Notizen und Übersichten zu

umFish30.DLL

Ulrich Müller

Inhalt

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873, Fax 05251/55709
eMail : UM@ftComputing.de

HomePage : www.ftComputing.de

Dokument : umFish30Notes.doc, Druckdatum : 12.05.2004

Überblick der umFish30.DLL Zugriffsfunktionen

umFish30.DLL selber bietet zwei gleichwertige Sätze von Zugriffsfunktionen :

- die **um-Variante** mit direktem Zugriff auf den verwendeten Kontrollblock
- und die **cs-Variante**, die die erforderlichen Zugriffe auf den Kontrollblock kapselt

Die um-Variante ist flexibler, die cs-Variante kann auch mit Sprachen eingesetzt werden, die mit einer Struktur (dem Kontrollblock) als Parameter Probleme haben.

Für verschiedene Sprachen werden einfache **Deklarationen** für den Zugriff auf die um-Schnittstelle angeboten :

- **Visual Basic** : umFish.BAS
- **VC++** : umFish30VC.H und umFish30.LIB (für VC++6.0, für kleinere Versionen : umFish30Load.H)
- **C++Builder** : umFish30Load.H (dynamisches Laden der DLL)
- **Delphi** : umFish30.PAS

Außerdem werden werden Klassen angeboten, die den Funktionsumfang der umFish30.DLL-Funktionen deutlich erweitern :

- **Visual Basic** : FishFa30 und weitere. In Paket vbFish30Setup.EXE enthalten.
- **C++Builder** : FishFa30.H/CPP
- **Delphi** : FishFa30.PAS. In Paket delphiFish30.Setup.EXE enthalten.
- **C#** : FishFa30.CS
- **VB.NET** : FishFa30.DLL (die Assembly zu FishFa30.CS)

Die Klassen bieten über die Sprachen einen einheitlichen Funktionsumfang an, die sprachbedingten Unterschiede werden, soweit es geht, ausgeglichen. Einheitlich werden die Funktionen NotHalt und ESC-Taste zum Abbruch laufender Methoden angeboten (umFish30.DLL) tut das nicht). Die zugehörenden Files sind in umFish30.ZIP zusammengefasst (Ausnahme VB und Delphi FishFa30 s.o.).

Im Nachfolgenden werden deren Funktionen im Einzelnen beschrieben.

Zugriff auf das Interface

Der Zugriff auf das Interface erfolgt indirekt über eine Poll-Routine, die in regelmäßigen Abständen die Werte des Interface ausliest und gleichzeitig den Status der M-Ausgänge setzt (schaltet). Damit sind auch die Refresh-Bedingungen (ca. alle 300 mSek ein Zugriff) erfüllt, ein Abschalten des Interfaces erfolgt nicht mehr. Eine konstante Zeitbasis (typisch : 10 mSek) wird durch den MultiMediaTimer des Systems gewährleistet, der die PollRoutine in einem eigenen Thread betreibt.

Die ausgelesenen Werte werden in einem Kontrollblock abgestellt bzw.die Werte für die M-Ausgänge werden dort entnommen. Der Kontrollblock enthält darüberhinaus alle Werte die für den Betrieb eines Interfaces (mit Slave) erforderlich sind. Ein Parallel-Betrieb mehrerer Interfaces (z.B. eins an LPT, ein weiteres an COM1) ist somit möglich.

Die Poll-Routine erledigt über den reinen Verkehr mit dem Interface hinaus noch weitere Aufgaben. Das sind die Zählung der Impulse an den E-Eingängen (Veränderung am true/false-Status eines Einganges), die Geschwindigkeitssteuerung (durch zyklisches Ein/Ausschalten der M-Ausgänge) und im RobMode das Abschalten eines M-Ausganges, wenn der zugehörende Impuls-Counter den Wert null erreicht hat. Kurz vor Erreichen des Wertes null wird der M-Ausgang "gebremst".

Die angebotenen Zugriffsfunktionen sind ein Mix aus Notwendigkeit und Komfort. Open/CloseInterface stellen die Verbindung zum Interface her, setzen default-Parameter, starten den MultiMediaTimer und beenden die Verbindung wieder. Die GetInput-Funktion liest lediglich den Wert für einen E-Ausgang aus dem Kontrollblock. Dabei wird das zutreffend bit maskiert, ähnliches gilt für SetMotor und SetLamp in der Gegenrichtung. Es erfolgt auch hier keine direkte Ansteuerung des Interfaces.

Das tut dagegen die Funktion GetAnalogDirect, die für die Dauer eines (direkten) Zugriff auf einen Analog-Eingang die Poll-Routine abschaltet. Grund : besonders das Auslesen der Analog-Eingänge des parallelen Interfaces dauert wesentlich länger als das Lesen/Schreiben der E-Eingänge/M-Ausgänge. So kann das PollInterval auf die Bedürfnisse des Motorbetriebes eingestellt werden und ein gelegentliches Lesen der Analog-Eingänge (typischerweise bei Stillstand der Motoren) ermöglicht werden.

SetMotor(s)Ex und RobMotor(s) arbeiten nur mit dem Kontrollblock zusammen, führen aber (über das reine Setzen der M-Ausgänge hinaus) etwas komplexere Operationen aus.

Die Komfort-Funktionen und weitere Operationen auf den Kontrollblock können auch durch die Anwendung direkt vorgenommen werden.

Da manche Programmiersprachen ein Problem mit der Übergabe (besonders auch der dauerhaften) einer Struktur an die DLL haben, werden zwei Version von Zugriffs-Funktionen angeboten : die um-Variante mit dem Kontrollblock als Parameter und die cs-Variante mit einem in der DLL gekapselten Kontrollblock, dessen Handle dann als Parameter dient. Da die cs-Variante keinen eigenen Zugriff auf den Kontrollblock mehr zuläßt, bietet sie einen deutlich größeren Funktionsumfang.

umFish30.DLL v3.0 : Deklarationen und Funktionen.

Hier wird eine Darstellung in einer C/C++ Notation verwendet. Für die einzelnen Sprachen werden sprachspezifische Deklarationsfile, je nach Gegebenheiten in der um- oder der cs-Variante in umFish30.ZIP mitgeliefert. Die Funktionen sind deckungsgleich. Zusätzlich enthält umFish30.ZIP jweils auch ein kleines Beispielprogramm.

um-Variante

Version unter Verwendung eines extern geführten Kontrollblocks in dem die für den Betrieb von umFish30.DLL benötigten Werte gehalten und ständig aktualisiert werden. Zusätzlich eine (kleine) Zahl von Funktionen zum Betrieb der DLL (umOpen/CloseInterface ...) sowie (Komfort)Funktionen für den Zugriff auf den Kontrollblock (umGetInput, umSetMotor ...).

Die um-Variante ist die flexibelste Lösung, da direkt auf alle Betriebsparameter zugegriffen werden kann. Kann aber nur bei Programmiersprachen eingesetzt werden, die Schnittstellen zu Strukturen anbieten.

typedef struct {

HANDLE **hCom**	Handle zum COM- bzw. LPT-Port, wird bei erfolgreichem Open gesetzt und bei close auf 0 gesetzt. Sollte nicht verändert werden.
DWORD **PortID**	nur LPT, I/O-Adresse des LPT-Ports, von OpenInterface gesetzt.
DWORD **LPTDelay**	Ausgabeverzögerung an dem LPT-Port. Von OpenInterface auf default = 10 gesetzt, kann ggf. vor dem Open verändert werden. Schnelle Rechner : großerer Wert, nur für LPT1-3.
DWORD **LPTAnalog**	Skalierung des Analogwertes, von OpenInterface auf default = 5 gesetzt, kann ggf. vor dem Open verändert werden. Der angezeigte Wertebereich sollte zwischen 0 und 1024 liegen, bei zu kleinen Werten größeres LPTAnalog wählen (LPT1-3). Bei LPT wird der Meßwert auf LPTAnalog begrenzt, Bei COM nicht ausgewertet.
DWORD **Ecount**	Anzahl E-Eingänge (default = 8 (ohne Slave), = 16 mit Slave)
DWORD **FID**	Handle MultiMediaTimer. sollte nicht geändert werden.
DWORD **PollInterval**	Interval des MM-Timers in MilliSekunden. Defaultwerte wird von OpenInterface gesetzt (wenn ungleich 0 oder zu klein). COM : ohne Slave = 10, mit 12, AnalogScan = 16 LPT : 1, AnalogScan = 100. Achtung : bei zu kleinen Werten kann sich der Rechner aufhängen.
DWORD **AnalogScan**	Abfrage auch der Analogeingänge (default = 0, mit = 1).
DWORD **OutputStatus**	Aktueller Status aller M-Ausgänge. Wird intern verwendet.
DWORD **MotorStatus**	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = ftiAus, 01 = ftiLinks, 10 = ftiRechts).
DWORD **SpeedStatus**	Status der Geschwindigkeiten aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 – 1111 (full).
DWORD **ModeStatus**	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 normal, 0001 RobMode.
DWORD **intern**	---
DWORD **InputStatus**	Status aller E-Eingänge. Jeweils 1 bit pro Eingang. Begonnen bei 0 für E1 (1 = true, 0 = false).
DWORD **Analogs**[2]	Werte der A-Eingänge (0 = EX, 1 = EY)
DWORD **Counters**[16]	Impulszähler an den E-Eingängen.

} ftiDCB;

Der Kontrollblock wird im Anwendungsprogramm als Struktur mit 32bit-Worten dargestellt. Ein simultaner Einsatz mehrerer Interfaces ist bei Verwendung entsprechend vieler Kontrollblöcke möglich. Ein simultaner Betrieb von zwei "LPT" (nicht LPT1..LPT3) Interfaces ist bedingt möglich, siehe Hinweis in umFish30.TXT.

Verwendete Variablenbezeichnungen

AnalogNr	Nummer eines Analog-Einganges (EX = 0, EY = 1)
Direction	Drehrichtung eines Motors (00 = ftiAus, 01 = ftiLinks, 10 = ftiRechts)
ICount	Positionsangabe in Anzahl Impulsen gerechnet von der aktuellen Position. Absoluter Wert, die Richtung wird durch Direction bestimmt.
InputNr	Nummer eines E-Einganges (1 – 8(16))
LampNr	Nummer eines "halben"-M-Ausganges (1-8(16))
MotorNr	Nummer eines M-Ausganges (1-4(8))
MotorStatus	SollStatus aller M-Ausgänge, korrespondiert mit dem ftiDCB-Wert
OnOff	Anstelle von true/false (1-0)
PortName	Namen der wählbaren Ports (LPT, COM1-8, LPT1-3)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (0-15(full))
SpeedStatus	Geschwindigkeit aller M-Ausgänge, korrespondiert mit dem ftiDCB-Wert.

Alle Variablen sind vom Typ DWORD (32bit, ohne Vorzeichen).

Funktionen

DWORD **umOpenInterface**(ftiDCB &Interface, LPCSTR PortName)
Setzen von Parametern, Herstellen der Verbindung zum Interface

DWORD **umCloseInterface**(ftiDCB &Interface)
Schließen der Verbindung zum Interface

DWORD **umGetVersion**()
Lesen der Version von umFish30.DLL

DWORD **umGetAnalogDirect**(ftiDCB &Interface, DWORD AnalogNr);
Direktes Lesen Analog EX/EY, dazu wird das Pollen vorübergehend abgeschaltet. Sinnvoll nur, wenn die Motoren stehen.

BOOL **umGetInput**(ftiDCB &Interface, DWORD InputNr)
Auslesen des Wertes des angegebenen E-Ausganges

DWORD **umSetMotor**(ftiDCB &Interface, DWORD MotorNr, DWORD Direction)
Setzen eines M-Ausganges

DWORD **umSetMotorEx**(ftiDCB &Interface, DWORD MotorNr,
DWORD Direction, DWORD Speed)
Setzen eines M-Ausganges einschl. Drehzahlstufe

DWORD **umSetMotors**(ftiDCB &Interface, DWORD MotorStatus)
Setzen aller M-Ausgänge

DWORD **umSetMotorsEx**(ftiDCB &Interface, DWORD MotorStatus, DWORD SpeedStatus)
Setzen aller M-Ausgänge einschl. Drehzahlstufe
Zu MotorStatus und SpeedStatus siehe auch die Anmerkungen zu den Rob-Funktionen.

DWORD **umSetLamp**(ftiDCB &Interface, DWORD LampNr, DWORD OnOff)
Setzen eines "halben"-M-Ausganges

DWORD **umRobMotor**(ftiDCB &Interface, DWORD MotorNr, DWORD Direction,
DWORD Speed, DWORD ICount)
Setzen eines M-Ausganges im RobMode

DWORD **umRobMotors**(ftiDCB &Interface, DWORD MotorStatus, DWORD SpeedStatus,
DWORD ModeStatus)
Setzen aller M-Ausgänge in wählbaren Mode
Vor dem Befehl sind die entsprechenden Counter zu setzen (fitDCB.Counter[c]).

DWORD **umStartDriver**()

DWORD **umStopDriver**()

cs-Variante

Version mit intern geführtem Kontrollblock (siehe auch um-Variante). umFish30.DLL führt intern einen Pool von Kontrollblöcken, deren jeweiliges Handle beim OpenInterface bestimmt wird (und beim CloseInterface wieder freigegeben wird). Es wird von allen weiteren Funktionen als Parameter benötigt. Die cs-Variante bietet eine größere Anzahl von Funktionen, da eine direkter Zugriff auf den Kontrollblock nicht möglich ist.

Die cs-Variante ist für den Einsatz mit Programmiersprachen gedacht, die keine Strukturen als Aufrufparameter unterstützen. Kann aber auch je nach Vorliebe in anderen Fällen genutzt werden. Ein Mischbetrieb ist jedoch nicht möglich.

Verwendete Variablenbezeichnungen

AnalogNr	Nummer eines Analog-Einganges (EX = 0, EY = 1)
CounterNr	Nummer eines ImpulsCounters (1-8(16))
Direction	Drehrichtung eines Motors (00 = ftiAus, 01 = ftiLinks, 10 = ftiRechts)
ICount	Positionsangabe in Anzahl Impulsen gerechnet von der aktuellen Position. Absoluter Wert, die Richtung wird durch Direction bestimmt.
iHandle	Handle zum internen Kontrollblock
InputNr	Nummer eines E-Einganges (1 – 8(16))
InputStatus	Status aller E-Eingänge. Jeweils 1 bit pro Eingang. Begonnen bei 0 für E1 (1 = true, 0 = false).
LampNr	Nummer eines "halben"-M-Ausganges (1-8(16))
Mode	BetriebsModus eines M-Ausganges (0 : normal, 1 : RobMode)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 normal, 0001 RobMode.
MotorNr	Nummer eines M-Ausganges (1-4(8))
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = ftiAus, 01 = ftiLinks, 10 = ftiRechts).
OnOff	Anstelle von true/false (1-0)
PortNr	Nummer der wählbaren Ports (0-11 : LPT, COM1-8, LPT1-3)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (0-15(full))
SpeedStatus	Status der Geschwindigkeiten aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 – 1111 (full).
Value	allgemeiner 32bit-Wert

Alle Variablen sind vom Typ int (32bit, mit Vorzeichen).

Funktionen

iHandle **csOpenInterface**(int PortNr, int AnalogScan, int Slave, int PollInterface);
Herstellen der Verbindung zum Interface, setzen von Parametern. Liefert ein Handle (oder ftiFehler), das bei den folgenden Funktionen zur Identifizierung verwendet werden muß.
AnalogScan = 0 ohne, = 1 mit Scannen Analogeingänge,
Slave = 0 ohne, = 1 mit Slave Modul,
PollInterval = 0 Bestimmung durch OpenInterface, > 0 einzusetzender Wert.

iHandle **csOpenInterfaceEx**(int PortNr, int AnalogScan, int Slave, int PollInterface, int LPTAnalog, int LPTDelay);
wie csOpenInterface aber mit zusätzlichen Parametern für das parallele Interface mit der direkten Schnittstelle LPT1 - 3
LPTAnalog = 0 Bestimmung durch OpenInterface, > 0 einzusetzender Wert,
LPTDelay = 0 Bestimmung durch OpenInterface, > 0 einzusetzender Wert.

int **csCloseInterface**(int iHandle);
Schließen der Verbindung zum Interface

int **csVersion**();
Ausgabe der umFish30.DLL Version

int **csGetAnalogScan**(int iHandle);
Lesen, ob auch die Analogeingänge gescannt werden sollen (0 = nein. 1 = ja)

int **csGetLPTAnalog**(int iHandle);
Lesen Analogskalierung

int **csGetLPTDelay**(int iHandle);
Lesen Ausgabeverzögerung

OnOff **csGetSlave**(int iHandle);
Lesen, ob Slave vorhanden

int **csGetPollInterval**(int iHandle);
Lesen PollInterval (Wert in MilliSekunden).

OnOff **csGetInput**(int iHandle, int InputNr);
Auslesen des Wertes des angegebenen Einganges

int **csGetInputs**(int iHandle);
Auslesen der Werte aller E-Eingänge

int **csGetAnalog**(int iHandle, int AnalogNr);
Auslesen des Wertes des angegebenen Analogeinganges

int **csGetAnalogDirect**(int iHandle, int AnalogNr);
Direktes Lesen Analog EX/EY, dazu wird das Pollen vorübergehend abgeschaltet. Sinnvoll nur, wenn die Motoren stehen.

int **csSetMotor**(int iHandle, int MotorNr, int Direction);
Setzen eines M-Ausganges

int **csSetMotorEx**(int iHandle, int MotorNr, int Direction, int Speed);
Setzen eines M-Ausganges einschl. Angabe der Drehzahlstufe

int **csGetMotors**(int iHandle);
Auslesen des Status aller M-Ausgänge

Mode **csGetModeStatus**(int iHandle, int MotorNr);
Auslesen des Modes eines M-Ausganges

void **csSetModeStatus**(int iHandle, int MotorNr, int Mode).
Setzen des Modes eines M-Ausganges

int **csSetMotors**(int iHandle, int MotorStatus);
Setzen des Status aller M-Ausgänge

int **csSetMotorsEx**(int iHandle, int MotorStatus, int SpeedStatus)
Setzen des Status aller M-Ausgänge einschl. Drehzahlstufe
Zu MotorStatus und SpeedStatus siehe auch die Anmerkungen zu den Rob-Funktionen.

int **csSetLamp**(int iHandle, int LampNr, int OnOff);
Setzen eines "halben"-M-Ausganges

int **csRobMotor**(int iHandle, int MotorNr, int Direction, int Speed, int ICount);
Setzen eines M-Ausganges im RobMode

int **csRobMotors**(int iHandle, int MotorStatus, int SpeedStatus, int ModeStatus);
Setzen aller M-Ausgänge in wählbarem Mode
Vor dem Befehl sind die entsprechenden Counter (csSetCounter) zu setzen.

int **csGetCounter**(int iHandle, int CounterNr);
Lesen eines ImpulsCounter-Standes

void **csSetCounter**(int iHandle, int CounterNr, int Value);
Setzen eines ImpulsCounter-Standes

void **csClearCounters**(int iHandle);
Löschen aller ImpulsCounter

Visual Basic 6

Für Visual Basis sind die schlichten Deklararationen des Files **umFish30.BAS** (in umFish30.ZIP enthalten) und die Klasse **FishFa30.CLS** (kompiliert : FishFa30.DLL) verfügbar. In umFish30.BAS wurde die um-Variante von umFish30.DLL verwendet, Beschreibung s.o. FishFa30.CLS ist in vbFish30Setup.EXE enthalten, dort auch mit einem ausführlichen Handbuch. Hier wird Vollständigkeitsgründen (besonders für den Autor) eine Übersicht der Funktionen gegeben.

FishFa30 mit Klasse FishFace

Allgemeine Anmerkungen zum Konzept

- mit Err.Raise 30001 (InterfaceProblem) und 30002 (KeinOpen), Source = "FishFace.methode", Text = "Interface Problem" / "Kein Open" in den meisten Methoden, es fallen daher oft die Return-Werte weg (also Sub statt Function). In der Anwendung sind jetzt On Error Goto ftiFehler ..... ftiFehler: erforderlich.
  Wichtigster Fall : OpenInterface
- keine Help Datei aber Handbuch im PDF-Format
- OpenInterface(PortName$, AnalogScan, Slave, PollInterval, LPTAnalog, LPTDelay). Parameter ab AnalogScan Optional. Grund Konstruktor hat keine Parameter.
- mit Enums ftiDir, ftiNr, ftiSpeed, ftiWait. Konstanten können auch ohen vorgestellte Enum-Namen verwendet werden, alternativ sind auch Long-Werte möglich.
- Die um-Variante von umFish30.DLL wird genutzt
- Anstelle von Überladung werden optionale Parameter mit Default-Werten (meist 0, auch ftiFehler) genutzt
- Counter zählen mit E-Eingängen von 1 To 16
- weitere Klassen sind in vbFish30Setup.EXE verfügbar

Enums

ftiDir	Angabe der Drehrichtung ...
ftiNr	Angabe der Ein-/Ausgangsnr
ftiSpeed	Geschwindigkeitsangabe
ftiWait	Return-Werte von WaitForMotors
	Parallel dazu können auch entsprechende numerische (int) Angaben gemacht werden.

Eigenschaften

bool **AnalogScan** (get)
Angabe ob auch die Analogeingänge gescannt werden sollen (default = False)

int **AnalogsEX**(get)
Lesen EX-Wert

int **AnalogsEY**(get)
Lesen EY-Wert

int **Inputs**(get)
Lesen der Werte aller E-Eingänge

int **LPTAnalog**(get)
Lesen Analogskalierung

int **LPTDelay**(get)
Lesen Ausgabeverzögerung

bool **NotHalt** (get/set)
Anmelden eines Abbruchwunsches (Default = false).

int **Outputs**(get)
Lesen Werte aller M-Ausgänge

int **PollInterval** (get)
Interval (in MilliSekunden) in dem der Status des Interfaces abgefragt(gepollt) und aufgefrischt (refresh) wird.

bool **Slave** (get)
Mit/ohne Extension Module

string **Version** (get)
Version der DLL

Verwendete Variablenbezeichnungen

Die Variablen haben weitgehend eine Enum als Typ, sie können aber genauso mit Longwerten versorgt werden. Hier werden zur Beschreibung des Wertebereichs einer Variablen die Enum-Namen angegeben.

AnalogNr	Nummer eines Analog-Einganges (ftiNr)
Analogwert	Rückgabewert beim Auslesen von EX / EY (0 – 1024)
Counter	Wert eines ImpulsCounters (Long)
Direction	Drehrichtung eines Motors (ftiDir)
InputNr	Nummer eines E-Einganges (ftiNr)
InputStatus	Rückgabewert beim Auslesen aller E-Eingänge (0 - &HFFFF)
LampNr	Nummer eines "halben"-M-Ausganges (ftiNr, M1 – M16)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge (Long). Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 normal, 0001 RobMode
MotorNr	Nummer eines M-Ausganges (ftiNr)
MotorStatus	SollStatus aller M-Ausgänge (Long). Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts).
mSek	Zeitangabe in MilliSekunden (Long)
NrOfChanges	Anzahl Impulse (Long)
OnOff	Ein/Ausschalten eines M-Ausganges (Boolean)
PortNr	Nummer der wählbaren Ports (String)
Position	Positionsangabe in Inpulsen (Long)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (ftiSpeed)
SpeedStatus	Status der Geschwindigkeiten aller M-Ausgänge (Long). Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1. Werte 0000 – 1111 (ftiFull).
TermInputNr	Nummer eines E-Einganges mit der die Methode beendet werden soll (ftiNr)
WaitWert	Rückgabewert von WaitForMotors (Long, ftiWait)
Value	allgemeiner Long-Wert

Die Aufrufparameter werden ByVal (Ausnahme WaitForPosition Counter) übergeben, Es gibt eine Reihe von optionalen Parametern, die teilweise den Ablauf der Methode starkt verändern.

Methoden

Sub **ClearCounter**(InputNr As ftiNr)
Löschen (0) des angegebenen Counters

Sub **ClearCounters**()
Löschen (0) aller Counter

Sub **ClearMotors**()
Abschalten aller M-Ausgänge

Sub **CloseInterface**()
Schließen der Verbindung zum Interface

Boolean **Finish**(InputNr As ftiNr)
Feststellen eines Endewunsches (NotHalt, Escape, InputNr(optional))

Analogwert **GetAnalog**(AnalogNr As ftiNr)
Auslesen der Werte von Ex / EY

Analogwert **GetAnalogDirect**(AnalogNr As ftiNr)
Direktes Auslesen der Werte von EX / EY. Dazu wird das Pollen vorübergehend abgeschaltet. Sinnvoll nur, wenn die Motoren stehen. Vorteil : Das PollInterval kann auf dem kleineren Wert von AnalogScan = False bleiben.

Long **GetCounter**(InputNr As ftiNr)
Auslesen des Wertes des angegebenen Counters

Boolean **GetInput**(InputNr As ftiNr)
Auslesen des Wertes des angegebenen E-Einganges

InputStatus **GetInputs**()
Auslesen der Werte aller E-Eingänge

Sub **OpenInterface**(PortName As String, AnalogScan As Boolean,
Slave As Boolean, PollInterval As Long,
LPTAnalog As Long, LPTDelay As Long)
Setzen von Parametern, Herstellen der Verbindung zum Interface. Mit Ausname von PortName sind die Parameter Optional, OpenInterface setzt dann default-Werte :
AnalogScan : default False
Slave : default False
PollInterval : default 0, es wird aber intern in Abhängigkeit von weiteren Parametern (LPT/COM, mit/ohne AnalogScan, Slave) ein praktikabler Wert gebildet, der nach dem OpenInterface mit der Property PollInterval abgefragt werden kann.
LPTAnalog, LPTDelay : default 0 intern wird dann modifiziert.

Sub **Pause**(mSek As Long)
Anhalten des Programmablauf für mSek MilliSekunden

Sub **SetCounter**(InputNr As ftiNr, Value As Long)
Setzen des Counters InputNr auf Value

Sub **SetLamp**(LampNr As ftiNr, OnOff As Boolean)
Setzen eines ‚halben' M-Ausganges (ein Pol an einem M-Ausgang und Masse).

Sub **SetMotor**(MotorNr As ftiNr, Direction As ftiDir, Speed As ftiSpeed,
Counter As ftiNr)

Setzen eines M-Ausganges, Optional mit Geschwindigkeitsangabe (Speed) und Angabe der zurückzulegenden Strecke in Impulsen (Counter))

Sub **SetMotors**(MotorStatus As Long, Speedstatus As Long, ModeStatus As Long)
Setzen des Status aller M-Ausgänge, Optional mit Geschwindigkeitsangabe (SpeedStatus) und des Betriebsmodes (ModeStatus, default = 0). Bei Betriebsmodus RobMode (1) sind vor dem Aufruf der Methode die entsprechenden Counter zu setzen (SetCounter[m])

Sub **WaitForChange**(InputNr As ftiNr, NrOfChanges As Long, TermInputNr As ftiNr)
Warten auf NrOfChanges Impulse an InputNr
oder TermInputNr(Optional) = True
alt -> LLWin-Kompatibilität

Sub **WaitForHigh**(InputNr As ftiNr)
Warten auf einen False/True-Durchgang an InputNr

Sub **WaitForInput**(InputNr As ftiNr, OnOff As Boolean)
Warten auf InputNr = OnOff. OnOff ist Optional (default = True)

Sub **WaitForLow**(InputNr As ftiNr)
Warten auf eine True/False-Durchgang an InputNr

WaitWert **WaitForMotors**(Time As Long, MotorNr As ftiNr ...)
Warten auf ein MotorReady-Ereignis oder Ablauf Time
Ein MotorReady-Ereignis wird durch SetMotor mit Parameter Counter bzw. ModeStatus (0001 ...) gestartet und tritt bei Counter = 0 aller MotorNr ein.
ftiTime = 0 : unbegrenztes Warten.
res = ftiWait.ftiEnde : alle Counter = 0 (auch durch Erreichen Endtaster gesetzt)
res = ftiWait.ftiTime : Ablauf Time
res = ftiWait.ftiNotHalt : Beendigung durch NotHalt.
res = ftiWait.ftiESC : Beendigung durch Escape-Taste
Bei ftiNotHalt und ftiESC werden außerdem alle betroffenen Motoren angehalten.

Sub **WaitForPositionDown**(InputNr As ftiNr, ByRef Counter As Long, Position As Long, TermInputNr As Long)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Herunterzählen der festgestellten Impulse an InputNr.
TermInputNr (Optional) = True beendet WaitForPosition vorzeitig
alt -> LLWin-Kompatibilität

Sub **WaitForPositionUp**(InputNr As ftiNr, Counter As Long, Position As Long, TermInputNr As ftiNr)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Heraufzählen der festgestellten Impulse an InputNr.
TermInputNr (Optional) = True beendet WaitForPosition vorzeitig
alt -> LLWin-Kompatibilität

Sub **WaitForTime**(mSek As Long)
Anhalten des Programmablauf für mSek MilliSekunden

Die Methoden erwarten ein vorhergehendes OpenInterface. Ggf. wird eine entsprechende **Exception** (Err.Raise) ausgelöst. Sie enthalten meist ein **DoEvents** um das Programm unterbrechbar zu machen. Wird im Ablauf ein InterfaceProblem festgestellt, wird eine entsprechende **Exception** ausgelöst. Die Wait-Methoden setzen bei Bedarf den zugehörenden **Counter** zurück.

Die SetMotor(s)-Methoden sind **asynchron** d.h. der oder die angesprochenen Motoren (Lampen) werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter. Sie werden durch ein weiteres SetMotors mit Direction = 0 beendet. Ausnahme : SetMotor mit Count-Parameter. Diese Methode beendet sich nach Erreichen der vorgegebenen Position selber.

Die Wait-Methoden koordinieren – meist in Verbindung mit End- bzw. ImpulsTastern den asynchronen Motorlauf mit dem Ablauf des Programms.Sie halten den weiteren Progammablauf an, bis das Wait-Ziel (Ablauf Zeit, erreichte Position, Tasterstellung ...) erreicht ist d.h. sie synchronisieren den Programmablauf wieder.

VC++

VC++ kann sowohl über die umFish30.DLL-Schnittstellen in der um-Variante wie auch der cs-Variante programmiert werden. Beschreibung siehe oben. Die erforderlichen Deklarationen sind im File umFish30VC.H zusammengefaßt. Zusätzlich einfache Beispiele.

Außerdem sind die Sources für umFish30.DLL selber beigefügt. Die Source mit der Ansteuerung des WinRT-Treibers fehlt aus linzenzrechtlichen Gründen.

C++Builder 4

Für C++Builder gibt es mit umFishLoad.H die Deklarationen der um-Variante für eine "schlichte" Version und die Unit FishFa30.H/CPP, die die Klasse TFishFace und Hilfsfunktionen enthält, sie ist mit Windows GUI Anwendungen einsetzbar, die mit dem C++Builder ab Version 4 erstellt wurden. Sie ist in das Projekt aufzunehmen und durch ein #include "FishFa30.h" zugänglich zu machen.

uint steht für unsigned int.

FishFa30 mit Klasse TFishFace

Enums

Dir	Angabe der Drehrichtung ...
Nr	Angabe der Ein-/Ausgangsnr
Speed	Geschwindigkeitsangabe
Wait	Return-Werte von WaitForMotors

Parallel dazu können auch entsprechende numerische (int) Angaben gemacht werden.

Konstruktor

- **FishFace**()
  Mit default-Werten für AnalogScan = false, Slave = false,
  PollInterval = 0, LPTAnalog = 0, LPTDelay = 0
- **FishFace**(bool AnalogScan, bool Slave, uint PollInterval)
  PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTAnalog = 0, LPTDelay = 0
- **FishFace**(bool AnalogScan, bool Slave, uint PollInterval,
  uint LPTAnalog, uint LPTDelay)
  PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTAnalog = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTDelay = 0 Bestimmung durch OpenInterface, >0 wird übernommen

Eigenschaften

bool **AnalogScan** (read)
Angabe ob auch die Analogeingänge gescannt werden sollen (Default = false)

uint	**AnalogsEX** (read) Lesen EX-Wert
uint	**AnalogsEY** (read) Lesen EY-Wert
uint	**Inputs** (read) Lesen der Werte aller E-Eingänge
uint	**LPTAnalog**(read) Lesen Analogskalierung
uint	**LPTDelay**(read) Lesen Ausgabeverzögerung
bool	**NotHalt** (read/write) Anmelden eines Abbruchwunsches (Default = false).
uint	**Outputs** (read) Lesen der Werte aller M-Ausgänge
uint	**PollInterval** (read) Interval (in MilliSekunden) in dem der Status des Interfaces abgefragt(gepollt) und aufgefrischt (refresh) wird.
bool	**Slave** (read) Mit/ohne Extension Module
uint	**Version** (read) Version von FishFa30

Verwendete Variablenbezeichnungen

Die Variablen sind durchweg vom Typ unsigned int.

AnalogNr	Nummer eines Analog-Einganges (Nr)
Counter	Wert eines ImpulsCounters (int)
Direction	Drehrichtung eines Motors (Nr)
InputNr	Nummer eines E-Einganges (Nr)
LampNr	Nummer eines "halben"-M-Ausganges (Nr)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 normal, 0001 RobMode
MotorNr	Nummer eines M-Ausganges (Nr)
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts).
mSek	Zeitangabe in MilliSekunden
NrOfChanges	Anzahl Impulse (int)
OnOff	Ein/Ausschalten eines M-Ausganges (Dir)
PortNr	Nummer der wählbaren Ports (Port)
Position	Positionsangabe in Inpulsen (int)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (Speed)

SpeedStatus	Status der Geschwindigkeiten aller M-ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1. Werte 0000 – 1111 (full).
TermInputNr	Nummer eines E-Einganges mit der die Methode beendet werden soll (Nr)
Value	allgemeiner int-Wert

Methoden

Die Methoden erwarten ein vorhergehendes OpenInterface. Sie enthalten meist ein Application->ProcessMessages(); um das Anwendungsprogramm auch in engen Schleifen unterbrechbar zu machen.

Die Mehrzahl der Methoden löst bei "InterfaceProblem" bzw. "KeinOpen" eine Exception aus, die über TFishFaceException abgefangen werden sollte.

void	**ClearCounter**(uint InputNr) Löschen (0) des angegebenen Counters
void	**ClearCounters**() Löschen (0) aller Counter
void	**ClearMotors**() Abschalten aller M-Ausgänge
void	**CloseInterface**() Schließen der Verbindung zum Interface
bool	**Finish**(uint InputNr) Feststellen eines Endewunsches (NotHalt, Escape, InputNr(optional))
uint	**GetAnalog**(uint AnalogNr) Auslesen der Werte von Ex / EY
uint	**GetAnalogDirect**(uint AnalogNr) Direktes Auslesen der Werte von EX / EY. Dazu wird das Pollen vorübergehen abgeschaltet. Sinnvoll nur, wenn die Motoeren stehen. Vorteil : Das PollInterval kann auf dem kleineren Wert von AnalogScan = false bleiben.
uint	**GetCounter**(uint InputNr) Auslesen des Wertes des angegebenen Counters
bool	**GetInput**(uint InputNr) Auslesen des Wertes des angegebenen E-Einganges
uint	**GetInputs**() Auslesen der Werte aller E-Eingänge
void	**OpenInterface**(LPCSTR PortName[, bool Messages=true]) Setzen von Parametern, Herstellen der Verbindung zum Interface PortName : LPT, COM1, COM2, COM3, COM4, COM5, COM6, COM7, COM8, LPT1, LPT2, LPT3 Message : Application.ProcessMessages in den Methoden einsetzen (Unterbrechbarkeit der Anwenderoberfläche, bei Threads nicht erforderlich).
void	**Pause**(uint mSek) Anhalten des Programmablauf für mSek MilliSekunden

void **SetCounter**(uint InputNr, uint Value)
Setzen des Counters InputNr auf Value

void **SetLamp**(uint LampNr, bool OnOff)
Setzen eines ‚halben' M-Ausganges (je ein Pol an einem M-Ausgang und Masse)

void **SetMotor**(uint MotorNr, uint Direction, uint Speed, uint Counter)
Setzen eines M-Ausganges, optional mit Geschwindigkeitsangabe (Speed) und Angabe der zurückzulegenden Strecke in Impulsen (Counter))

void **SetMotors**(uint MotorStatus, uint Speedstatus, uint ModeStatus)
Setzen des Status aller M-Ausgänge, optional mit Geschwindigkeitsangabe (SpeedStatus) und des Betriebsmodes (ModeStatus, default = 0). Bei Betriebsmodus RobMode (1) sind vor dem Aufruf der Methode die entsprechenden Counter zu setzen (SetCounter[m])

void **WaitForChange**(uint InputNr, uint NrOfChanges, uint TermInputNr)
Warten auf NrOfChanges Impulse an InputNr oder TermInputNr(optional) = true
alt -> LLWin

void **WaitForHigh**(uint InputNr)
Warten auf einen false/true-Durchgang an InputNr

void **WaitForInput**(uint InputNr, bool OnOff)
Warten auf InputNr = OnOff. OnOff ist optional (default = true)

void **WaitForLow**(uint InputNr)
Warten auf eine true/false-Durchgang an InputNr

res **WaitForMotors**(uint Time, uint MotorNr ...)
Warten auf ein MotorReady-Ereignis oder Ablauf Time
Ein MotorReady-Ereignis wird durch SetMotor mit Parameter Count bzw ModeStatus (0001 ...) gestartet und tritt bei Counter = 0 aller MotorNr ein.
Time = 0 : unbegrenztes Warten.
res = ftiEnde : alle Counter = 0 (auch durch Erreichen Endtaster gesetzt)
res = ftiTime : Ablauf Time
res = ftiNotHalt : Beendigung durch NotHalt.
res = fitESC : Beendigung durch Escape-Taste

Bei ftiNotHalt und ftiESC werden außerdem alle betroffenen Motoren angehalten.

void **WaitForPositionDown**(uint InputNr, uint Counter, uint Position, uint TermInputNr)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Herunterzählen der festgestellten Impulse an InputNr.
alt -> LLWin

void **WaitForPositionUp**(uint InputNr, uint Counter, uint Position, uint TermInputNr)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Heraufzählen der festgestellten Impulse an InputNr.
alt -> LLWin

void **WaitForTime**(uint mSek)
Anhalten des Programmablauf für mSek MilliSekunden. wie Pause.

Die **SetMotor**(s)-Methoden sind asynchron d.h. der oder die angesprochenen Motoren (Lampen) werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter. Sie werden durch ein weiteres SetMotors mit Direction = 0 beendet. Ausnahme : SetMotor mit Count-Parameter. Diese Methode beendet sich nach Erreichen der vorgegebenen Position selber.

Die **Wait**-Methoden koordinieren – meist in Verbindung mit End- bzw. ImpulsTastern den asynchronen Motorlauf mit dem Ablauf des Programms.Sie halten den weiteren Progammablauf an, bis das Waitziel (Ablauf Zeit, erreichte Position, Tasterstellung ...) erreicht ist d.h. sie synchronisieren den Programmablauf wieder.
Die Wait-Methoden setzen bei Bedarf den entsprechenden Counter neu.

Delphi 4

Für Delphi gibt es die Unit umFish30.PAS für die um-Variante von umFish30.DLL und die Unit FishFa30 mit der Klasse TFishFace (in delphiFish30Setup.EXE enthalten).

.NET : FishFa30 mit Klasse FishFace

Enthalten in der Assembly FishFa30.DLL mit der Source FishFa30.CS.

In gleicher Weise mit **C#** und **VB.NET** (hier in kompilerter Form : FishFa30.DLL) einsetzbar. Die C# Datentypen int heißen in VB.NET Integer und bool – Boolean

Zu VB.NET : Handbuch "FishFa30 für VB.NET"

in www.ftcomputing.de/zip/vb7model.zip

Enums

Dir	Angabe der Drehrichtung ...
Nr	Angabe der Ein-/Ausgangsnr
Port	Angabe des zu nutzenden Ports
Speed	Geschwindigkeitsangabe
Wait	Return-Werte von WaitForMotors

Parallel dazu können auch entsprechende numerische (int) Angaben gemacht werden.

Konstruktor

- **FishFace**()
  Mit default-Werten für AnalogScan = false, Slave = false,
  PollInterval = 0, LPTAnalog = 0, LPTDelay = 0
- **FishFace**(bool AnalogScan, bool Slave, int PollInterval)
  PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTAnalog = 0, LPTDelay = 0
- **FishFace**(bool AnalogScan, bool Slave, int PollInterval,
  int LPTAnalog, int LPTDelay)
  PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTAnalog = 0 Bestimmung durch OpenInterface, >0 wird übernommen
  LPTDelay = 0 Bestimmung durch OpenInterface, >0 wird übernommen

Eigenschaften

bool **AnalogScan** (get)
Angabe ob auch die Analogeingänge gescannt werden sollen (default = false)

int **AnalogsEX** (get)
Lesen EX-Wert

int **AnalgosEY** (get)
Lesen EY-Wert

int **Inputs** (get)
Lesen der Werte aller E-Eingänge

int **LPTAnalog**(get)
Lesen Analogskalierung

int **LPTDelay**(get)
Lesen Ausgabeverzögerung

bool **NotHalt** (get/set)
Anmelden eines Abbruchwunsches (Default = false).

int **Outputs** (get)
Lesen der Werte aller M-Ausgänge

int **PollInterval** (get)
Interval (in MilliSekunden) in dem der Status des Interfaces abgefragt(gepollt) und aufgefrischt (refresh) wird.

string **PortName** (get)
Aktueller PortName

bool **Slave** (get)
Mit/ohne Extension Module

string **Version** (get, static)
Version der DLL

Verwendete Variablenbezeichnungen

Die Variablen sind durchweg vom Typ int. Parallel dazu gibt es eine Aufrufvariante (overload), die enums verwendet. Hier werden zur Beschreibung des Wertebereichs einer Variablen die enum-Namen angegeben.

AnalogNr	Nummer eines Analog-Einganges (Nr)
Counter	Wert eines ImpulsCounters (int)
Direction	Drehrichtung eines Motors (Dir)
InputNr	Nummer eines E-Einganges (Nr)
LampNr	Nummer eines "halben"-M-Ausganges (Nr)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1, Werte 0000 normal, 0001 RobMode
MotorNr	Nummer eines M-Ausganges (Nr)
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts).
mSek	Zeitangabe in MilliSekunden
NrOfChanges	Anzahl Impulse (int)
OnOff	Ein/Ausschalten eines M-Ausganges (Dir)
PortNr	Nummer der wählbaren Ports (Port)
Position	Positionsangabe in Inpulsen (int)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (Speed)
SpeedStatus	Status der Geschwindigkeiten aller M-ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1. Werte 0000 – 1111 (full).
TermInputNr	Nummer eines E-Einganges mit der die Methode beendet werden soll (Nr)
Value	allgemeiner int-Wert

Methoden

void **ClearCounter**(Nr InputNr)
Löschen (0) des angegebenen Counters

void **ClearCounters**()
Löschen (0) aller Counter

void **ClearMotors**()
Abschalten aller M-Ausgänge

void **CloseInterface**()
Schließen der Verbindung zum Interface

bool **Finish**(Nr InputNr)
Feststellen eines Endewunsches (NotHalt, Escape, InputNr(optional))

int **GetAnalog**(Nr AnalogNr)
Auslesen der Werte von Ex / EY

int **GetAnalogDirect**(Nr AnalogNr)
Auslesen der Werte von Ex / EY. Direktes Auslesen der Werte von Ex / EY. Dazu wird das Pollen vorübergehend abgeschaltet. Sinnvoll nur, wenn die Motoren stehen. Vorteil : Das PollInterval kann auf dem kleineren Wert von AnalogScan = false bleiben.

int **GetCounter**(Nr InputNr)
Auslesen des Wertes des angegebenen Counters

bool **GetInput**(Nr InputNr)
Auslesen des Wertes des angegebenen E-Einganges

int **GetInputs**()
Auslesen der Werte aller E-Eingänge

void **OpenInterface**(Port PortNr, bool DoEvents)
Setzen von Parametern, Herstellen der Verbindung zum Interface
DoEvents : mit/ohne DoEvents

void **Pause**(int mSek)
Anhalten des Programmablauf für mSek MilliSekunden

void **SetCounter**(Nr InputNr, int Value)
Setzen des Counters InputNr auf Value

void **SetLamp**(Nr LampNr, bool OnOff)
Setzen eines ‚halben' M-Ausganges

void **SetMotor**(Nr MotorNr, Dir Direction, Speed Speed, Nr Counter)
Setzen eines M-Ausganges, optional mit Geschwindigkeitsangabe (Speed) und Angabe der zurückzulegenden Strecke in Impulsen (Count))

void **SetMotors**(int MotorStatus, int Speedstatus, int ModeStatus)
Setzen des Status aller M-Ausgänge, optional mit Geschwindigkeitsangabe (SpeedStatus) und des Betriebsmodes (ModeStatus, default = 0). Bei Betriebsmodus RobMode sind vor dem Aufruf der Methode die entsprechenden Counter zu setzen (SetCounter[m])

void **WaitForChange**(Nr InputNr, Nr NrOfChanges, Nr TermInputNr)
Warten auf NrOfChanges Impulse an InputNr

oder TermInputNr(optional) = true
alt -> LLWin

void **WaitForHigh**(Nr InputNr)
Warten auf einen false/true-Durchgang an InputNr

void **WaitForInput**(Nr InputNr, bool OnOff)
Warten auf InputNr = OnOff. OnOff ist optional (default = true)

void **WaitForLow**(Nr InputNr)
Warten auf eine true/false-Durchgang an InputNr

res **WaitForMotors**(int Time, Nr MotorNr ...)
Warten auf ein MotorReady-Ereignis oder Ablauf Time
Ein MotorReady-Ereignis wird durch SetMotor mit Parameter Count bzw ModeStatus (0001 ...) gestartet und tritt bei Counter = 0 aller MotorNr ein.
Time = 0 : unbegrenztes Warten.
res = Wait.Ende : alle Counter = 0 (auch durch Erreichen Endtaster gesetzt)
res = Wait.Time : Ablauf Time
res = WaitNotHalt : Beendigung durch NotHalt.
res = Wait.ESC : Beendigung durch Escape-Taste

void **WaitForPositionDown**(Nr InputNr, int Counter, int Position, Nr TermInputNr)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Herunterzählen der festgestellten Impulse an InputNr.
alt -> LLWin

void **WaitForPositionUp**(Nr InputNr, int Counter, int Position, Nr TermInputNr)
Warten auf Erreichen einer vorgegebenen Position (Counter) ausgehend von der aktuellen (Position) durch Heraufzählen der festgestellten Impulse an InputNr.
alt -> LLWin

void **WaitForTime**(int mSek)
Anhalten des Programmablauf für mSek MilliSekunden

Die Methoden erwarten ein vorhergehendes OpenInterface. Ggf. wird eine entsprechende Exception ausgelöst. Sie enthalten meist ein **DoEvents** um das Programm unterbrechbar zu machen. Wird im Ablauf ein InterfaceProblem festgestellt, wird eine entsprechende **Exception** ausgelöst. Die Wait-Methoden setzen bei Bedarf den zugehörenden **Counter** zurück.

Die SetMotor(s)-Methoden sind **asynchron** d.h. der oder die angesprochenen Motoren (Lampen) werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter. Sie werden durch ein weiteres SetMotors mit Direction = 0 beendet. Ausnahme : SetMotor mit Count-Parameter. Diese Methode beendet sich nach Erreichen der vorgegebenen Position selber.

Die Wait-Methoden koordinieren – meist in Verbindung mit End- bzw. ImpulsTastern den asynchronen Motorlauf mit dem Ablauf des Programms.Sie halten den weiteren Progammablauf an, bis das Waitziel (Ablauf Zeit, erreichte Position, Tasterstellung ...) erreicht ist d.h. sie synchronisieren den Programmablauf wieder.

Anmerkungen zum Kontrollblock

Der Kontrollblock vom Typ ftiDCB enthält zum Betrieb des fischertechnik Interfaces erforderliche Informationen und wird in durch das PollInterval vorgegebenen Abständen im MilliSekundenbereich durch die Routinen von umFish30.DLL aktualisiert. Wenn mehrere Interfaces unabhängig voneinander (also nicht Master/Slave) betrieben werden sollen (z.B. eins an LPT, ein weiteres an COM1) sind entsprechend viele Kontrolblöcke erforderlich.

Der Kontrollblock ist nur bei der um-Variante der umFish30.DLL Funktionen direkt zugänglich. Bei der cs-Variante und bei auf umFish30.DLL aufsetzenden Klassen sind einzelne Felder des Kontrollblocks über entsprechende Zugriffsfunktionen bzw. Eigenschaften/Methoden zugänglich.

Anmerkungen zu den Counters

Ein wesentliches Element zur Positionsbestimmung sind die Counters. Sie sind den E-Eingängen zugeordnet (Achtung : E1 wird je nach Sprache und Implementierung auf Counter 0 bzw. 1 abgebildet). In den Countern wird jede Veränderung des Zustandes der E-Eingänge gezählt. Also z.B. das Öffnen oder auch das Schließen eines Tasters.

Die Counter sind Teil des Kontrollblocks und können dort abgefragt bzw. gesetzt werden. Wenn der Kontrollblock nicht direkt zugänglich ist, werden entsprechende Funktionen/Methoden angeboten. Die Counter werden auch intern von einigen Funktionen/Methoden (z.B. SetMotor mit Parameter Counter und den meisten Wait-Methoden) genutzt – Bei umFish30-Funktionen sind es nur um/csRobMotor(s).

Anmerkungen zur Geschwindigkeitssteuerung

Die Geschwindigkeitssteuerung beruht auf einem zyklischen Ein- und Ausschalten der betroffenen M-Ausgänge (Motoren). Dazu wird intern für jede Geschwindigkeitsstufe eine entsprechende Schaltliste vorgehalten. Die Geschwindigkeit wird durch den Parameter Speed für einen Motor und den Parameter SpeedStatus für alle Motoren angewählt. Die Geschwindigkeitssteuerung erfolgt in einem separaten Thread von umFish30.DLL, der die Motoren bis zu ihrem Ausschalten durch SetMotor(s) so steuert.

Anmerkungen zu den Rob-Funktionen

Die Rob-Funktionen laufen in einem bsonderen Betriebsmodus, dem RobMode. In diesem Modus werden die betroffenen Counter decrementiert. Bei Erreichen des Wertes 0 wird der betroffen Motor abgeschaltet. Während der letzten 6 Impulse fahren sie nur noch mit halber Geschwindigkeit um ein sicheres Erreichen der Endposition zu erreichen. Gelegentlich kann es trotzdem vorkommen, daß noch um einen Impuls weiter gefahren wird. Das kann man durch Abfrage des entsprechenden ImpulsCounters (wert > 0) feststellen und bei der Speicherung der aktuellen Position entsprechend berücksichtigen.

Der Betrieb eines Motors mit den Rob-Funktionen setzt ein festes Anschlußkonzept voraus. Zum jeweiligen Motor gehören je ein Impulstaster und ein Endtaster. Dazu folgende Tabelle :

Motor	**Endtaster**	**Impulstaster**
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10
6	11	12
7	13	14
8	15	16

Die Motoren sind „linksdrehend“ d.h. sie drehen bei ftiLinks in Richtung Endtaster.

Die Motoren können einzeln über umRobMotor/csRobMotor bzw. eine Variante von SetMotors geschaltet werden. Das Argument ICount/Counter gibt die Fahrstrecke in Impulsen (true/false- oder false/true-Durchgang am zugehörenden Impulstaster) an. Sie werden während des Pollens auf Null heruntergezählt. Sie sind über den ftiDCB.Counter bzw. die Funktion csGetCounter zugänglich. Also Achtung, etwa durch die Anwendung gesetzte Counter können in diesem Fall geändert werden.

Die Motoren können auch alle mit einem Befehl geschaltet werden : umRobMotors / csRobMotors bzw einer Variante von SetMotor. Dazu müssen vorher die Parameter aufbereitetet werden.
MotorStatus : pro Motor 2bit, mit M1 : bit 0 und 1 beginnend.
00 : aus, 01 links, 10 rechts.
SpeedStatus : pro Motor 4bit, mit M1 : bit 0-3 beginnend,
0000 aus, 1000 halbe Kraft, 11111 voll.
ModeStatus : proMotor 4 bit, mit M1 : bit 0-3 beginnend,
0000 Normal-Mode, 0001 Rob-Mode, Rest z.Zt. nicht besetzt
(vorgesehen z.B. für Schrittmotorenbetrieb).

Beispiel : csRobMotors(ft, 0x9, 0xF6, 0x11);
0x steht für Hexa, binär : 1001 | 11110110 | 10001 -> M2 = rechts, Speed 15 im Rob-Mode, M1 = links, Speed 6 im RobMode. Der Rest steht.

Vor umRobMotors / csRobMotors sind für jeden Motor einzeln die Impuls-Counter auf die gewünschte Fahrstrecke zu setzen.

Direction = 0 bzw. die Angabe im MotorStatus hält den Motor unabhängig von den Speed-Werten an.

Die Motoren laufen simultan (ggf. auch alle acht), sie können der Reihe nach mit umRobMotor bzw. csRobMotor geschaltet werden. Sie starten dann beim nächsten Pollzyklus (Abfragezyklus) automatisch und laufen asynchron (d.h. unabhängig von den Aktionen des rufenden Programms) bis sie die vorgegebene Position erreicht haben. Sie werden dann ebenfalls (einzeln) während des Pollens abgeschaltet.

Um Festzustellen, ob die Motoren ihr Ziel erreicht haben und um das Programm mit den durch durch die Rob-Funktionen ausgelösten Aktionen wieder zu synchronisieren ist ein WaitForRobMotor(s) erforderlich. Bei den FishFace-Klassen ist eine entsprechende Methode zu finden, umFish30.DLL bietet keine entsprechende Funktion. Bei Einsatz von um- bzw. cs-Funktionen ist sie selber zu programmieren (siehe auch Beispiele, Grund für das Fehlen : umFish30.DLL soll keine sprachabhängigen Elemente enthalten).