000.....	1-3
1.2.4	Windows XP.....	1-4
1.3	Hinweise für Programmierer.....	1-5
1.3.1	Allgemeine Hinweise.....	1-5
1.3.2	Hinweise für C/C++ - Programmierer.....	1-5
1.3.3	Hinweise für Visual-Basic – Programmierer.....	1-6

### 2 KONSTANTEN UND TYPEN..... 2-1

2.1	Konstanten.....	2-2
2.2	Typen.....	2-3

### 3 FUNKTIONEN..... 3-1

3.1	Verwaltung und Kommunikation.....	3-2
3.1.1	I332IFCInit.....	3-2
3.1.2	I332IFCDone.....	3-2
3.1.3	I332Reset.....	3-3
3.1.4	I332Restart.....	3-3
3.1.5	I332Write0.....	3-3
3.1.6	I332Write1.....	3-3
3.1.7	I332Write2.....	3-4
3.1.8	I332Read1.....	3-4
3.1.9	I332Read2.....	3-4
3.1.10	I332DLLVersion.....	3-4
3.1.11	I332GetIFCErr.....	3-4
3.1.12	I332GetInfo.....	3-5
3.1.13	I332GetVerStr.....	3-5
3.1.14	I332MutexName.....	3-5
3.1.15	I332WaitForMutex.....	3-5
3.1.16	I332ReleaseMutex.....	3-6
3.1.17	I332SetAchsReg.....	3-6
3.1.18	I332GetAchsReg.....	3-6
3.1.19	I332AchsReg.....	3-6
3.2	Konfiguration.....	3-8
3.2.1	I332XmtCFG.....	3-8
3.2.2	I332ExecCfgDlg.....	3-8
3.2.3	I332InitMP.....	3-8
3.3	Statusfunktionen.....	3-10
3.3.1	I332GetStatus.....	3-10
3.3.2	I332ZyklusStart.....	3-10
3.3.3	I332ZyklusStop.....	3-10
3.3.4	I332DoZyklusStart.....	3-10
3.3.5	I332GetRefInfo.....	3-11

## **Inhaltsverzeichnis**

3.3.6	I332GetPos .....	3-11
3.3.7	I332Pos .....	3-11
3.3.8	I332GetVlst .....	3-11
3.3.9	I332Vlst .....	3-12
3.3.10	I332GetAxArray .....	3-12
3.4	Sonderfunktionen .....	3-13
3.4.1	I332DoAxisRef .....	3-13
3.4.2	I332DoAxisAsync .....	3-13
3.4.3	I332DoAxisHand .....	3-14
3.5	Interpolation .....	3-15
3.5.1	I332DoPos1 .....	3-15
3.5.2	I332DoIPLLin .....	3-15
3.5.3	I332DoIPLCW .....	3-16
3.5.4	I332DoIPLCCW .....	3-16
3.5.5	I332DoDelay .....	3-17
3.6	USB- / Blockübertragung .....	3-18
3.6.1	Allgemeines .....	3-18
3.6.2	I332XmtBuffer .....	3-19
3.6.3	I332InitBuffer .....	3-19
3.6.4	I332FreeBuffer .....	3-20
3.6.5	I332FlushBuffer .....	3-20
3.6.6	I332ClearBuffer .....	3-20
3.6.7	I332GetBuffer .....	3-21
3.6.8	Arbeiten mit Buffern .....	3-21
<b>4</b>	<b>DEMO-PROGRAMM I332DEMO.EXE .....</b>	<b>4-1</b>
4.1	Die Demo-Funktionen .....	4-4
4.2	Referenzfahrt – Demo .....	4-6
4.3	Handbetrieb – Demo .....	4-8
4.4	Interpolation – Demo .....	4-10
4.5	Spindel + Verweilzeit – Demo .....	4-12
4.6	Asynchron fahren – Demo .....	4-13
4.7	Joystick – Demo .....	4-14
4.8	Handrad – Demo .....	4-15
4.9	I332-Buffer – Demo .....	4-16

# 1 Übersicht

Dieses Dokument beschreibt die I332.DLL zur Kommunikation mit der Motorsteuer-Karte I332 der Fa. Pfortner GmbH.

Die DLL ist lauffähig unter Windows 95/98/ME und Windows NT/2000/XP (mit Treiber gwiopm.sys).

Die DLL unterstützt sowohl die ISA-, die PCI und die USB-Version der I332 als auch die Kommunikation über die serielle Schnittstelle. Insbesondere werden auch mehrere I332 im System unterstützt (z.B. 2 PCI-Karten oder auch die Kombination PCI / ISA-Karten).

**Hinweis:** *Bisher wird nur eine I332-USB im System unterstützt.*

Die folgende Beschreibung der in der DLL enthaltenen Funktionen orientiert sich an der Syntax der Sprache Delphi/PASCAL; es sollte jedoch keine Probleme bereiten, die DLL auch aus anderen Sprachen anzusprechen. Das Aufrufmodell ist, wie von Microsoft empfohlen, „stdcall“.

Bis auf einige Ausnahmen sind alle Routinen der I332.DLL Funktionen und liefern den (negativen) Fehler-Code der Schnittstelle bzw. ein (positives) Funktionsergebnis zurück; Details s. bei der Beschreibung der jeweiligen Funktion.

Die DLL ist thread-sicher, d.h. es können mehrere Threads parallel auf die Schnittstelle zugreifen. Die Absicherung erfolgt intern über ein Mutex, eine explizite Absicherung durch die Anwendung ist nicht erforderlich; bei Bedarf kann eine Anwendung / ein Thread jedoch die Schnittstelle auch exklusiv belegen (s.u. I332WaitForMutex bzw. I332ReleaseMutex).

Einige Besonderheiten für die USB-Version sind im Kap. 3.6 erläutert.

# 1.1 Dateien

## 1.1.1 Win9x\_ME

### PCI-Version:

Unter Windows 9x/ME sind keine Treiber notwendig. Unter Windows 95/98 ist die Erweiterung DCOM98 notwendig (s. <http://www.microsoft.com>).

**Hinweis:** Ein Indiz für die Installation dieser Erweiterung ist das Vorhandensein der Datei OLEAUT32.DLL im Windows-Systemverzeichnis.

### Unterverzeichnis USB:

- EZUSB.SYS Treiber
- EZUSBW2K.INF Installations-Skript für den Treiber

## 1.1.2 WinNT

### Unterverzeichnis PCI:

- GWIOPM.SYS Treiber
- INST\_332.EXE Installations-Programm für den Treiber

## 1.1.3 Win2k

### Unterverzeichnis PCI:

- GWIOPM.SYS Treiber
- GWIOPM.INF Installations-Skript für den Treiber
- INST\_332.EXE Installations-Programm für den Treiber (zur manuellen Installation der ISA-Karte)

### Unterverzeichnis USB:

- EZUSB.SYS Treiber
- EZUSBW2K.INF Installations-Skript für den Treiber

## 1.1.4 Verzeichnis Dok

- I332\_DLL.PDF Diese Datei

## 1.1.5 Verzeichnis Demo

- I332.DLL DLL zur Kommunikation mit der I332
- I332DEMO.EXE DEMO-Programm
- I332.INI I332-Konfigurationsdaten für das Demo-Programm

Source-Code des Demoprogramms; Details s. Kap. 4, „Demo-Programm I332Demo.EXE“.

# 1.2 Hinweise zur Installation

## 1.2.1 Windows 95/98/ME

PCI-Version: Unter diesen Windows-Versionen sind für die ISA- und PCI-Version keine Treiber erforderlich. Die DLL kann direkt verwendet werden. Unter Windows 95/98 ist jedoch die Erweiterung DCOM98 notwendig (s. <http://www.microsoft.com>).

**Hinweis:** *Ein Indiz für die Installation dieser Erweiterung ist das Vorhandensein der Datei OLEAUT32.DLL im Windows-Systemverzeichnis.*

Für die PCI-Version der I332 können die Abfragen des Hardware-Managers bei der Installation jeweils mit ok bestätigt werden.

USB-Version: Ein neues USB-Gerät wird automatisch erkannt; der Treiber **EZUSB.SYS** bzw. die zugehörige Datei **EZUSBW2K.INF** ist im Unterverzeichnis Verzeichnis **USB** zu finden.

## 1.2.2 Windows NT

Für die ISA- und PCI-Version ist der Treiber **GWIOPM.SYS** notwendig.

Unter einem Admin-Account kann das Programm I332Demo.EXE bzw. die I332.DLL ohne explizite Installation genutzt werden, wenn sich dieser Treiber im gleichen Verzeichnis wie die Applikation befindet.

Ansonsten muß der Treiber unter einem Admin-Account mit Hilfe des Programms **Inst\_332.EXE** zunächst installiert werden und steht dann für andere Benutzer zur Verfügung. Installationsprogramm und Treiber stehen im Verzeichnis **WinNT** zur Verfügung.

USB-Version: icht verfügbar

## 1.2.3 Windows 2000

PCI-Version: PCI-Karten werden wie unter Win9x/ME vom Hardware-Manager erkannt; der Treiber **GWIOPM.SYS** bzw. die zugehörige Datei **GWIOPM.INF** ist im Unterverzeichnis Verzeichnis **PCI** zu finden.

USB-Version: Ein neues USB-Gerät wird automatisch erkannt; der Treiber **EZUSB.SYS** bzw. die zugehörige Datei **EZUSBW2K.INF** ist im Unterverzeichnis Verzeichnis **USB** zu finden.

Die Treiber-Installation läuft in der Regel folgendermaßen ab:

- Eingabefenster des Hardware-Assistenten
  - *Eingaben des Benutzers*
- Win2000 erkennt die I332-Karte (PCI/USB) und startet den Hardware-Assistenten
  - weiter
- "Passenden Treiber für das Gerät suchen?"
  - weiter
- "Andere Quellen für die Suche"
  - Auswahl: "Andere Quelle angeben"
  - weiter
- Es erscheint ein Eingabe-Dialog, mit dem auch der Rechner nach dem passenden Verzeichnis durchsucht werden kann ("Durchsuchen"):
  - Verzeichnis **Win2k**, Unterverzeichnis **PCI** bzw. **USB** auswählen
  - Treiber **GWIOPM.INF** bzw. **EZUSBW2K.INF** auswählen
  - OK
- "Für folgendes Gerät wurde ein Treiber gefunden: <devicename>"
  - weiter
- "Fertigstellen des Assistenten: <devicename>"

## Übersicht

- fertigstellen

Installationsskripte und Treiber stehen im Verzeichnis **Win2k\_XP** zur Verfügung.

ISA-Version: Da die ISA-Version der I332-Karte nicht Plug-and-Play-fähig ist, Win2000 aber auf dieser Technologie aufsetzt, muß der Treiber für die I332-ISA-Version explizit installiert werden. Dies läuft genauso ab, wie unter Windows NT (s. Kap. 1.2.2).

### 1.2.4 Windows XP

Die Installation verläuft im wesentlichen wie unter Windows 2000; der Hinweis, daß Treiber nicht von Microsoft zertifiziert sind, kann ignoriert werden.

# 1.3 Hinweise für Programmierer

## 1.3.1 Allgemeine Hinweise

- Die DLL wurde in Delphi erstellt, die folgende Beschreibung der Schnittstelle verwendet deshalb die Delphi-Syntax.
- Eine DLL kann prinzipiell unabhängig von der Programmiersprache von jeder Anwendung verwendet werden, sofern die Programmiersprache/-umgebung das Einbinden von DLLs unterstützt. Notfalls kann dazu auf entsprechende Windows-API-Routinen zurückgegriffen werden (s. Hinweise für C/C++ - Programmierer).

- Verwendete Integer-Typen:

INTEGER unter Win32: C/C++:	32 Bit mit Vorzeichen int
Byte C/C++:	1 Byte ohne Vorzeichen (0..255) int
ShortInt C/C++:	1 Byte mit Vorzeichen (-128..+127) signed char
LONGINT C/C++:	32 Bit mit Vorzeichen long
WORD C/C++:	16 Bit ohne Vorzeichen unsigned short

- Wahrheitswerte / Datentyp `BOOL`:  
4-Byte-Wert, wobei gilt: `FALSE = 0`, `TRUE = 1`

## 1.3.2 Hinweise für C/C++ - Programmierer

- Formale Parameter, die durch das Schlüsselwort **VAR** bzw. **CONST** eingeleitet werden, sind aus C/C++-Sicht vom Typ long pointer auf eine Variable vom angegebenen Typ, z.B.

```
FUNCTION I332Read2(I332Handle: INTEGER; cmd: WORD; VAR Value: LONGINT):  
INTEGER;
```

kann übersetzt werden als

```
int I332Read2(INT I332Handle, WORD cmd, LPLONG Value);
```

- Einbinden der DLL in eigene Projekte:  
Da die DLL mit Delphi erstellt wurde existiert keine LIB-Datei um die DLL statisch an ein Projekt zu linken. Dazu gibt es zwei Lösungen:

1. Entweder greift man auf ein entsprechendes Tool zurück, das eine LIB-Datei aus einer gegebenen DLL generiert

oder

2. Man bindet die DLL dynamisch ins Projekt ein, indem die DLL mit der WinAPI-Funktion `LoadLibrary` geladen wird und die Einsprung-Adressen der Funktionen per WinAPI-Funktion `GetProcAddress` ermittelt werden; die gefundenen Adressen werden dann einer Pointer-Variablen vom Typ der entsprechenden Funktion zugewiesen. Bei Programm-Ende muß die DLL mit `FreeLibrary` wieder freigegeben werden.



### 1.3.3 Hinweise für Visual-Basic – Programmierer

- Formale Parameter, die durch das Schlüsselwort **VAR** bzw. **CONST** eingeleitet werden, werden aus VB-Sicht **ByRef** übergeben, alle anderen **ByVal**, insbesondere gilt dies für String-Parameter. Visual Basic verwendet bei fehlendem Schlüsselwort standardmäßig „**ByRef**“, im Unterschied zu Delphi/Pascal (→ **ByVal**)! Es empfiehlt sich also, immer **ByRef** bzw. **ByVal** explizit anzugeben!
- Die Einbindung der DLL geschieht über eine globale Deklaration, z.B.

```
FUNCTION I332ExecCfgDlg(AI332Handle:  INTEGER;  AINIFileName:  PChar):  
INTEGER;
```

kann übersetzt werden als

```
Declare Function I332ExecCfgDlg Lib "I332.DLL" (ByVal I332Handle AS  
LONG, ByVal AINIFileName AS String) AS LONG
```

Eine komplette Umsetzung der PASCAL-Deklarationen ist in den Dateien  
I332\_DEF.BAS bzw. I332\_IFC.BAS zu finden.

## **2 Konstanten und Typen**

Für die Programmierung über die Basis-Funktionen I332Write0, I332Write1, I332Write2 sowie I332Read1 und I332Read2 werden die im I332-Handbuch beschriebenen und in der Datei I332.DEF aufgeführten Befehlskonstanten verwendet.

**Hinweise** für C/C++ und BASIC-Programmierer:

- Allgemein: In Delphi gelten Schlüsselwörter wie **CONST** oder **TYPE** solange kein anderes Schlüsselwort angegeben ist
- Konstanten-Vereinbarungen (Schlüsselwort **CONST**) in Delphi können als C-Macros mit **#define** und in BASIC als „**public const**“ geschrieben werden.
- Typ-Vereinbarungen (Schlüsselwort **TYPE**) können entsprechend mit „**typedef**“ bzw. „**public type**“ definiert werden.
- Bei der Vereinbarung von Konstanten wird gelegentlich die hexadezimale Schreibweise für INTEGER-Konstanten mit vorangestelltem „\$“ verwendet; in C/C++ entspricht dies der Schreibweise mit vorangestelltem „0x“, in BASIC mit vorangestelltem &H.

Darüber hinaus werden folgende Konstanten und Typen verwendet:

### 2.1 Konstanten

Konstanten zur Identifikation der Schnittstelle (z.B. in der Funktion `I332IFCInit`):

#### CONST

<code>imNoIFC</code>	<code>= 0;</code>	kein Interface
<code>imISA</code>	<code>= \$08;</code>	ISA-Bus-Version
<code>imPCI</code>	<code>= \$0C;</code>	PCI-Bus-Version
<code>imUSB</code>	<code>= \$0D;</code>	USB-Version
<code>imCOM</code>	<code>= \$10;</code>	serielle Version – binäre Kommunikation
<code>imCOM1</code>	<code>= imCOM+1;</code>	
<code>imCOM2</code>	<code>= imCOM+2;</code>	
<code>imCOM3</code>	<code>= imCOM+3;</code>	
<code>imCOM4</code>	<code>= imCOM+4;</code>	
(ab Version 6.4.13: <code>imCOM</code> bzw. <code>imCOM+1</code> bis <code>imCOM+15</code> zulässig, s. <code>I332IFCInit</code> )		
<code>imHEX</code>	<code>= \$30;</code>	serielle Version – ASCII-Hex-Kommunikation (nur für I332-Version 2.5.x)
<code>imHEX1</code>	<code>= imHEX+1;</code>	
<code>imHEX2</code>	<code>= imHEX+2;</code>	
<code>imHEX3</code>	<code>= imHEX+3;</code>	
<code>imHEX4</code>	<code>= imHEX+4;</code>	

#### Fehlerkonstanten:

<code>ieNoErr</code>	<code>= 0;</code>	kein Fehler
<code>ieInternal</code>	<code>= -1;</code>	interner Fehler
<code>ieNotImplemented</code>	<code>= -2;</code>	Funktion nicht implementiert
<code>ieUnknown</code>	<code>= -3;</code>	unbekannter Fehler
<code>ieNoIFC</code>	<code>= -10;</code>	Interface nicht initialisiert
<code>ieIFC</code>	<code>= -11;</code>	unzulässige Interface-Nr.
<code>ieCOM</code>	<code>= -12;</code>	keine Schnittstelle mit dieser Nr.
<code>ieDev</code>	<code>= -12;</code>	keine Schnittstelle mit dieser Nr.
<code>ieInvalid</code>	<code>= -13;</code>	Funktionsaufruf nicht zulässig
<code>ieDevErr</code>	<code>= -14;</code>	Device-Treiber-Fehler
<code>ieIFCRequired</code>	<code>= -15;</code>	I332Handle muß ein Schnittstelle-Handle (kein Buffer-Handle!) sein
<code>ieNoI332</code>	<code>= -20;</code>	I332 nicht gefunden
<code>ieTimeout</code>	<code>= -21;</code>	Timeout: I332 antwortet nicht
<code>ieSync</code>	<code>= -22;</code>	I332 konnte nicht synchronisiert werden
<code>ieXmt</code>	<code>= -23;</code>	Fehler beim Senden
<code>ieRcv</code>	<code>= -24;</code>	Fehler beim Empfang
<code>ieBuffer</code>	<code>= -30;</code>	allg. Bufferproblem
<code>ieNoBuffer</code>	<code>= -31;</code>	kein Buffer vorhanden / Bufferfunktionen werden nicht unterstützt
<code>ieBufferFull</code>	<code>= -32;</code>	Buffer voll

### 2.2 Typen

Typdefinition zur Abfrage der Schnittstellen- und I332-Information:

```
TYPE TI332ID = ARRAY[0..15] OF Char;      I332-Seriennummer: 16 Hexziffern
TYPE TI332Info = RECORD
    ifcMode: LONGINT;      verwendete Schnittstelle (eine der Konstanten imXXX)
    ifcPara: LONGINT;      Schnittstellenparameter (s. I332IFCInit)
    ifcErr: LONGINT;       zuletzt aufgetretener Schnittstellenfehler (Konstante ieXXX)
    i332Ver: LONGINT;      I332-Version
    i332ID: TI332ID;       I332-Seriennummer: 16 Hexziffern
END;
```

Konstantendefinition zur Programmierung von Verfahrenswegen:

```
CONST Axes      = 8;          max. Anzahl der I332-Achsen
    FEilgang     = 0;          Für Interpolations-Routinen: F = 0 → Eilgang
    UndefL       = $80000000; ungültige Achsposition

    fAbs         = $00000001; Absolut-Position
    fRel         = $00000000; Relativ-Position
```

Typdefinition zur Programmierung von Achspositionen:

```
TYPE TAxisArray = ARRAY [0..Axes-1] OF LONGINT;
```

Wird hier für eine der Achsen ein Wert UndefL programmiert, so wird dieser nicht an die I332 übergeben (→ die Achse wird nicht verfahren)

Rückgabewerte div. Routinen (fehlerhafte Daten/Parameter u.ä.)

```
deNoError = 0;    kein Fehler
deAxisNo  = 1;    falsche Achsnummer(n)
deSFaktiv = 2;    Funktion kann nicht ausgeführt werden (Sonderfunktion aktiv)
deSFrtg0  = 3;    fehlende Richtungsangabe
deRtgLS   = 4;    Endschalter aktiv
deSFDist0 = 5;    fehlende Distanzangabe / Distanz = 0
deIPL_OV  = 6;    Interpolationsbuffer der I332 ist voll
```

## **3 Funktionen**

Zur Kommunikation mit der I332 genügen die ersten 8 Funktionen (s.a. Handbuch I332); alle weiteren Funktionen dienen dem einfacheren Zugriff auf einzelne Register (z.B. `I332Set/GetAchsreg` oder `I332GetStaus`), realisieren auf dieser Basis komplexere Abläufe (z.B. `I332DoAxisRef` oder `I332DoIPLLin`) oder sind nur für spezielle Anforderungen notwendig (z.B. `I332WaitForMutex`).

Der besseren Übersichtlichkeit wegen wurden die Routinen in mehrere Gruppen unterteilt. Alle Funktionen mit Ausnahme der Funktion `I332IFCInit` erwarten als 1. Parameter ein `I332Handle`, das die Schnittstelle zur I332 identifiziert; die Funktion `I332IFCInit` liefert dieses Handle zurück.

Ab V.6.0 der I332.DLL sind Funktionen zur Block-Übertragung implementiert; diese benutzen einen Buffer, der durch ein `I332BufHandle` identifiziert wird. Einige der Funktionen akzeptieren auch ein solches Handle. Details dazu finden sich in Kap. 3.6 - USB- / Blockübertragung.

### 3.1 Verwaltung und Kommunikation

#### 3.1.1 I332IFCInit

**FUNCTION** I332IFCInit(ifcMode: WORD; ifcPara: LONGINT): INTEGER;

**Beschreibung** Diese Funktion initialisiert die Schnittstelle zur I332 und versucht eine Verbindung aufzubauen. Als Ergebnis wird ein Handle zurückgeliefert, das die I332-Karte identifiziert und bei allen anderen Funktionen der DLL als Parameter erwartet wird.

**Parameter**

ifcMode	Schnittstelle: eine der Konstanten imXXX wie oben beschrieben
ifcPara	je nach Schnittstelle sind folgende Parameter erforderlich:
imISA	kein Parameter erforderlich, es sollte ifcPara = 0 übergeben werden
imPCI	ifcPara = lfd. Nummer der PCI-Karte, wenn mehrere I332-PCI im System vorhanden sind; ifcPara = 0: 1. Karte im System etc.
imCOM+x	ifcPara = Baudrate; zulässig sind die Werte 9600, 19200, 38400 und 57600 ifcPara = 0: Standardbaudrate (57600) Für „x“ sind die Werte 1 bis 4 zulässig <b>Ab Version 6.4.13:</b> Es die Werte 1 bis 15 zulässig (serielle Schnittstellen COM1..COM15)
imUSB	kein Parameter erforderlich, es sollte ifcPara = 0 übergeben werden
imCOM	<b>Ab Version 6.4.13:</b> ifcPara = Pointer auf einen String der Form „COMx,Baudrate“; damit ist jede im System vorhandene serielle Schnittstelle ansprechbar. Wird „Baudrate“ nicht angegeben, so wird die Standardbaudrate (57600) verwendet.

**Funktionsergebnis** Handle auf die I332-Schnittstelle bzw. Schnittstellen-Fehlercode ieXXX

**Hinweise:** Solange keine speziellen Anforderungen gestellt werden, kann immer ifcPara = 0 übergeben werden.  
Es werden keine Achs- oder Maschinendaten übertragen oder initialisiert!

#### 3.1.2 I332IFCDone

**FUNCTION** I332IFCDone(I332Handle: INTEGER): INTEGER;

**Beschreibung** Beendet die Kommunikation mit der I332 und gibt die Schnittstelle wieder frei.

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.1.3 I332Reset

**FUNCTION** I332Reset(I332Handle: INTEGER): INTEGER;

**Beschreibung** Initialisiert die I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.1.4 I332Restart

**FUNCTION** I332Restart(I332Handle: INTEGER; icRestart: WORD): INTEGER;

**Beschreibung** Initialisiert die I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

icRestart Ein Restart-Befehl (s. Handbuch I332)

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

**Hinweis:** Solange ein Restart-Befehl abgearbeitet wird (je nach Restart-Befehl und Hardware-Version bis zu 250 ms), nimmt die I332 keine weiteren Befehle an; die Funktion I332Restart trägt diesem Umstand Rechnung. Restart-Befehle sollten deshalb nur über diese Funktion, nicht über I332Write0 aufgerufen werden.

### 3.1.5 I332Write0

**FUNCTION** I332Write0(I332Handle: INTEGER; icSet: WORD): INTEGER;

**Beschreibung** Überträgt das Befehlswort icSet an die I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

icSet Befehlswort vom Typ W0 (s. I332.def)

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

**Hinweis:** Restart-Befehle sollten über die Funktion I332Restart, nicht über I332Write0 aufgerufen werden.

### 3.1.6 I332Write1

**FUNCTION** I332Write1(I332Handle: INTEGER; icSet: WORD; Value: WORD):  
INTEGER;

**Beschreibung** Schreiben des Registers icSet: Überträgt das Befehlswort icSet und den dazugehörigen Wortparameter Value an die I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

icSet Befehlswort vom Typ W1 (s. I332.def)

Value Befehlsparameter

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## Funktionen

### 3.1.7 I332Write2

**FUNCTION** I332Write2(I332Handle: INTEGER; icSet: WORD; Value: LONGINT):  
INTEGER;

**Beschreibung** Schreiben des Registers icSet: Überträgt das Befehlswort icSet und den dazugehörigen Langwortparameter Value an die I332

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
icSet	Befehlswort vom Typ W2 (s. I332.def)
Value	Befehlsparameter

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.1.8 I332Read1

**FUNCTION** I332Read1(I332Handle: INTEGER; icGet: WORD; **VAR** Value: WORD):  
INTEGER;

**Beschreibung** Lesen des Registers icGet: Überträgt das Befehlswort icGet und liefert den Inhalt des Registers im Wortparameter Value zurück

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
icGet	Befehlswort vom Typ R1 (s. I332.def)
Value	Ergebniswort von der I332

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.1.9 I332Read2

**FUNCTION** I332Read2(I332Handle: INTEGER; icGet: WORD; **VAR** Value: LONGINT):  
INTEGER;

**Beschreibung** Lesen des Registers icGet: Überträgt das Befehlswort icGet und liefert den Inhalt des Registers im Langwortparameter Value zurück

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
icGet	Befehlswort vom Typ R2 (s. I332.def)
Value	Ergebnis-Langwort von der I332

**Funktionsergebnis** Schnittstellen-Fehlercode

### 3.1.10 I332DLLVersion

**FUNCTION** I332DLLVersion: LONGINT;

**Beschreibung** Liest die Versionsnummer der DLL

**Parameter** keine

**Funktionsergebnis** Versionsnummer, zu interpretieren als 4 einzelne Bytes

### 3.1.11 I332GetIFCErr

**FUNCTION** I332GetIFCErr(I332Handle: INTEGER): INTEGER;



## Funktionen

**Beschreibung** Liest den zuletzt aufgetretenen Schnittstellen-Fehlercode

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode

### 3.1.12 I332GetInfo

```
FUNCTION I332GetInfo(I332Handle: INTEGER; VAR I332Info: TI332Info):  
INTEGER;
```

**Beschreibung** Füllt eine Struktur vom Typ TI332Info mit Schnittstellen- und I332-Informationen

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

I332Info Pointer auf eine Struktur vom Typ TI332Info

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.1.13 I332GetVerStr

```
FUNCTION I332GetVerStr(I332Handle: INTEGER; VerStr: PChar; BufSize:  
INTEGER): INTEGER;
```

**Beschreibung** Liest den Versionsstring aus der I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

VerStr Pointer auf einen Buffer, der den nullterminierten String aufnimmt; der Buffer sollte eine Größe von 513 Byte haben (512 Zeichen + 0-Byte)

BufSize Größe des Buffers

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX bzw. tatsächliche Länge des Strings

### 3.1.14 I332MutexName

```
FUNCTION I332MutexName(I332Handle: INTEGER; Buffer: PChar; BufSize:  
INTEGER): INTEGER;
```

**Beschreibung** Liefert den Namen des für I332Handle verwendeten Mutex als nullterminierten String

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

Buffer Pointer auf einen Buffer, der den Namen aufnehmen soll

BufSize Größe des Buffers

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX bzw. tatsächliche Länge des Strings

**Hinweis:** Diese Funktion ist nur zu Informationszwecken vorhanden; bei Bedarf sollten die Funktionen I332WaitForMutex und I332ReleaseMutex verwendet werden.

### 3.1.15 I332WaitForMutex

```
FUNCTION I332WaitForMutex(I332Handle: INTEGER): INTEGER;
```

## **Funktionen**

**Beschreibung** Belegt den internen Mutex der I332-Schnittstelle

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### **3.1.16 I332ReleaseMutex**

**FUNCTION** I332ReleaseMutex(I332Handle: INTEGER): INTEGER;

**Beschreibung** Gibt den internen Mutex der I332-Schnittstelle wieder frei

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### **3.1.17 I332SetAchsReg**

**FUNCTION** I332SetAchsReg(I332Handle: INTEGER; icSet: WORD; AxisNo: ShortInt; Value: LONGINT): INTEGER;

**Beschreibung** Schreibt das Langwort Value in das Achsregister icSet der Achse AxisNo

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle  
icSet Befehlswort für ein Achsregister vom Typ W2 (s. I332.def)  
AxisNo Achsnummer im Bereich 0..7  
Value Befehlsparameter

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### **3.1.18 I332GetAchsReg**

**FUNCTION** I332GetAchsReg(I332Handle: INTEGER; icGet: WORD; AxisNo: ShortInt; VAR Value: LONGINT): INTEGER;

**Beschreibung** Liest das Achsregister icGet der Achse AxisNo

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle  
icGet Befehlswort für ein Achsregister vom Typ R2 (s. I332.def)  
AxisNo Achsnummer im Bereich 0..7  
Value Ergebnis-Langwort der I332 (Inhalt des Registers)

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### **3.1.19 I332AchsReg**

**FUNCTION** I332AchsReg(I332Handle: INTEGER; icGet: WORD; AxisNo: ShortInt): LONGINT;

**Beschreibung** Liest das Achsregister icGet der Achse AxisNo

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle  
icGet Befehlswort für ein Achsregister vom Typ R2 (s. I332.def)

## **Funktionen**

AxisNo                      Achsnummer im Bereich 0..7

**Funktionsergebnis**      Ergebnis-Langwort der I332 (Inhalt des Registers)

**Hinweis:**                  *Ein evtl. aufgetretener Fehler kann (muß aber nicht) über die Funktion I332GetIFCErr ermittelt werden*

### 3.2 Konfiguration

**Hinweise** zur Konfigurationsdatei bzw. zum Parameter *AINIFileName*:  
Enthält der Dateiname keine Pfadangabe, so wird die Datei im Verzeichnis der Applikation gesucht; wird als Dateiname lediglich eine Dateiendung (mit Punkt davor, also z.B. '.INI', angegeben, so wird im Verzeichnis der Applikation nach der Datei <Name der Applikation>.INI gesucht.  
Der formale Aufbau der Datei entspricht einer windows-üblichen INI-Datei.  
Im Gegensatz zur Datei C332.CFG für den DOS-Interpreter C332.EXE werden die Werte für die einzelnen Achsen in eigenen Abschnitten gespeichert. Um alte Konfigurationen jedoch einfach übernehmen zu können, wird beim Einlesen der Datei zunächst nach Abschnitten mit dem alten Format gesucht und die gefundenen Werte ggf. mit den Werten aus den Abschnitten für die einzelnen Achsen überschrieben.

#### 3.2.1 I332XmtCFG

**FUNCTION** I332XmtCFG(I332Handle: INTEGER; AINIFileName: PChar): INTEGER;

**Beschreibung** Überträgt achs- und maschinenspezifische Daten an die I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle  
AINIFileName Name der Datei, die die Daten enthält

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

#### 3.2.2 I332ExecCfgDlg

**FUNCTION** I332ExecCfgDlg(I332Handle: INTEGER; AINIFileName: PChar): INTEGER;

**Beschreibung** Ruft einen Dialog zur Einstellung der achs- und maschinenspezifischen Daten auf; beim Beenden des Dialogs mit der Schaltfläche „OK“ werden die Daten sowohl an die I332 übertragen als auch in einer Datei gespeichert.

Zusätzlich besteht die Möglichkeit einen Dialog zur Schnittstellen-Auswahl aufzurufen.

**Achtung:** In diesem Fall ist das bisherige I332Handle anschließend nicht mehr gültig! Die Funktion liefert das neue Handle zurück.

**Parameter**

I332Handle Handle auf die I332-Schnittstelle  
AINIFileName Name der Datei für die Daten

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX im Fehlerfall (ggf. neues) I332Handle sonst.

#### 3.2.3 I332InitMP

**FUNCTION** I332InitMP(I332Handle: INTEGER): INTEGER;

**Beschreibung** Initialisiert die zuvor übergebenen Maschinen- und Achs-Parameter der I332

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## **Funktionen**

**Hinweis:** *Da je nach Hardware-Version die Initialisierung bis zu 250ms dauern kann, ist der Aufruf dieser Funktion ist sicherer als der direkte Funktionsaufruf `I332Write0(icInitMP)`, da die Funktion diesem Umstand Rechnung trägt.*

## 3.3 Statusfunktionen

### 3.3.1 I332GetStatus

**FUNCTION** I332GetStatus(I332Handle: INTEGER; **VAR** I332Status: LONGINT):  
INTEGER;

**Beschreibung** Liest die Status-Information aus der I332; zur Bedeutung der einzelnen Bits s. Handbuch zur I332 (I332.PDF)

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.3.2 I332ZyklusStart

**FUNCTION** I332ZyklusStart(I332Handle: INTEGER): INTEGER;

**Beschreibung** Setzt die beiden Statusbits **stSSB** und **stSFB**; befinden sich Verfahrssätze in der I332, so werden sie jetzt ausgeführt bzw. fortgeführt

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.3.3 I332ZyklusStop

**FUNCTION** I332ZyklusStop(I332Handle: INTEGER): INTEGER;

**Beschreibung** Löscht die beiden Statusbits **stSSB** und **stSFB**; befinden sich Verfahrssätze in der I332, so werden sie jetzt angehalten

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.3.4 I332DoZyklusStart

**FUNCTION** I332DoZyklusStart(I332Handle: INTEGER; Start: BOOL): INTEGER;

**Beschreibung** Startet und stoppt Verfahrssätze in der I332

**Parameter**  
I332Handle Handle auf die I332-Schnittstelle  
Start TRUE: Startet Verfahrssätze (→ I332ZyklusStart)  
FALSE: Stoppt Verfahrssätze (→ I332ZyklusStop)

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

**Hinweis** zu den Bits **stSSB** und **stSFB**.  
Die Bits werden durch die I332 nicht verändert, insb. dann nicht wenn ein Verfahrssatz abgearbeitet ist. Das bedeutet auch, daß ein Verfahrssatz sofort abgearbeitet wird, wenn die Bits bei der Übergabe des Satzes noch bzw. schon gesetzt sind.

### 3.3.5 I332GetRefInfo

**FUNCTION** I332GetRefInfo(I332Handle: INTEGER; **VAR** RefInfo: WORD): INTEGER;

**Beschreibung** liest die Referenzinformation für die einzelnen Achsen

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
RefInfo	Lowbyte - Bit gesetzt: Achse wurde referenziert

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

### 3.3.6 I332GetPos

**FUNCTION** I332GetPos(I332Handle: INTEGER; AxisNo: ShortInt; **VAR** Position: LONGINT): INTEGER;

**Beschreibung** liest die aktuelle Position einer Achse

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
AxisNo	Achsnummer im Bereich 0..7
Position	aktuelle Position in $\mu\text{m}$

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

**Hinweis:** *Für Servo-Achsen ist dies die Soll-Position!*

### 3.3.7 I332Pos

**FUNCTION** I332Pos(I332Handle: INTEGER; AxisNo: ShortInt): LONGINT;

**Beschreibung** liest die aktuelle Position einer Achse (wie I332GetPos)

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
AxisNo	Achsnummer im Bereich 0..7

**Funktionsergebnis** aktuelle Position in  $\mu\text{m}$

**Hinweis:** *Ein evtl. aufgetretener Fehler kann (muß aber nicht) über die Funktion I332GetIFCErr ermittelt werden*

### 3.3.8 I332GetVlSt

**FUNCTION** I332GetVlSt(I332Handle: INTEGER; **VAR** Value: LONGINT): INTEGER;

**Beschreibung** liest die aktuelle Bahngeschwindigkeit bei Interpolationssätzen

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
Value	aktuelle Bahngeschwindigkeit in mm/min; die Geschwindigkeit bezieht sich auf die in den achsspezifischen Daten definierten Bahnachsen. Ist keine der Bahnachsen an der Interpolation beteiligt, so bezieht sich die Geschwindigkeit auf die Achse mit dem längsten Weg.

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## Funktionen

**Hinweis:** Soll die Geschwindigkeit einer einzelnen Achse abgefragt werden, so muß das Achsregister *icGetSFVCur* der gewünschten Achse gelesen werden. Dies gilt insbesondere, wenn diese Achse eine Sonderfunktion (z.B. eine Referenzfahrt) ausführt.

### 3.3.9 I332VlSt

**FUNCTION** I332VlSt(I332Handle: INTEGER): LONGINT;

**Beschreibung** liest die aktuelle Bahngeschwindigkeit bei Interpolationssätzen (wie I332GetVlSt)

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

**Funktionsergebnis** aktuelle Bahngeschwindigkeit in mm/min; die Geschwindigkeit bezieht sich auf die in den achsspezifischen Daten definierten Bahnachsen. Ist keine der Bahnachsen an der Interpolation beteiligt, so bezieht sich die Geschwindigkeit auf die Achse mit dem längsten Weg.

**Hinweis:** Ein evtl. aufgetretener Fehler kann (muß aber nicht) über die Funktion *I332GetIFCErr* ermittelt werden.

### 3.3.10 I332GetAxAArray

**FUNCTION** I332GetAxAArray(I332Handle: INTEGER; icGet: WORD; VAR AxAArray: TAxisArray): LONGINT;

**Beschreibung** liest die einen Langwort-Registersatz für alle Achsen.

**Parameter**

I332Handle Handle auf die I332-Schnittstelle

icGet ein Achsregister-Befehl vom Typ R2 (z.B. *icGetPos*)

AxAArray Langwortfeld für die Ergebnisse (für *icGet* = *icGetPos* stehen hier dann z.B. die Positionen aller Achsen).

**Funktionsergebnis** Schnittstellen-Fehlercode *ieXXX*



### 3.4 Sonderfunktionen

Sonderfunktionen lassen sich für alle Achsen unabhängig voneinander und parallel zur Interpolation ausführen. Für die Programmierung steht ein Puffer zur Verfügung mit einem Registersatz für jede Achse; Details dazu s. I332-Handbuch.

Die Achsbewegung startet, wenn das Statusbit `stSFB` gesetzt ist.

**Hinweis:** *Wegen der internen Verwaltung der Positionsdaten von Interpolationssätzen und Sonderfunktionen in getrennten Registersätzen kann es vorkommen, daß die von den Registersätzen verwalteten Positionen nicht synchron sind; ein Restart-Befehl (`icRestart1..5`, `icRestart11..15`) synchronisiert die beiden Registersätze wieder neu. Dies ist immer dann notwendig, wenn eine Achse Sonderfunktionen ausgeführt hat und danach mit anderen Achsen zusammen wieder interpolierend verfahren soll. Details dazu finden sich im Handbuch I332.*

#### 3.4.1 I332DoAxisRef

**FUNCTION** I332DoAxisRef(I332Handle: INTEGER; AxisNo: ShortInt): INTEGER;

**Beschreibung** Führt eine Referenzfahrt für die Achse `AxisNo` aus; die Parameter (Geschwindigkeit, Richtung etc.) werden in den achsspezifischen Daten (s. Kap. 3.2) festgelegt

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
AxisNo	Achsnummer im Bereich 0..7

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Referenzfahrt nicht ausgeführt werden konnte

Die Referenzfahrt läuft in folgenden Phasen ab:

1. Die Achse fährt in der in `apLS` definierten Richtung mit der Geschwindigkeit `apVRef`.
2. Wird der Endschalter für diese Richtung erkannt, so wird die Achse gestoppt
3. Wurde der ES überfahren, so wird zunächst mit der Geschwindigkeit `apVHys` auf den ES zurück positioniert.
4. Anschließend wird der ES ebenfalls mit der Geschwindigkeit `apVHys` freigefahren, entspr. der in den Maschinendaten definierten Richtung (`apLS.lsPHRtgB` bzw. `lsMHRtgB`)
5. Danach wird in die gleiche Richtung wieder mit der Geschwindigkeit `apVHys` um den in `apLSHys` definierten Weg vom ES weg positioniert (Schalt-Hysterese des Endschalters / Feinreferenz). Kann der ES nach einer bestimmten Strecke nicht freigefahren werden (`apHysMax`), so wird mit einer Fehlermeldung abgebrochen.

Auswertung des Refpuls-Eingangs (nur mit Servoverversionen):

6. Ist das Bit `apDC.dcRefInB` gesetzt, so wird entspr. der in `apLS` definierten Richtung (`lsPRRtgB` bzw. `lsMRtgB`) der Referenz-Puls mit der Geschwindigkeit `apVRefP` gesucht. Sobald der entsprechende Eingang aktiv ist, wird die Achse gestoppt.
7. Ist außerdem noch ein Wert `apRefOffs` definiert, so wird die Achse um diesen Wert vom Referenzpuls weg positioniert.
8. Als letztes werden das Positionsregister entspr. `apRefPos` und ggf. die Verfahrgrenzen (Software-Endschalter) `apLSDeltaM` und `apLSDeltaP` gesetzt.

#### 3.4.2 I332DoAxisAsync

**FUNCTION** I332DoAxisAsync(I332Handle: INTEGER; AxisNo: ShortInt; Dist, HndF: LONGINT; sfMode: WORD): INTEGER;

## Funktionen

**Beschreibung** Verfährt die Achse `AxisNo` zu einer laufenden Interpolation

### Parameter

<code>I332Handle</code>	Handle auf die I332-Schnittstelle
<code>AxisNo</code>	Achsnummer im Bereich 0..7
<code>Dist</code>	Distanz in $\mu\text{m}$ ; wird als Distanz = 0 übergeben, so ist die Funktion identisch mit <code>I332DoAxisHand</code>
<code>HndF</code>	Geschwindigkeit in mm/min
<code>sfMode</code>	eine der Konstanten <code>sfXXX</code> (s. I332.def)

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn der Befehl nicht ausgeführt werden konnte

### 3.4.3 I332DoAxisHand

**FUNCTION** `I332DoAxisHand(I332Handle: INTEGER; AxisNo: ShortInt; HndF: LONGINT; sfMode: WORD): INTEGER;`

**Beschreibung** Verfährt die Achse `AxisNo` mit der Geschwindigkeit `HndF`

### Parameter

<code>I332Handle</code>	Handle auf die I332-Schnittstelle
<code>AxisNo</code>	Achsnummer im Bereich 0..7
<code>HndF</code>	Geschwindigkeit in mm/min, das Vorzeichen bestimmt die Richtung der Bewegung; wird <code>HndF = 0</code> übergeben, so wird die Bewegung beendet.
<code>sfMode</code>	eine der Konstanten <code>sfXXX</code> (s. I332.def)

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn der Befehl nicht ausgeführt werden konnte

## 3.5 Interpolation

Interpolationsfunktionen werden intern in einem Satzpuffer zwischengespeichert; dieser umfaßt je nach Speicherausbau der I332 ca. 150-1000 Sätze bzw. Interpolationsfunktionen. Organisiert ist der Satzpuffer als FIFO-Speicher (First In - First Out, auch Warteschlange). Die Daten vom PC gelangen dabei in den Satz am einen Ende, während der Interpolator bereits die vorher eingegebenen Sätze verarbeitet (d.h. die Achsen verfährt). Damit lassen sich kontinuierliche Bahnbewegungen aus aufeinanderfolgenden Bahnstücken ohne weitere Maßnahmen realisieren.

Die Achsbewegung startet, wenn das Statusbit `stSSB` gesetzt ist.

**Hinweis:** *Wegen der internen Verwaltung der Positionsdaten von Interpolationssätzen und Sonderfunktionen in getrennten Registersätzen kann es vorkommen, daß die von den Registersätzen verwalteten Positionen nicht synchron sind; ein Restart-Befehl (`icRestart1..5`, `icRestart11..15`) synchronisiert die beiden Registersätze wieder neu. Dies ist immer dann notwendig, wenn eine Achse Sonderfunktionen ausgeführt hat und danach mit anderen Achsen zusammen wieder interpolierend verfahren soll. Details dazu finden sich im Handbuch I332.*

### 3.5.1 I332DoPos1

**FUNCTION** I332DoPos1(I332Handle: INTEGER; AxisNr: ShortInt; Dist,F: LONGINT; Flags: LONGINT): INTEGER;

**Beschreibung** Positioniert die Achse `AxisNo` absolut oder relativ

#### Parameter

I332Handle	Handle auf die I332-Schnittstelle
AxisNo	Achsnummer im Bereich 0..7
Dist	Zielposition ( <code>Flags = fAbs</code> ) bzw. Distanz ( <code>Flags = fRel</code> ) in µm
F	Geschwindigkeit in mm/min; wird <code>F=0</code> angegeben, so wird im Eilgang positioniert
Flags	Verknüpfung von Konstanten <code>fXXX</code>

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Funktion nicht ausgeführt werden konnte

### 3.5.2 I332DoIPLLin

**FUNCTION** I332DoIPLLin(I332Handle: INTEGER; AxisCnt: ShortInt; CONST AxArray: TAxisArray; F: LONGINT; Flags: LONGINT): INTEGER;

**Beschreibung** Positioniert die ersten `AxisCnt` Achsen absolut oder relativ; die Bewegung erfolgt interpolierend

#### Parameter

I332Handle	Handle auf die I332-Schnittstelle
AxisCnt	Achsanzahl im Bereich 1..8; die in <code>AxArray</code> angegebenen Daten werden nur für die Achsen 0.. <code>AxisCnt-1</code> ausgewertet
AxArray	Array mit Zielpositionen ( <code>Flags = fAbs</code> ) bzw. Distanzen ( <code>Flags = fRel</code> ) der Achsen in µm; wird für eine Achse der Wert <code>UndefL</code> (\$80000000) programmiert, so wird diese Achse nicht berücksichtigt.
F	Bahngeschwindigkeit in mm/min; wird <code>F=0</code> angegeben, so wird im Eilgang positioniert. Die Geschwindigkeit bezieht sich auf die in den achsspezifischen Daten definierten Bahnachsen. Ist keine der Bahnachsen an der Interpolation beteiligt, so bezieht sich die Geschwindigkeit auf die Achse mit dem längsten Weg.
Flags	Verknüpfung von Konstanten <code>fXXX</code>

## Funktionen

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Funktion nicht ausgeführt werden konnte

### 3.5.3 I332DoIPLCW

**FUNCTION** I332DoIPLCW(I332Handle: INTEGER; AxisCnt: ShortInt; **CONST** AxArray: TAxisArray; F: LONGINT; Flags: LONGINT; XAx, YAx: ShortInt; XMP, YMP: LONGINT): INTEGER;

**Beschreibung** Führt einen Kreisbogen im Uhrzeigersinn aus; Achsen die nicht an der Kreisbewegung beteiligt sind, werden linear zum Kreisbogen interpoliert (Helix-Interpolation)

#### Parameter

I332Handle	Handle auf die I332-Schnittstelle
AxisCnt	Achsanzahl im Bereich 1..8; die in AxArray angegebenen Daten werden nur für die Achsen 0..AxisCnt-1 ausgewertet
AxArray	Array mit Zielpositionen (Flags = fAbs) bzw. Distanzen (Flags = fRel) der Achsen in µm; wird für eine Achse der Wert <code>UndefL</code> (\$80000000) programmiert, so wird diese Achse nicht berücksichtigt.
F	Bahngeschwindigkeit in mm/min; wird F=0 angegeben, so wird im Eilgang positioniert. Die Geschwindigkeit bezieht sich auf die beiden Kreisachsen.
Flags	Verknüpfung von Konstanten <code>fXXX</code>
XAx, YAx	Achsnummern der beiden Kreisachsen
XMP, YMP	Distanzen Anfangspunkt→Kreismittelpunkt in µm (unabhängig von Flags immer relativ)

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Funktion nicht ausgeführt werden konnte

### 3.5.4 I332DoIPLCCW

**FUNCTION** I332DoIPLCCW(I332Handle: INTEGER; AxisCnt: ShortInt; **CONST** AxArray: TAxisArray; F: LONGINT; Flags: LONGINT; XAx, YAx: ShortInt; XMP, YMP: LONGINT): INTEGER;

**Beschreibung** Führt einen Kreisbogen im Gegenuhrzeigersinn aus; Achsen die nicht an der Kreisbewegung beteiligt sind, werden linear zum Kreisbogen interpoliert (Helix-Interpolation)

#### Parameter

I332Handle	Handle auf die I332-Schnittstelle
AxisCnt	Achsanzahl im Bereich 1..8; die in AxArray angegebenen Daten werden nur für die Achsen 0..AxisCnt-1 ausgewertet
AxArray	Array mit Zielpositionen (Flags = fAbs) bzw. Distanzen (Flags = fRel) der Achsen in µm; wird für eine Achse der Wert <code>UndefL</code> (\$80000000) programmiert, so wird diese Achse nicht berücksichtigt.
F	Bahngeschwindigkeit in mm/min; wird F=0 angegeben, so wird im Eilgang positioniert. Die Geschwindigkeit bezieht sich auf die beiden Kreisachsen.
Flags	Verknüpfung von Konstanten <code>fXXX</code>
XAx, YAx	Achsnummern der beiden Kreisachsen
XMP, YMP	Distanzen Anfangspunkt→Kreismittelpunkt in µm (unabhängig von Flags immer relativ)

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Funktion nicht ausgeführt werden konnte

### 3.5.5 I332DoDelay

**FUNCTION** I332DoDelay(I332Handle: INTEGER; F: LONGINT; Flags: LONGINT):  
INTEGER;

**Beschreibung** Fügt eine Wartezeit zwischen zwei Interpolationsätze ein.

**Parameter**

I332Handle	Handle auf die I332-Schnittstelle
F	Wartezeit in ms
Flags	immer 0: noch nicht implementiert

**Funktionsergebnis** Schnittstellen-Fehlercode `ieXXX` bzw. eine der Konstanten `deXXX`, wenn die Funktion nicht ausgeführt werden konnte

## 3.6 USB- / Blockübertragung

Dieser Übertragungsmodus wurde zur besseren Unterstützung der USB-Schnittstelle geschaffen. Diese ist zwar schnell, jedoch vorwiegend zur Übertragung großer Datenmengen vorgesehen, was im Kommunikationsprotokoll der I332 bisher nicht vorgesehen war.

Die USB-Übertragung basiert auf Datenpaketen, die zwischen Sender und Empfänger in einem festen Zeitraster (sog. Frames) von 1ms ausgetauscht werden, unabhängig von der Größe der Datenpakete. Dazu kommt noch der Protokoll-Overhead zur Datensicherung (Handshake-/Quittungs-Pakete auf USB-Ebene), so daß für ein einzelnes Datenpaket (bei fehlerfreier Übertragung!) 3 Frames notwendig sind.

Zu beachten ist außerdem, daß sich alle Geräte an einem USB-Anschluß die zur Verfügung stehende Bandbreite teilen müssen, was zur Verringerung der Übertragungsleistung an ein einzelnes Gerät führen kann!

Für weitere Details wird auf die einschlägige Literatur verwiesen.

### 3.6.1 Allgemeines

Um die Übertragung über die USB-Schnittstelle effizienter zu machen, wird ein Buffer verwendet, in den die zu übertragenden I332-Befehle eingetragen werden. Der Buffer ist wortweise organisiert, wobei für jeder Befehl entsprechend viele aufeinanderfolgende Worte belegt: ein R1- oder W0-Befehl belegt 1 Wort, ein R2- oder W1-Befehl 2 Worte und ein W2-Befehl 3 Worte. Jeweils das erste Wort enthält den Befehl; bei W1- oder W2- Befehlen folgen anschließend die Daten, bei einem R2-Befehl bleibt das anschließende Wort leer. Nach der Übertragung des Buffers enthält er die Quittungsworte der I332 an der entsprechenden Position im Buffer, d.h. die I332-Befehle werden mit den Werten aus der I332 überschrieben; bei R1- und R2-Befehlen sind dies die angeforderten Werte, bei W0-, W1- und W2-Befehlen Quittungsdaten. Der Buffer kann beliebig groß sein.

Buffer-Belegung - Schema:

Index 0..i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9	i+10...
.....	icCmd (R1)	icCmd (R2)	Dummy Wort	icCmd (W0)	icCmd (W1)	Daten- Wort	icCmd (W2)	Daten Lowort	Daten Hiwort	.....

Dabei bedeutet `icCmd` ein I332-Befehl vom angegebenen Typ. Nach der Übertragung dieses Buffers mit der Funktion `I332XmtBuffer` finden sich dort die Rückgabewerte der I332:

Index 0..i	i+1	i+2	i+3	i+4	i+5	i+6	i+7	i+8	i+9	i+10...
.....	Result R1	Result R2 Lowort	Result R2 Hiwort	Unde- finiert	Unde- finiert	Unde- finiert	Unde- finiert	Unde- finiert	Unde- finiert	.....

D.h. an der Position `i+1` steht der Wert, den ein Funktionsaufruf `I332Read1(i332Handle,icCmd,ResultR1)` und bei `i+2` und `i+3` der, den ein Aufruf `I332Read2(i332Handle,icCmd,ResultR2)` geliefert hätte; die Funktionen `I332Write..` liefern keine expliziten Daten aus der I332, deshalb sind die entsprechenden Buffereinträge undefiniert.

Im Prinzip genügt die Funktion `I332XmtBuffer`, um diesen Buffer-Mechanismus zu nutzen.

Um jedoch bestehende Applikationen mit wenig Aufwand an die USB-Gegebenheiten anpassen zu können und von der Bufferverwaltung zu entlasten, wurden Funktionen implementiert, die zum einen diese Verwaltung zum großen Teil übernehmen, zum andern den Mechanismus auch für die bisherigen Funktionen, insb. für die Basifunktionen `I332Read..` und `I332Write..`, verfügbar machen:

- `I332InitBuffer`
- `I332FreeBuffer`
- `I332FlushBuffer`

## Funktionen

- I332ClearBuffer
- I332GetBuffer

### Hinweise:

- Die USB-Übertragung funktioniert ohne Änderung auch mit den bisherigen Funktionen I332Read1, I332Read2, I332Write0, I332Write1 und I332Write2, ist jedoch nicht so effizient.
- Umgekehrt funktioniert der Buffer-Mechanismus auch bei den anderen I332-Varianten, führt dort aber nicht zu einer wesentlichen Steigerung der Leistung.
- Komplexere Funktionen (wie z.B. die I332DoIPL...-Funktionen und insb. I332GetAxArray) machen intern vom beschriebenen Buffer-Mechanismus Gebrauch.

## 3.6.2 I332XmtBuffer

**FUNCTION** I332XmtBuffer(I332Handle: INTEGER; Buffer: Pointer; WordCnt: Cardinal): INTEGER;

**Beschreibung** Überträgt die I332-Befehle im Buffer an die I332; die Ergebnisse befinden sich anschließend ebendort, wie oben beschrieben.

### Parameter

I332Handle	Handle auf die I332-Schnittstelle
Buffer	Pointer auf den Buffer
WordCnt	Größe des Buffers in Worten!

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## 3.6.3 I332InitBuffer

**FUNCTION** I332InitBuffer(I332Handle: INTEGER; Buffer: Pointer; MaxSize: INTEGER; AW\_O: BOOL): INTEGER;

**Beschreibung** Initialisiert einen Buffer und erzeugt ein Handle für diesen Buffer und liefert es zurück; dieses kann anstelle eines I332Handle beim Aufruf anderer Funktionen übergeben werden. Damit können (fast) alle Funktionen, die ein I332Handle erwarten, den Buffer-Mechanismus ohne weitere Maßnahmen benutzen. Insb. sind das die Funktionen I332Read1, I332Read2, I332Write0, I332Write1 und I332Write2. Eine detaillierte Beschreibung findet sich weiter unten.

### Parameter

I332Handle	Handle auf die I332-Schnittstelle
------------	-----------------------------------

Buffer	Pointer auf den Buffer; wird der Wert NIL übergeben, so wird intern der benötigte Speicherplatz bereitgestellt. In diesem Fall können die Ergebnisse der Buffer-Übertragung mit der Funktion I332GetBuffer ausgelesen werden.
--------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**Hinweis:** *Buffer = NIL ist vor allem dann sinnvoll, wenn nur W-Befehle übertragen werden sollen.*

MaxSize	maximale Größe des Buffers in Worten!
---------	---------------------------------------

AW_O	Der Buffer enthält nur I332-Befehle vom Typ W0, W1 oder W2; da keine expliziten Daten aus der I332 benötigt werden, kann auf das Abholen eines ohnehin irrelevanten Datenblocks über die USB-Schnittstelle verzichtet werden.
------	-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

**NB:** *Die Anwendung ist dafür verantwortlich, daß im Buffer tatsächlich nur W-Befehle stehen!*

## Funktionen

**Funktionsergebnis** I332BufHandle des erzeugten Buffers (> 0)  
oder Schnittstellen-Fehlercode ieXXX

### Hinweise:

- Implizit wird das I332BufHandle dem I332Handle und damit der I332-Schnittstelle zugeordnet; dies bedeutet auch, daß das I332BufHandle nur solange gültig ist, wie auch das I332Handle gültig ist. Wird also die Schnittstelle mit I332IFCDone geschlossen und damit das I332Handle ungültig, dann gilt dies auch für alle I332BufHandles die mit I332InitBuffer und dem (jetzt ungültigen I332Handle) erzeugt wurden.
- Für ein I332Handle können beliebig viele I332BufHandles erzeugt werden, die voneinander unabhängig sind.
- Stellt die Anwendung den Buffer zur Verfügung, so ist sie auch für die Freigabe des Speichers zuständig.

## 3.6.4 I332FreeBuffer

**FUNCTION** I332FreeBuffer(I332BufHandle: LONGINT): INTEGER;

**Beschreibung** Gibt den Buffer wieder frei; wurde beim Initialisieren kein Buffer angegeben (Buffer = NIL, d.h. es wurde ein interner Buffer erzeugt), so wird auch der (intern) für den Buffer belegte Speicher wieder freigegeben; im andern Fall ist die Anwendung dafür zuständig (nach Aufruf von I332FreeBuffer), den bei I332InitBuffer im Parameter „Buffer“ übergebenen Speicher wieder freizugeben! Das I332BufHandle ist anschließend ungültig! Enthält der dem I332BufHandle zugeordnete Buffer noch nicht übertragene Daten, so werden diese noch übertragen!

### Parameter

I332BufHandle Handle des Buffers

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## 3.6.5 I332FlushBuffer

**FUNCTION** I332FlushBuffer(I332BufHandle: LONGINT): INTEGER;

**Beschreibung** Überträgt etwaige noch ausstehende Daten an die I332. Der Buffer wird aber nicht zurückgesetzt, d.h. der Buffer kann noch weiter mit Daten gefüllt werden.

### Parameter

I332BufHandle Handle des Buffers

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

**Hinweis:** Implizit wird diese Funktion aufgerufen, wenn der Buffer gefüllt ist oder wenn die maximal mögliche Blockgröße für die Übertragung erreicht ist. Letzteres kann bei der USB-Version auftreten, da hier die Blockgröße durch das USB Protokoll beschränkt ist; für die Anwendung ist dieses Verhalten transparent. Zu beachten ist, daß deshalb u.U. ein Teil der Daten nicht erst beim expliziten Aufruf der Funktion I332FlushBuffer übertragen wird, sondern bereits früher.

## 3.6.6 I332ClearBuffer

**FUNCTION** I332ClearBuffer(I332BufHandle: LONGINT): INTEGER;



## Funktionen

**Beschreibung** Überträgt alle Daten im Buffer, soweit noch nicht geschehen und setzt ihn zurück; der Buffer kann anschließend für weitere (neue) Daten-Übertragungen genutzt werden.

### Parameter

I332BufHandle Handle des Buffers

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## 3.6.7 I332GetBuffer

**FUNCTION** I332GetBuffer(I332BufHandle: LONGINT; ExtBuffer: Pointer;  
WordCnt, WordPos: INTEGER): INTEGER;

**Beschreibung** Wurde beim Aufruf der Funktion I332InitBuffer kein Buffer (Buffer = NIL) angegeben, so müssen die Daten mit dieser Funktion vom intern erzeugten Buffer ausgelesen werden. Beginnend an der Position WordPos werden WordCnt Worte aus dem internen Buffer gelesen; ExtBuffer zeigt auf den Speicherbereich der die Daten aufnehmen soll; dieser muß mindestens WordCnt Worte groß sein.

### Parameter

I332BufHandle Handle des Buffers

ExtBuffer Pointer auf den Speicherbereich für die gelesenen Daten

WordCnt Anzahl der zu lesenden Datenworte

WordPos Startposition für das Auslesen im internen Buffer

**Funktionsergebnis** Schnittstellen-Fehlercode ieXXX

## 3.6.8 Arbeiten mit Buffern

### 3.6.8.1 I332Read.. / I332Write..

**Aufruf der Funktionen I332Read.. und I332Write.. mit einem I332BufHandle anstelle eines I332Handle:**

1. Beschaffen eines I332BufHandle: Aufruf der Funktion I332InitBuffer.
2. Aufruf der gewünschten Funktionen: anstatt eines I332Handle wird das I332BufHandle als 1. Parameter übergeben.  
Die I332-Befehle werden zunächst automatisch korrekt in den durch das I332BufHandle identifizierten Buffer eingetragen.  
Im Fehlerfall wird eine (negative) Fehlernummer ieXXX zurückgegeben. Dies ist insbesondere der Fall, wenn der Buffer voll ist.
3. Sind alle Befehle übergeben, so wird durch Aufruf der Funktion I332FlushBuffer der Buffer an die I332 übertragen bzw. mit den Rückgabewerten der I332 gefüllt.  
**Hinweis:** *Implizit wird diese Funktion aufgerufen, wenn der Buffer gefüllt ist oder wenn die maximal mögliche Blockgröße für die Übertragung erreicht ist.  
Letzteres kann bei der USB-Version auftreten, da hier die Blockgröße durch das USB Protokoll beschränkt ist; für die Anwendung ist dieses Verhalten transparent.  
Zu beachten ist deshalb, daß u.U. ein Teil der Daten nicht erst beim expliziten Aufruf der Funktion I332FlushBuffer übertragen wird, sondern bereits vorher übertragen wurde.*
4. Wurde der Buffer von der Anwendung selbst zur Verfügung gestellt, so stehen die Daten direkt zur Verfügung, andernfalls können sie nun mit der Funktion I332GetBuffer ausgelesen werden.
5. Der Buffer kann nun mit der Funktion I332ClearBuffer zurückgesetzt und für weitere Zwecke verwendet werden.
6. Wird der Buffer nicht mehr gebraucht, so wird er mit der Funktion I332FreeBuffer gelöscht.

## **Funktionen**

### **Rückgabewert der Funktionen:**

Anstatt einer 0 (= kein Fehler) liefern die Funktionen die Position im Buffer zurück, an der der I332-Befehl eingetragen wurde bzw. an der nach der Übertragung der zugehörige Wert aus der I332 steht.

### **Parameter „Value“ der Funktionen I332Read1 und I332Read2:**

Dieser Parameter ist zwar aus syntaktischen Gründen erforderlich, sein Rückgabewert ist jedoch undefiniert.

### **3.6.8.2 Interpolationsfunktionen I332DoIPL... bzw. I332DoDelay**

Diese Funktionen akzeptieren ebenfalls ein I332BufHandle anstatt eines I332Handle, jedoch sind einige Besonderheiten zu beachten:

- Die Prüfung, ob genügend Platz für den nächsten Interpolationssatz in der I332 ist, obliegt der Anwendung.
- Spätestens nach dem letzten Interpolationssatz muß mit der Funktion I332FlushBuffer oder I332ClearBuffer die Übertragung der Daten veranlaßt werden.

### **3.6.8.3 sonstige Funktionen**

Alle Funktionen, die kein I332BufHandle akzeptieren, liefern den Fehlercode ieIFCRequired zurück.

## 4 Demo-Programm I332Demo.EXE

Das Demo-Programm I332Demo.EXE dient dazu, einige Funktionen sowohl der I332.DLL als auch der I332 selbst zu demonstrieren. Das Programm ist lauffähig unter Windows 9x/ME/NT/2000/XP. Der Quelltext steht als Delphi-Programm ebenfalls zur Verfügung. Folgende Dateien enthalten den Quelltext für das Demo-Programm:

I332Demo.DPR            Delphi-Projektdatei

frm\_I332Main.PAS        Quelltext für das Hauptfenster des Programms

fm\_I332XXX.PAS         Quelltext der Demo-Programme, realisiert als TFrame-Objekte

Die DLL-Aufrufe befinden sich jeweils in den Routinen, die bei einem Klick auf das entsprechende Fensterelement aufgerufen werden; diese tragen jeweils die Namen

```
procedure TfrmI332.XXXXXClick(Sender: TObject);
```

wobei das XXXXX das jeweilige Fenster-Element kennzeichnet. Die Zuordnung sollte aus den Namen ersichtlich sein (z.B. heißt die Klick-Routine für den Referenzfahrt-Button

```
procedure TfrmI332.btnRefClick(Sender: TObject);
```

frm\_I332Main.DFM        Hauptfenster des Programms

fm\_I332XXX.DFM         Fenster der Demo-Frames  
Delphi-Ressourcen-Dateien (werden von der Delphi-Entwicklungs-Umgebung erzeugt und verwaltet).

Im Verzeichnis „Common“:

I332\_DEF.PAS            Konstanten und Typdefinitionen

I332\_IFC.PAS            Interface zur I332.DLL

I332.DEF                Befehlskonstanten

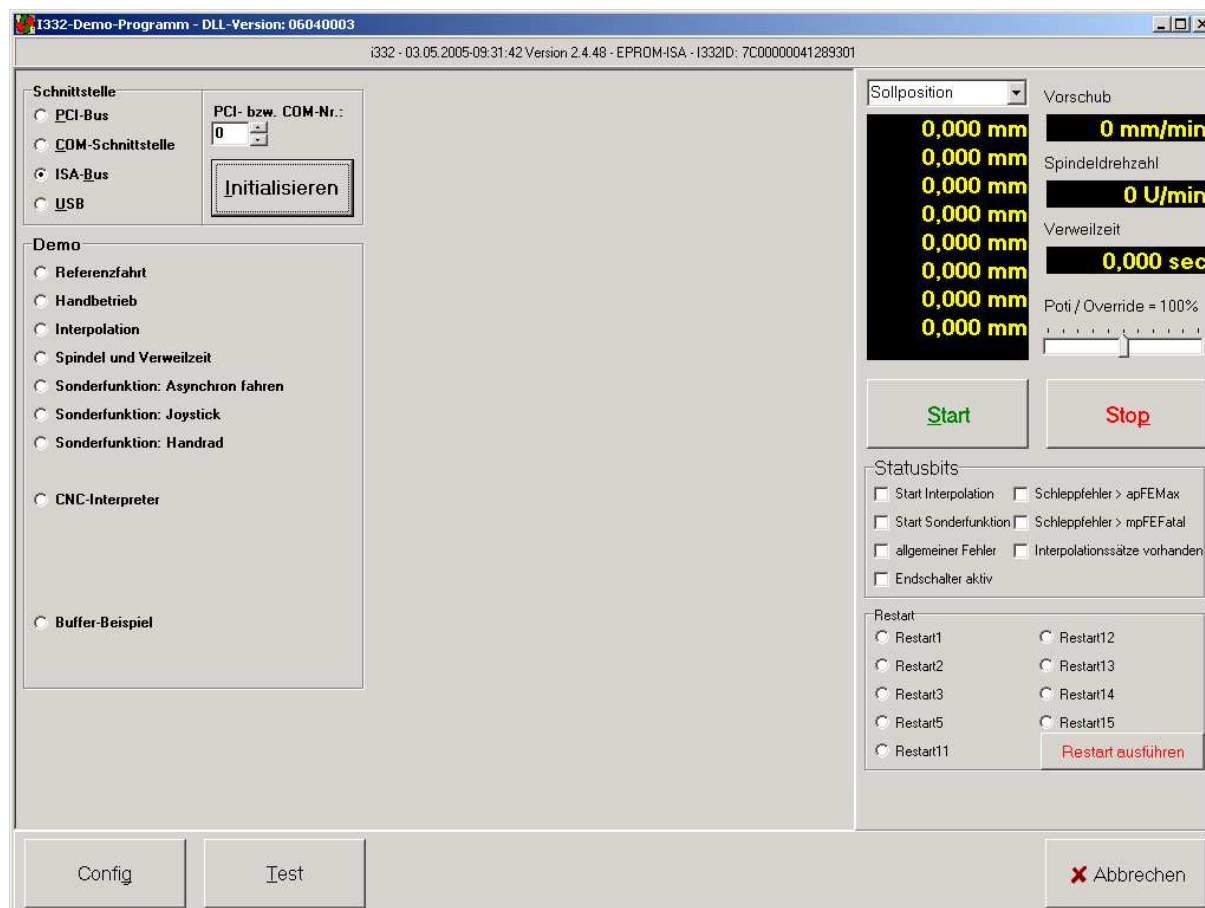
Die übrigen Dateien werden von der Delphi-Entwicklungsumgebung erzeugt bzw. verwaltet.

**Hinweis:**            *Der Übersichtlichkeit halber werden außer bei der Initialisierung die Rückgabewerte der Funktionen (Fehlercodes der Schnittstelle) nicht ausgewertet.*

**Hinweis:**            *Die beiliegende Datei I332.INI initialisiert die I332 **ohne** Endschalter und alle Achsen als Schrittmotor-Achsen; damit lassen sich die Verfahrbewegungen ohne weitere Änderungen ausprobieren. Für eine Referenzfahrt müssen jedoch Endschalter angeschlossen und über den Konfigurationsdialog konfiguriert werden!*

Und noch ein **Hinweis** für C/C++/VB-Programmierer:

*In PASCAL/Delphi ist es nicht nur möglich, innerhalb eines Unterprogramms Typen, Variablen oder Konstanten zu definieren, sondern auch weitere (untergeordnete) Unterprogramme, die nur innerhalb des übergeordneten Unterprogramms sichtbar sind (s.z.B. **procedure** TfrmI332.Timer1Timer)!*



## Schnittstelle

Im Feld Schnittstelle kann eine der möglichen Schnittstellen ausgewählt werden; falls PCI-Bus oder COM-Schnittstelle ausgewählt ist, muß außerdem die lfd. Nr. der PCI-Karte bzw. die Nr. der seriellen Schnittstelle ausgewählt werden.

## Initialisieren

**procedure** TfrmI332.btnInitClick

Der Button ruft die Funktion I332IFCInit auf; liefert die Funktion ein gültiges Handle, so werden über die Funktionen I332GetInfo und I332GetVerStr Informationen zur Schnittstelle und zur I332 abgefragt und in einem Panel am oberen Rand des Fensters angezeigt (s.Abb.). Zum Schluß werden die Achs- und Maschinendaten aus der Datei I332.INI mit der Funktion I332XmtCFG an die I332 übertragen.

**Hinweis:** War die Schnittstelle zuvor schon geöffnet, so wird sie zunächst mit I332IFCDone geschlossen und danach neu geöffnet

## Demo

**procedure** TfrmI332.rbDemoClick

Die ausgewählte Demo-Funktion wird eingeblendet; die einzelnen Demo-Funktionen befinden sich als TFrame-Objekte in eigenen Dateien fm\_I332XXX.PAS.

## Anzeige

**procedure** TfrmI332.Timer1Timer

Die Anzeige der Achs-Positionen, der Bahngeschwindigkeit und einiger Statusbits erfolgt über einen Systemtimer. Innerhalb der Timer-Routine werden regelmäßig die Funktionen I332GetStatus, I332VIst und I332Pos aufgerufen.

Im Aufklappfeld oberhalb der Achsanzeige können alternativ

- Sollposition

- Istposition
  - Schleppfehler
  - Achs-Vorschub
- zur Anzeige ausgewählt werden.

<b>Poti / Override</b>	Ist ein Poti angeschlossen, so wird hier die aktuelle Potistellung angezeigt.
<b>Start</b>	<b>procedure</b> TfrmI332.btnStartClick
<b>Stop</b>	<b>procedure</b> TfrmI332.btnStopClick Mit den beiden Buttons kann eine zuvor programmierte Bewegung gestartet und gestoppt werden; benutzt werden die DLL-Funktionen I332ZyklusStart und I332ZyklusStop.
<b>Restart ausführen</b>	<b>procedure</b> TfrmI332.btnRestartClick Schreibt den Befehl im Auswahlfeld „Restart“ ausgewählten Restart-Befehl an die I332; Details zu den Restart-Befehlen s. im I332-Handbuch
<b>Config</b>	<b>procedure</b> TfrmI332.btnCfgDlgClick Ruft über die DLL-Funktion I332ExecCfgDlg den Konfigurationsdialog zur Einstellung von Achs- und Maschinendaten auf. Die Daten werden in der Datei I332.INI gespeichert.
<b>Test</b>	<b>procedure</b> TfrmI332.btnTstDlgClick Ruft über die DLL-Funktion I332ExecTstDlg den Testdialog zur Anzeige der IO-Ports, AD-Eingänge, DA-Ausgänge und Drehgeber auf.
<b>Abbrechen</b>	Das Demo-Programm wird beendet.

## 4.1 Die Demo-Funktionen

Die Datei fm\_I332.PAS enthält die Basisklasse für die einzelnen Demo-Frames; alle anderen sind hiervon abgeleitet. Die Vorlage enthält einige Routinen, die von den abgeleiteten Frames benutzt werden.

```
FUNCTION TfmI332.FindAxComp (VAR AxComp: TComponent; TAxComp:
TComponentClass; BaseName: String; AchsNr: INTEGER): BOOLEAN;
```

Findet eine Fenster-Komponente mit dem Name BaseName<AchsNr>, wenn sie existiert. Diese Funktion wird aufgerufen, um Daten aus den verschiedenen achsbezogenen Eingabefeldern auszulesen. Die Namensgebung für alle achsbezogenen Eingabe-Elemente folgt dabei der Konvention <Name><AchsNr>.

```
FUNCTION TfmI332.DispIFCerr(IFCerr: LONGINT): LONGINT;
```

Liest zur Fehlernummer IFCerr die Klartext-Meldung aus der I332.DLL und schreibt sie in das Label lblIFCerr.

**Abbrechen** Die Demo-Funktion wird beendet; dabei wird der Destructor des Frames aufgerufen, der auf jeden Fall einen Restart-Befehl ausführt:

```
destructor TfmI332.Destroy;
BEGIN
    I332Restart(I332Handle,icRestart3);
    inherited Destroy;
END;
```

Anschließend befindet man sich wieder im Hauptfenster.

Die Basisklasse implementiert außerdem die Funktion DoDisp, die von der Anzeige-Routine des Hauptfensters aufgerufen wird. Einige abgeleitete Frames überschreiben diese Routine, um eigene Elemente zu aktualisieren.

Einige abgeleitete Frames überschreiben außerdem den Constructor, um eigene Initialisierungen durchzuführen; insb. lesen einige Funktionen die Achsbezeichnungen aus der DLL um Bedien-Elemente korrekt zu beschriften. Für die Funktionalität der jeweiligen Demo-Funktion ist dies aber nicht von Belang. Beispielhaft der Constructor aus der Referenz-Demo:

```
constructor TfmI332Ref.Create(AOwner: TComponent);
VAR AxLbIs: TAxLbIs;
    i: INTEGER;
    btnRef: TSpeedButton;
BEGIN
    // ererbten constructor aufrufen:
    inherited Create(AOwner);

    // Achsbezeichnungen aus der DLL auslesen:
    I332GetAxLabels(I332Handle,AxLbIs);

    // Komponenten mit dem Namen "btnRef0"... "btnRef7"
    // suchen und korrekt beschriften:
    FOR i := 0 TO Axes-1 DO BEGIN
        IF FindAxComp(TComponent(btnRef),TSpeedButton,'btnRef',i)
        THEN btnRef.Caption := '&'+AxLbIs[i];
    END;
END;
```

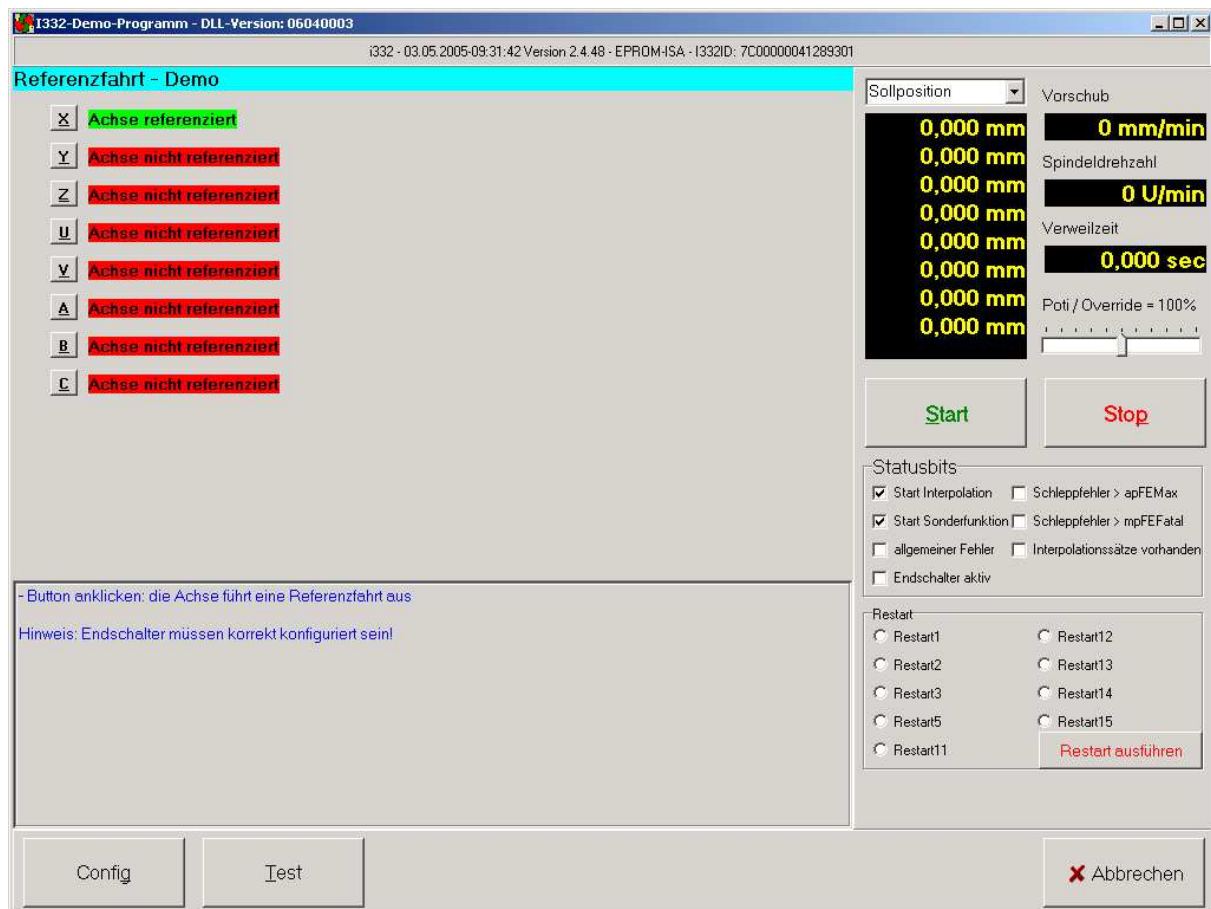
Alle Demo-Frames enthalten außerdem einen Bereich, in dem die Funktion / Bedienung kurz erklärt wird und ggf. weitere Hinweise gegeben werden.

### ***Demo-Programm I332Demo.EXE***

Die Demo-Funktionen sind fast alle so aufgebaut, daß beim Auslösen der Funktion eine Routine aufgerufen wird, die zunächst die relevanten Parameter aus den Eingabe-Elementen ausliest und danach mit diesen Parametern die entsprechende Funktion startet, entweder als DLL-Aufruf, wenn eine entsprechende Funktion zur Verfügung steht, oder als Unterprogramm innerhalb der Funktion bestehend aus einzelnen I332Read../I332Write..-Aufrufen.

## 4.2 Referenzfahrt – Demo

Datei: fm\_I332Ref.PAS



Beim Klicken auf einen der Buttons mit den Achsbezeichnungen wird die Routine `btnRefClick` aufgerufen:

```
procedure TfmI332Ref.btnRefClick(Sender: TObject);
VAR Axis: INTEGER;
begin
  try
    // Achsnr. = letzter Buchstabe der Button-Bezeichnung:
    Axis := StrToInt(TSpeedButton(Sender).Name[Length(Name)]);

    // Referenzfahrt für diese Achse starten:
    I332DoAxisRef(I332Handle, Axis);

    // Startbit setzen:
    I332ZyklusStart(I332Handle);
  except
    end;
end;
```

Außerdem wird die Routine `DoDisp` implementiert, um den Referenz-Status der einzelnen Achsen zu aktualisieren:

```
PROCEDURE TfmI332Ref.DoDisp;
VAR refInfo: WORD;
    i: INTEGER;
```



## **Demo-Programm I332Demo.EXE**

```
    lblRef: TLabel;  
BEGIN  
    // Referenzstatus lesen;  
    I332GetRefInfo(I332Handle,refInfo);  
  
    FOR i := 0 TO Axes-1 DO BEGIN  
  
        // Label für die Achse i finden:  
        IF FindAxComp(TComponent(lblRef),TLabel,'lblRef',i)  
        THEN BEGIN  
  
            // Beschriftung und Farbe entspr. anpassen:  
            IF (refInfo and (1 shl i) <> 0) THEN BEGIN  
                lblRef.Caption := 'Achse referenziert';  
                lblRef.Color := clLime;  
            END ELSE BEGIN  
                lblRef.Caption := 'Achse nicht referenziert';  
                lblRef.Color := clRed;  
            END;  
  
        END;  
    END;  
END;
```

## 4.3 Handbetrieb – Demo

Datei: fm\_I332Hnd.PAS

<b>Achse</b>	Achsauswahl
<b>Geschwindigkeit</b>	Geschwindigkeit wählen
<b>Modus</b>	Geschwindigkeitsangabe in % bzw. mm/min
<b>Poti aus</b>	Poti (falls vorhanden) wird nicht berücksichtigt
<b>Endschalterfehler nicht melden</b>	Die normale Endschalterbehandlung (→ Anhalten, Fehlermeldung, Restart-Befehl) wird abgeschaltet. Wird ein Endschalter erreicht, so wird nur angehalten und diese Verfahrrichtung gesperrt. Wegfahren vom Endschalter ist möglich.
<b>+ / - Buttons</b>	Die Achse wird solange in der angeklickten Richtung verfahren, bis der Button wieder losgelassen wird.

Wird einer der Buttons angeklickt, so wird folgende Routine aufgerufen:

```

PROCEDURE TfmI332Hnd.DoSFHand(Rtg: INTEGER);
VAR sfMode: WORD;
    V: INTEGER;
BEGIN
    // Daten aus den einzelnen Eingabe-Elementen einsammeln:

```

## **Demo-Programm I332Demo.EXE**

```
// Geschwindigkeit:
try V := StrToInt(edtFeed.Text); except exit; end;
V := V*Rtg;

// Modus:
CASE rgVMode.ItemIndex OF
  0: sfMode := sfVMax; // in %
  1: sfMode := sfVCmd; // in mm/min
  ELSE Exit;
END;

IF cbPotiOff.Checked THEN sfMode := sfMode or sfPotiOff;
IF cbLSOff.Checked THEN sfMode := sfMode or sfLSOff;

// Sonderfunktion setzen:
I332DoAxisHand(I332Handle,cbAchsen.ItemIndex,V,sfMode);

// Startbits setzen:
I332ZyklusStart(I332Handle);
END;
```

Beim Loslassen wird die Sonderfunktion Handbetrieb abgebrochen:

```
// Funktion abbrechen
I332DoAxisHand(I332Handle,cbAchsen.ItemIndex,0,sfBreak);
// Startbits zurücksetzen:
I332ZyklusStop(I332Handle);
```

## 4.4 Interpolation – Demo

Datei: fm\_I332IPL.PAS

The screenshot shows the 'Interpolation - Demo' window of the I332-Demo-Program. The interface includes several sections:

- Positionen (µm):** Input fields for X, Y, Z, U, V, A, B, and C. A checkbox for 'Absolut' is present.
- Vorschub (mm/min):** Input field for feed rate.
- Interpolationsmodus:** Radio buttons for 'Linear-Interpolation', 'Kreis - UZS', and 'Kreis - GZS'.
- Kreisinterpolation: Mittelpunkt-Koordinaten rel. zum Startpunkt:** Input fields for 'Mittelpunkt "X"' and 'Mittelpunkt "Y"', and radio buttons for 'Kreisene - "X"-Achse' and 'Kreisene - "Y"-Achse'.
- Statusbits:** Checkboxes for 'Start Interpolation', 'Start Sonderfunktion', 'allgemeiner Fehler', 'Endschalter aktiv', 'Schleppfehler > apFEMax', and 'Schleppfehler > mpFEFatal'.
- Restart:** Radio buttons for 'Restart1' through 'Restart15' and a 'Restart ausführen' button.
- Buttons:** 'Start', 'Stop', 'Ausführen', 'Config', 'Test', and 'Abbrechen'.

**Positionen** Zielkoordinaten; Achsen mit leeren Feldern werden nicht verfahren.

**Absolut** Zielkoordinaten sind Absolut-Position bzw. Distanzen zum Zielpunkt

**Vorschub** Vorschub eingeben

**Interpolationsmodus** Auswahl Linear-Interpolation oder Kreis-Interpolation im UZS oder GZS

Wird eine Kreis-Interpolation ausgewählt, müssen anschließend die

**Mittelpunkt-Koordinaten** relativ zum Startpunkt

eingetragen werden.

**Kreisebene** Die ausgewählten Achsen legen die beiden Kreis-Achsen fest; jede beliebige Achs-Kombination aus zwei verschiedenen Achsen ist zugelassen!

**Button Ausführen** Die Daten werden als Interpolationssatz an die I332 übergeben. Verfahren werden sie allerdings erst, wenn der Start-Button angeklickt wird!

Der Button „Ausführen“ ruft folgende Routine auf:

```
procedure TfmI332IPL.btnIPLClick(Sender: TObject);
VAR AxisArray: TAxisArray;
    i: INTEGER;
    edtPos: TLabelEdit;
```

## Demo-Programm I332Demo.EXE

```
F: INTEGER;
Flags: INTEGER;
s: String;
mpx, mpy: INTEGER;
IFCerr: INTEGER;
begin
  // Positionsdaten aller Achsen einsammeln:
  FOR i := 0 TO Axes-1 DO BEGIN
    AxisArray[i] := UndefL; // Achsposition undefiniert
    IF FindAxComp(TComponent(edtPos), TLabelEdit, 'edtPos', i)
    THEN BEGIN
      s := Trim(edtPos.Text);
      IF s = '' THEN Continue; // leere Felder ignorieren
                                // Achsposition weiterhin undefiniert:
                                // --> wird nicht verfahren
      AxisArray[i] := StrToInt(s);
    END;
  END;

  // Vorschub:
  F := StrToInt(edtFeed.Text);

  // relativ/absolut:
  IF cbAbs.Checked THEN Flags := fAbs ELSE Flags := fRel;

  IFCerr := 0;
  IF rgIPLMode.ItemIndex = 0 THEN BEGIN // Linear-Interpolation
    IFCerr := I332DoIPLLin(I332Handle,
                          Axes, AxisArray,
                          F, Flags);
  END ELSE BEGIN // Kreis-Interpolation
    // Mittelpunktdaten:
    mpx := StrToInt(Trim(edtMPX.Text));
    mpy := StrToInt(Trim(edtMPY.Text));
    IF rgIPLMode.ItemIndex = 1 THEN BEGIN // Kreis-Interpolation UZS
      IFCerr := I332DoIPLCW(I332Handle,
                            Axes, AxisArray,
                            F, Flags,
                            rgX.ItemIndex, rgY.ItemIndex, mpx, mpy);
    END ELSE BEGIN // Kreis-Interpolation GZS
      IFCerr := I332DoIPLCCW(I332Handle,
                             Axes, AxisArray,
                             F, Flags,
                             rgX.ItemIndex, rgY.ItemIndex, mpx, mpy);
    END;
  END;

  // ggf. Fehler anzeigen:
  DispIFCerr(IFCerr);
end;
```

## 4.5 Spindel + Verweilzeit – Demo

Datei: fm\_I332SP.PAS

I332-Demo-Programm - DLL-Version: 06040003  
i332 - 03.05.2005-09:31:42 Version 2.4.48 - EPROM-HSA - I332ID: 7C00000041289301

### Spindel + Verweilzeit - Demo

Verweilzeit (ms)

Spindeldrehzahl (U/min)

Spindel-Ausgänge setzen:

am Satzanfang	am SatzEnde
<input checked="" type="radio"/> unverändert	<input checked="" type="radio"/> unverändert
<input type="radio"/> Spindel aus (M05)	<input type="radio"/> Spindel aus (M05)
<input type="radio"/> Spindel ein UZS (M03)	<input type="radio"/> Spindel ein UZS (M03)
<input type="radio"/> Spindel ein GZS (M04)	<input type="radio"/> Spindel ein GZS (M04)

Sollposition

Vorschub

Spindeldrehzahl

Verweilzeit

Poti / Override = 100%

Statusbits

☐ Start Interpolation ☐ Schleppfehler > apFEMax

☐ Start Sonderfunktion ☐ Schleppfehler > mpFEFatal

☐ allgemeiner Fehler ☐ Interpolationssätze vorhanden

☐ Endschalter aktiv

Restart

☐ Restart1 ☐ Restart12

☐ Restart2 ☐ Restart13

☐ Restart3 ☐ Restart14

☐ Restart5 ☐ Restart15

☐ Restart11 ☐ Restart15

Damit dieses Beispiel funktioniert ist folgendes zu beachten:

- Die Spindel muß im Konfigurationsdialog korrekt definiert werden
- für die Spindel-Ausgänge wird vom Konfig-DIALOG das entsprechende Bit im Register mpXXXOutPAR = 0 gesetzt; dies sollte nicht geändert werden!!

sonstiges (s. Quelltext!):

- Drehzahl ist modal, d.h. muß nur einmal übergeben werden
- Drehzahl wird IMMER am Satzanfang gesetzt
- die Spindel-Ausgänge (M03/M04/M05) dagegen je nach Programmierung:

## 4.6 Asynchron fahren – Demo

Datei: fm\_I332AS.PAS

I332-Demo-Programm - DLL-Version: 06040003  
i332 - 03.05.2005-09:31:42 Version 2.4.48 - EPROM-HSA - I332ID: 7C00000041289301

### Asynchron fahren – Demo

Positionen (µm):	Vorschübe:	Modus:	Start
X 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
Y 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
Z 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
U 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
V 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
A 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
B 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus
C 0	<input type="checkbox"/> Abs 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input type="checkbox"/> Poti aus

Sollposition: 0,000 mm  
Vorschub: 0 mm/min  
Spindeldrehzahl: 0 U/min  
Verweilzeit: 0,000 sec  
Poti / Override = 100%

**Start** **Stop**

Statusbits

- ☐ Start Interpolation
- ☐ Start Sonderfunktion
- ☐ allgemeiner Fehler
- ☐ Endschalter aktiv
- ☐ Schleppfehler > apFEMax
- ☐ Schleppfehler > mpFEFatal
- ☐ Interpolationssätze vorhanden

Restart

- ☐ Restart1
- ☐ Restart2
- ☐ Restart3
- ☐ Restart5
- ☐ Restart11
- ☐ Restart12
- ☐ Restart13
- ☐ Restart14
- ☐ Restart15

**Restart ausführen**

- Position angeben  
- Abs: Position = Absolutposition  
- Vorschub angeben  
- Modus: Vorschub in % / Vorschub in mm/min  
- ES aus: am Endschalter anhalten, aber keine Fehlermeldung  
- Poti aus: Poti wird nicht berücksichtigt  
- Start-Button klicken

Config Test **Abbrechen**

## 4.7 Joystick – Demo

Datei: fm\_I332JS.PAS

I332-Demo-Programm - DLL-Version: 06040003  
i332 - 03.05.2005-09:31:42 Version 2.4.48 - EPROM-HSA - I332ID: 7C00000041289301

### Joystick - Demo

Vorschub:	Modus:	Optionen:	JS-Eingang
X 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
Y 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
Z 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
U 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
V 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
A 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
B 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %
C 100	<input checked="" type="radio"/> in % <input type="radio"/> in mm/min	<input type="checkbox"/> ES aus <input checked="" type="checkbox"/> Poti aus	-1 0.0 %

Sollposition: 0,000 mm

Vorschub: 0 mm/min

Spindeldrehzahl: 0 U/min

Verweilzeit: 0,000 sec

Poti / Override = 100%

**Start** **Stop**

Statusbits

☐ Start Interpolation ☐ Schleppfehler > apFEMax

☐ Start Sonderfunktion ☐ Schleppfehler > mpFEFatal

☐ allgemeiner Fehler ☐ Interpolationssätze vorhanden

☐ Endscharter aktiv

Restart

☐ Restart1 ☐ Restart12

☐ Restart2 ☐ Restart13

☐ Restart3 ☐ Restart14

☐ Restart5 ☐ Restart15

☐ Restart11 ☐ Restart15

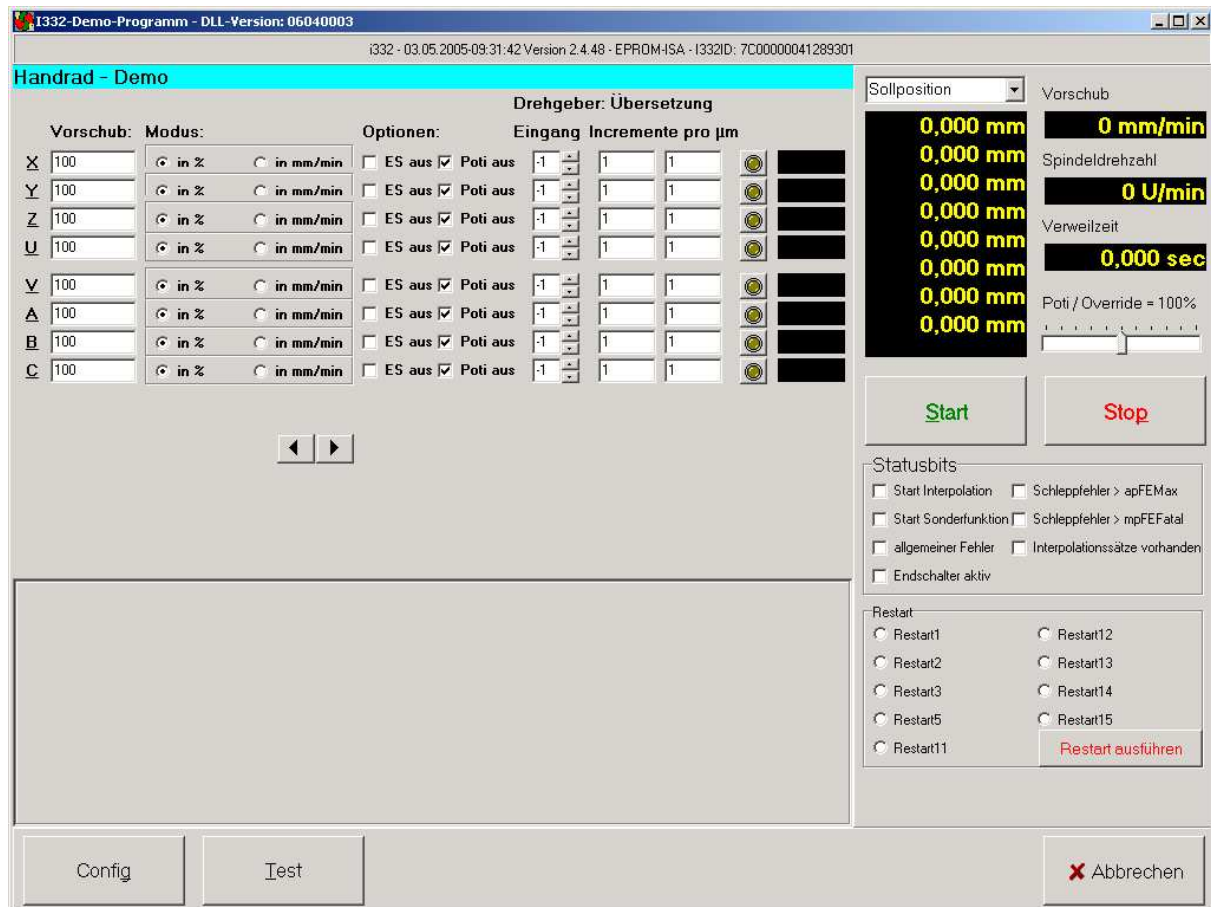
**Restart ausführen**

Config Test **X Abbrechen**



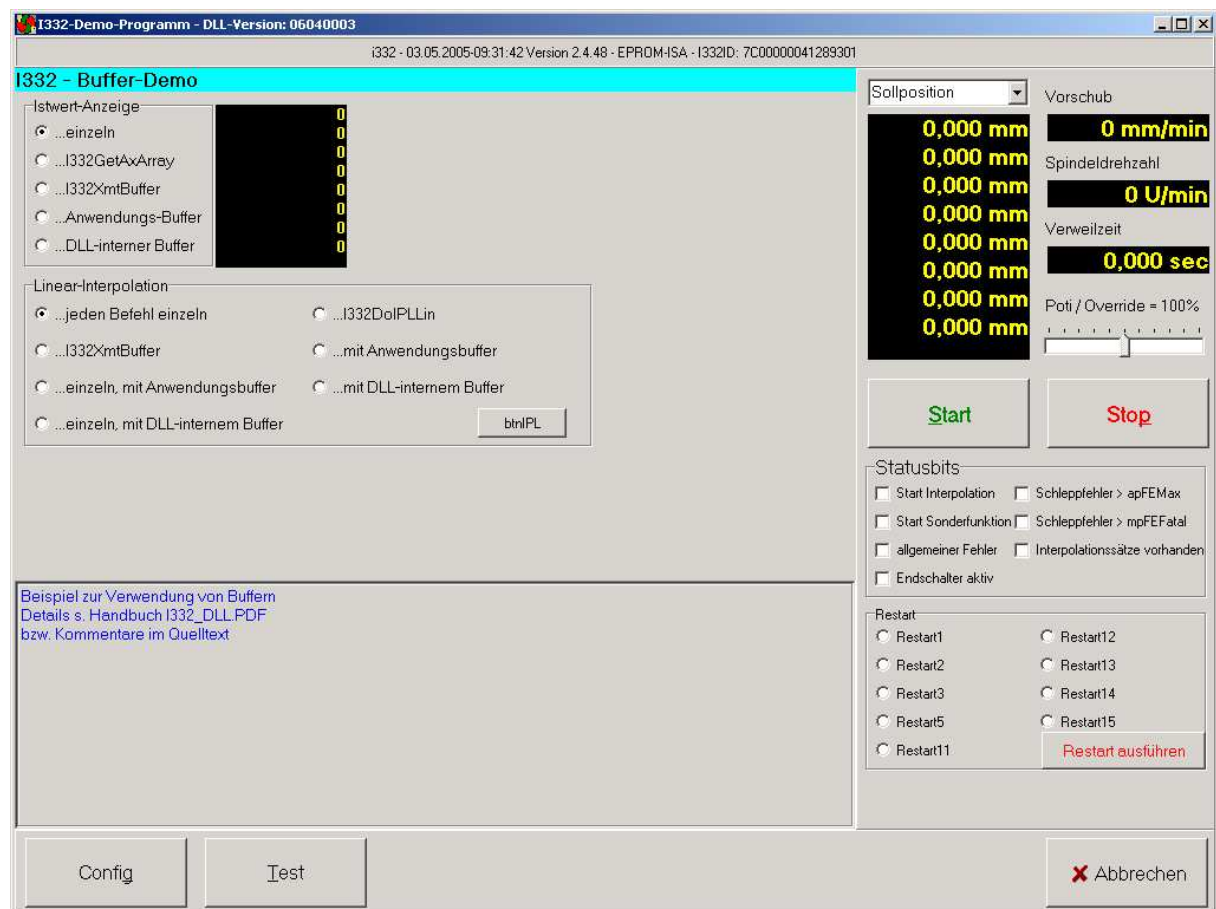
## 4.8 Handrad – Demo

Datei: fm\_l332HR.PAS



## 4.9 I332-Buffer – Demo

Datei: fm\_I332Buffer.PAS



Dieses Beispiel demonstriert die Verwendung von Buffern. Dazu werden im Constructor des Frames global die notwendigen Buffer-Handles bereitgestellt und im Destructor wieder freigegeben.

Die Demo enthält zwei Beispiele:

1. Positionsanzeige mit verschiedenen Varianten
2. Linearinterpolation mit verschiedenen Varianten

Details dazu s. Kommentare im Quelltext