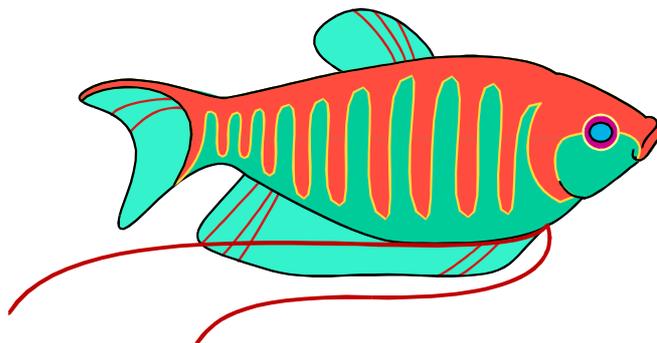


---

ftComputing

# FishFa30 für VB.NET

Ulrich Müller



# Inhaltsverzeichnis

<b>Einführung</b>	<b>4</b>
Allgemeines	4
Interface Panel	5
Literatur zu VB.NET	5
Die StartAmpel	6
<b>Beispiel Riesenrad</b>	<b>8</b>
Allgemeines	8
Das Modell	9
Programmrahmen	10
Schritt 1 : Aus- und Einsteigen	11
Schritt 2 : Die Fun-Runden	11
Schritt 3a : Echt-Betrieb	12
Schritt 3b : Symbolische Namen	13
Schritt 4 : Zählen statt warten	13
Schritt 5 : Überkreuz Beladen	14
<b>Referenz</b>	<b>15</b>
Programmrahmen	15
Projekt in der VS.NET Version	15
Projekt in der SharpDevelop Version	15
Die Source Form1.VB   MainForm.VB	16
Verwendete Variablenbezeichnungen	17
Enums	18
Klasse FishFace	18
Konstruktor	18
Eigenschaften	19
Methoden	20
Anmerkungen	27
Klasse FishRobot	28
Test-Programmrahmen	28
Konstruktor	29
Eigenschaften	29
Methoden	30
Ereignisse	31
Klasse FishStep	32
Test-Programmrahmen	32
Konstruktor	33
Eigenschaften	34
Methoden	34
Ereignisse	36

<b>Tips &amp; Tricks</b>	<b>37</b>
<hr/>	
Programmrahmen	37
Techniken	38
Blinker/Schleife	38
WechselBlinker	38
Abfrage eines E-Einganges	39
Warten auf einen E-Eingang	39
Anzeige des Status der E-Eingänge	39
Analog-Anzeige	39
Fahren für eine bestimmte Zeit	40
Fahren zum Endtaster	40
Fahren um eine vorgegebene Anzahl von Schritten	40
Lampen	41
Lichtschranken	41
Gleichzeitiges Schalten aller M-Ausgänge	42
Betrieb eines Robots	44
Robot-Fahren	44
Positionsanzeige	45
Betrieb von Schrittmotoren	46
Einzelner Schrittmotor	46
Zwei Motoren im XY-Verbund : Plotten	47
<b>Anmerkungen zum Verständnis</b>	<b>48</b>
<hr/>	
Zugriff auf das Interface	48
Anmerkungen zu den Counters	49
Anmerkungen zur Geschwindigkeitssteuerung	49
Anmerkungen zu den Rob-Funktionen	49
Anmerkungen zu den Step-Funktionen	50

Copyright © 1998 – 2004 für Software und Dokumentation :

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873, Fax 05251/55709

eMail : [UM@ftComputing.de](mailto:UM@ftComputing.de)

HomePage : [www.ftcomputing.de](http://www.ftcomputing.de)

Freeware : Eine private – nicht gewerbliche – Nutzung ist kostenfrei gestattet.

Haftung : Software und Dokumentation wurden mit Sorgfalt erstellt, eine Haftung wird nicht übernommen.

Dokumentname : FishFa30VB.NET.DOC. Druckdatum : 16.06.2004

Titelbild : Einfügen | Grafik | AusDatei | Office | Fish9.WMF

# Einführung

---

## Allgemeines

Mit der in C# geschriebenen Assembly FishFa30.DLL wird die Möglichkeit geboten, die fischertechnik Interfaces unter einer .NET Sprache zu programmieren (beschrieben wird hier der Einsatz von VB.NET). FishFa30.DLL setzt auf umFish30.DLL (Systemkonforme.DLL) auf. Die zentrale Klasse FishFace von FishFa30.DLL erlaubt die Ansteuerung der parallelen (Universal) und der seriellen (Intelligent) Interfaces jeweils wahlweise mit Slave/Extension Module. Es können mehrere Interfaces innerhalb eines Programmes simultan betrieben werden.

Angeboten werden Befehle zur Schaltung der M-Ausgänge und zur Abfrage der Eingänge eines Interfaces. Dazu wird das Interface in einem besonderen Thread von umFish30.DLL in regelmäßigen Abständen abgefragt (gepollt -> PollInterval). Zusätzlich werden die Veränderungen (Ein/Aus) an den E-Eingängen gezählt, sie werden außerdem zur Bestimmung der Schaltdauer der M-Ausgänge herangezogen. Dazu ist eine feste Zuordnung des an einen M-Ausgang angeschlossenen Motors und der an die E-Eingänge angeschlossenen Ende- und Impulstaster notwendig (RobMotoren). Die M-Ausgänge können außerdem mit verschiedener "Geschwindigkeit" betrieben werden, dazu werden sie in Intervallen ein- und ausgeschaltet (PWM).

Die A-Eingänge (Analog-Eingänge) liefern Raw-Werte im Bereich von 0 – 1024. Der Betrieb eines Interfaces mit Auslesen der A-Eingänge benötigt ein größeres PollInterval als ohne, das trifft besonders für das parallele Interface zu (Instanzierung mit Parameter AnalogScan = True ist erforderlich).

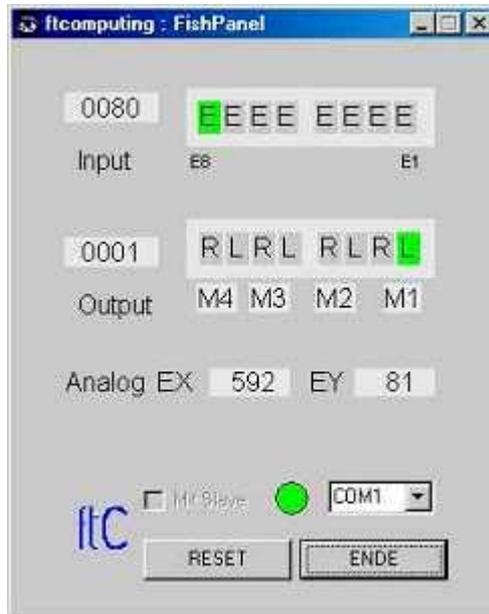
Getestet wurde mit : VS.NET Final auf Windows 2000 mit .NET Framework v1.0

und SharpDevelop milestone 099b Beta auf Windows XP mit .NET Framework v1.1

Ein praktischer Unterschied beim Einsatz von FishFa30.DLL wurde nicht festgestellt. Jedoch verwenden die IDEs unterschiedliche Projekt-Dateien, daß eine Übernahme kompletter Projekte nicht möglich ist. Ebenso gibt es Unterschiede im erzeugten Code von Windows Form Anwendungen. Siehe auch Referenz - Programmrahmen

---

## Interface Panel



Das Interface Panel dient zur Anzeige der Werte eines fischertechnik Interfaces und zum Schalten der M-Ausgänge (Output).

Nach Start des Panels kann eingestellt werden, ob mit Slave (Extension Module) gearbeitet werden soll.

Über die ComboBox kann der Interface-Anschluß gewählt werden (PortName). Bei Wahl von LPT erscheint ein weiterer Button LPT-Optionen, mit denen die Einstellungen des zuvor installierten LPT-Treibers modifiziert werden können (das ist etwas mühselig und erfordert Kenntnisse über die vorhandene Hardware, ist aber meist nicht erforderlich).

Neben der ComboBox wird nach Klick auf START die Betriebsbereitschaft angezeigt.

Die Input-Zeile zeigt den Status aller E-Eingänge an, links als Hexa-Wert.

Die Output-Zeile zeigt den Status der M-Ausgänge an, links wieder als Hexa-Wert. Ein Klick auf L bzw. R schaltet den entsprechenden Ausgang für die Dauer des Klicks ein, wird gleichzeitig die Strg-Taste gedrückt auch dauerhaft. Das Ausschalten erfolgt dann durch Klick auf M1 ... Alle M-Ausgänge können durch Klick auf den RESET-Button gleichzeitig ausgeschaltet werden.

L legt gleichzeitig die Richtung Links (Dir.Links, Dir.Left) fest, also nach dem Modellaufbau testen in welche Richtung es beim L-Klick geht und bei der Programmierung dann berücksichtigen (bei Nichtgefallen : die Motoren umpolen). Analoges gilt für einen R-Klick.

Die Analog-Zeile zeigt die (dezimalen) Werte an die an den Eingängen EX und EY gemessen werden.

---

## Literatur zu VB.NET

- Andreas Kühnel : VB.NET, Galileo, ISBN 3-89842-129-5 (deutsch) als fundierte Einführung.
- Nitty Gritty Visual Basic.NET, Addison-Wesley (deutsch). Handfeste und preiswerte Einführung/Übersicht für Programmierer mit Erfahrung, ISBN 3-8273-1857-2.
- O'Reilly : Programming Visual Basic.NET, ISBN 0-596-00093-6, als Übersicht.
- O'Reilly : VB.NET in a Nutshell, 0-596-00308-0, als Referenz neben der recht ansprechenden Hilfe des Visual Studio.NET.

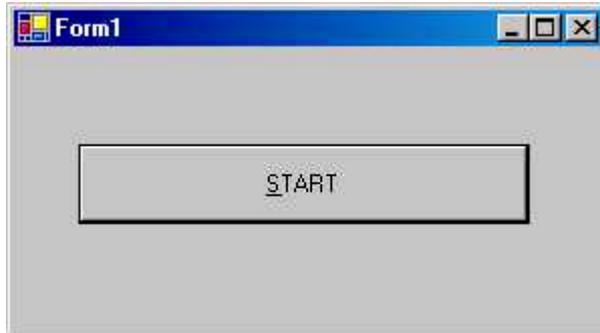
Und dann gibt es jetzt auch von den altbekannten VB-Autoren eine VB.NET Version :

- Michael Kofler : Visual Basic.NET - Grundlagen, Programmiertechniken, Windowsanwendungen - Addison-Wesley ISBN 3-8273-1982-X
- Doberenz / Kowalski : Grundlagen und Profiwissen - Visual Basic.NET, Hanser ISBN 3-446-22024-0

In den beiden letztgenannten Büchern wird auch ausführlich auf die Programmierung mit Windows.Forms eingegangen.

---

# Die StartAmpel



So geht's los :

- Interface anschließen, Funktion mit dem Interface Panel testen
- An das Interface anschließen M1 : grüne, M2 gelbe, M3 rote Lampe
- Neues Projekt anlegen (Windows Anwendung mit einer einfachen Form)
- In der Projektübersicht Verweis (Referenz / Verweise) FishFa30.DLL (Assemblies) eintragen.
- Ganz zu Beginn der Source der ebenangelegten Form `Imports FishFa30` eintragen und vorsichtshalber auch noch `Option Strict Off` und `Option Explicit On`.
- Einen Button `cmdAction` mit der Beschriftung `START` anlegen
- Für `cmdAction` eine Click-Routine anlegen und den unten angegebenen Text eingeben. `Port.COM1` ggf. anpassen.
- `START` Drücken : Es geht los – und das wars denn auch schon.
- Wenn mans genauer wissen will gleich noch mal im Einzelschritt durchlaufen lassen.

Code der `cmdAction` – Click Routine :

```
Dim ft As New FishFace()  
ft.OpenInterface(Port.COM1)  
ft.SetMotor(Nr.M3, Dir.Ein)  
ft.Pause(1000)  
ft.SetMotor(Nr.M2, Dir.Ein)  
ft.Pause(500)  
ft.SetMotor(Nr.M3, Dir.Aus)  
ft.SetMotor(Nr.M2, Dir.Aus)  
ft.SetMotor(Nr.M1, Dir.Ein)  
ft.Pause(2000)  
ft.SetMotor(Nr.M1, Dir.Aus)  
ft.CloseInterface()
```

Das Programm steht in einer Datei mit der Endung `.VB` (Form1 / MainForm / eigener Name).

Hinzu kommen noch die Projekt-Dateien.

Sub `cmdAction` - Click ist die einzige Nutzroutine des Programmes, die die M-Ausgänge schaltet.

Zu den Elementen :

- Dim ft As New FishFace : anlegen einer neuen Instanz der Klasse FishFace (Teil von FishFa30.DLL) mit dem Name ft. Unter diesem Namen werden dann die Methoden (Funktionen) der Klasse angesprochen.
- ft.OpenInterface "Port.COM1" : Herstellen einer Verbindung zum Interface, hier dem Intelligent Interface an COM1 Es werden außerdem eine Reihe von Eigenschaften gesetzt.
- ft.SetMotor(Nr.M3, Dir.Ein) : Einschalten der roten Lampe  
Die Methoden von FishFace bieten meist einen Auswahlliste (Enum) möglicher Parameterwerte. Hier aus der Aufzählung Nr und Dir. Es können aber auch einfache Zahlen oder eigene Konstanten angegeben werden.
- ft.Pause : Das Programm wird für 1000 MilliSekunden (1 Sekunde) angehalten.
- ft.SetMotor(Nr.M2, Dir.Ein) die gelbe Lampe wird für 500 MilliSekunden zugeschaltet. und dann werden beide aus und die grüne Lampe an M1 wird für 2000 MilliSekunden angeschaltet
- und der Ordnung halber : ft.CloseInterface(), die Verbindung zum Interface gekappt.

Das wars denn auch schon für den Anfang. Weiter kann es mit dem Durcharbeiten des Abschnitts Riesenrad. Zu empfehlen, wenn die eigenen Programmierkünste noch nicht besonders ausgeprägt sind. Der Abschnitt Referenz beschreibt Eigenschaften und Methoden von FishFace im Detail. Im Abschnitt Tips & Tricks werden kleine Problemlösungen vorgestellt. Will man noch mehr Beispiele, sollte man sich auf [www.ftcomputing.de/csecke.htm](http://www.ftcomputing.de/csecke.htm) umsehen.

# Beispiel Riesenrad

---

## Allgemeines

Es wird die schrittweise Entwicklung eines Betriebsprogrammes für das Modell Riesenrad aus dem fischertechnik Kasten "Fun Park" (57 484, Anleitung allein 62 959) mit der Programmiersprache VB.NET unter der Entwicklungsumgebung VS.NET, sowie der Assembly FishFa30.DLL beschrieben. Getestet wurde unter Windows 2000 SP2 und .NET Framework 1.0 SP2.

Der angegebene Code für das Betriebsprogramm ist unabhängig von der Entwicklungsumgebung. Jedoch treten bei Einsatz der freien Entwicklungsumgebung SharpDevelop eine Reihe von Abweichungen auf. Das gilt besonders für die Projektdateien, die Handhabung der Importe und die Generierung der visuellen Elemente einschließlich der zugehörigen Ereignisroutinen. Das Programm für Schritt 1 wurde deswegen auch mit SharpDevelop v0.99b unter Windows XP erstellt. Das dürfte eine Adaption der weiteren Sources ermöglichen. Ein Importieren von VS.NET Projekten ist möglich, machte aber bei den getesteten Projekten mehr Probleme als das Neuanlegen von SharpDevelop Projekten mit anschließendem Kopieren des "Nutz"-Codes.

Zum Anlegen eines VS.NET bzw. SharpDevelop Projektes siehe Abschnitt. Referenz.

---

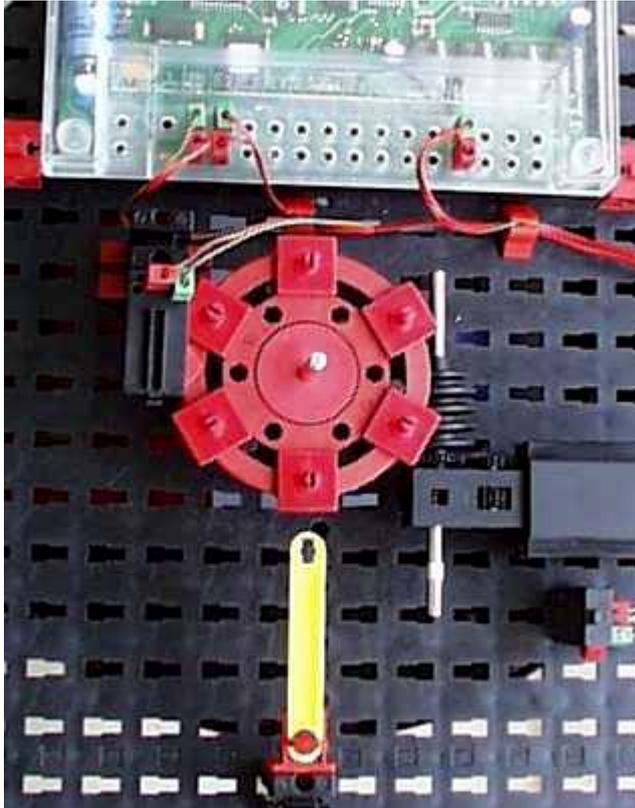
## Das Modell



Das mit Motorantrieb ausgerüstete Modell entspricht weitgehend dem Original des Kastens Fun Park. Es wurde auf den Betrieb mit dem Intelligent Interface umgerüstet :

- Antriebsmotor an M1
- Im Fußbereich wurden das Intelligent Interface und der Taster E1 untergebracht.
- Am hinteren Scheibenrad wurden Schaltnocken (siehe Bild rechts) und der Taster E2 montiert.

Wenn der Wunsch der Einarbeitung in VB.NET im Vordergrund steht, kann alternativ zum realen Modell kann aber auch ein Simulationsmodell eingesetzt werden. Es ist bei weitem nicht so sperrig :



Es kann die gleichen Steuerfunktionen ausführen wie das Original. Das Zeitverhalten entspricht nicht ganz dem großen Modell. Besonders beim Anfahren und Anhalten macht sich das bemerkbar. Die verwendete Zeitkonstante bewegt sich im Bereich 1500 ( Riesenrad mit 12 Männchen / Simulations Modell) bis zu 3200 (leeres Riesenrad). Der gelbe Zeiger markiert die Position der unteren Gondel.

## Programmrahmen

Anlegen des Projektes siehe FishFa30 Handbuch Abschnitt Referenz



Das Windows Form Programm ist sehr einfach gehalten, hier der Aufbau von Schritt 5, die vorhergehenden Schritte enthalten teilweise noch weniger Elemente :

- lblStatus : Label Control zur Anzeige des aktuellen Status
- nudRunden bzw. nudFaktor : Spin Control zur Eingabe der Rundenzahl bzw. des Rundenfaktors.
- cmdAction : Button Control zum Start des Programms. Zugehörige Ereignis-Routine : cmdAction\_Click (VB.NET) bzw. CmdActionClick bei SharpDevelop.

Das Betriebsprogramm läuft überwiegend in der Action-Ereignis-Routine ab.

---

## Schritt 1 : Aus- und Einsteigen

```
Private Sub cmdAction_Click( ...
    Dim i%
    Try
        ft.OpenInterface(Port.COM2)
        For i = 1 To 6
            ft.SetMotor(Nr.M1, Dir.Links)
            ft.WaitForHigh(Nr.E2)
            ft.Pause(1500)
            ft.SetMotor(Nr.M1, Dir.Aus)
            ft.Pause(4000)
        Next
    Catch eft As FishFaceException
        MsgBox(eft.Message)
    Finally
        ft.CloseInterface()
    End Try
End Sub
```

Alle Gondeln werden nacheinander zur Einsteige-Position gefahren. Zum Aus- und Einsteigen wird eine feste Zeit gehalten.

Kernpunkt der Positionserkennung ist der Befehl WaitForHigh (Warten auf einen False/True-Durchgang). ein schlichtes Warten auf E2 = True reicht nicht, da der Taster noch auf True stehen kann. Da mit E2 = True die exakte Einsteige-Position nicht gewährleistet ist (Lage der Schalt-nocken, Bremsverhalten des Modells), wird nach E2 = True noch ein paar (1,5 Sekunden) weitergefahren und dann erst abgeschaltet. Danach folgt die Pause für das Aus- und Einsteigen. Das ganze immer schön in einer 6er-Schleife.

ACHTUNG : Die Länge der Pause (hier Pause 1500) hängt beim Original Riesenrad von der Beladung und der Befestigung der Hauptachse ab. Sie lag bei mir zwischen 1500 (12 Männeken) und 3200(leer).

---

## Schritt 2 : Die Fun-Runden

```
Private Sub cmdAction_Click( ...
    Dim i%
    Try
        ft.OpenInterface(Port.COM2)
        Do
            lblStatus.Text = "--- Ein- und Aussteigen ---"
            For i = 1 To 6
                ft.SetMotor(Nr.M1, Dir.Links)
                ft.WaitForHigh(Nr.E2)
                ft.Pause(1500)
                ft.SetMotor(Nr.M1, Dir.Aus)
                lblStatus.Text = "Gondel : " & i
                ft.Pause(4000)
            Next
            lblStatus.Text = "--- Tour linksrum ---"
            ft.SetMotor(Nr.M1, Dir.Links)
            ft.Pause(1000 * nudFaktor.Value)
            ft.SetMotor(Nr.M1, Dir.Aus)
            ft.Pause(1000)
            lblStatus.Text = "--- Tour rechtsrum ---"
            ft.SetMotor(Nr.M1, Dir.Rechts)
```

```

        ft.Pause(1000 * nudFaktor.Value)
        ft.SetMotor(Nr.M1, Dir.Aus)
        ft.Pause(1000)
    Loop Until ft.Finish(Nr.E1)
    lblStatus.Text = "--- Demo beendet ---"
Catch eft As FishFaceException
    MsgBox(eft.Message)
Finally
    ft.CloseInterface()
End Try
End Sub

```

Das ganze Programm wird in eine Do ... Loop Schleife gepackt, so erhält man ein schönes Demo-Programm, das durch E1 = True oder die ESC-Taste abgebrochen werden kann. Zusätzlich wird in lblStatus die aktuelle Funktion angezeigt.

Nach dem Ein-/Aussteigen (wie bisher) wird 15 Sekunden links und dann 15 rechts gedreht. Da es bei sofortiger Richtungsumschaltung richtig knirschen kann, wird dazwischen 1 Sekunde Pause eingelegt.

Wenn man die Runden gerne länger hätte, kann man im Spin Control einen Rundenfaktor eingeben (Vorgabe 10, Rundenzeit also 10 Sekunden).

---

## Schritt 3a : Echt-Betrieb

```

Private Sub cmdAction_Click( ...
    Dim i%
    Try
        ft.OpenInterface(Port.COM2)
    Do
        lblStatus.Text = "--- Ein- und Aussteigen ---"
        For i = 1 To 6
            ft.SetMotor(Nr.M1, Dir.Links)
            ft.WaitForHigh(Nr.E2)
            ft.Pause(1500)
            ft.SetMotor(Nr.M1, Dir.Aus)
            lblStatus.Text = "Gondel : " & i
            ft.WaitForLow(Nr.E1)
        Next
        lblStatus.Text = "Fahrbetrieb : Quittungs-Taster"
        ft.Pause(3000)
        If Not ft.GetInput(Nr.E1) Then Exit Do
..... weiter wie gehabt .....
    End Try
End Sub

```

Bei einem Echt-Betrieb sind die genauen Zeiten für Aus- und Einsteigen nicht vorhersehbar, die Pause(4000) durch eine WaitForLow(Nr.E1) ersetzt. Das Programm wartet bis E1 gedrückt und wieder freigegeben wird.

Nach dem Aus- und Einsteigen muß der Betrieb durch erneutes Drücken der E1-Taste innerhalb von 3 Sekunden freigegeben werden, sonst wird das Programm beendet : Feierabend.

Man kann sich zum Betrieb natürlich locker noch mehr einfallen lassen.

---

## Schritt 3b : Symbolische Namen

```
Const mRadMotor = Nr.M1, eRadPos = Nr.E2, eQuittung = Nr.E1
Dim ft As New FishFace()

Private Sub cmdAction_Click( ...
    Dim i%
    Try
        ft.OpenInterface(Port.COM2)
        Do
            lblStatus.Text = "--- Ein- und Aussteigen ---"
            For i = 1 To 6
                ft.SetMotor(mRadMotor, Dir.Links)
                ft.WaitForHigh(eRadPos)
            ... weiter wie gehabt, aber mit den symbolischen Namen ...
```

Anstelle der allgemeinen Bezeichnungen (Enums) für die Ein- und Ausgänge des Interfaces sollte man, wenn's was größeres wird, besser spezielle symbolische Namen setzen. Hier mRadMotor, eRadPos, eQuittung.

Sie können dann überall verwendet werden, wo jetzt die Enums stehen.

---

## Schritt 4 : Zählen statt warten

```
Private Sub cmdAction_Click( ...
    Dim i%
    Try
        .....
        If Not ft.GetInput(eQuittung) Then Exit Do
        ft.SetMotor(mRadMotor, Dir.Links)
        For i = 1 To nudRunden.Value
            lblStatus.Text = "--- Runde : " & i & " linksrum ---"
            ft.WaitForChange(eRadPos, 12)
        Next
        ft.SetMotor(mRadMotor, Dir.Aus)
        ft.Pause(1000)
        ft.SetMotor(mRadMotor, Dir.Rechts)
        For i = 1 To nudRunden.Value
            lblStatus.Text = "--- Runde : " & i & " rechtsrum ---"
            ft.WaitForChange(eRadPos, 12)
        Next
        ft.SetMotor(mRadMotor, Dir.Aus)
        ft.Pause(1000)
        Loop Until ft.Finish(eQuittung)
        lblStatus.Text = "--- Betrieb beendet ---"
        Catch eft As FishFaceException
            MsgBox(eft.Message)
        Finally
            ft.CloseInterface()
        End Try
    End Sub
```

Die Pause(1000 \* nudFaktor.Value) wurde durch eine Schleifenkonstruktion ersetzt. Die Schleife selber enthält ein WaitForChange(eRadPos, 12), das ist genau eine Runde, da WaitForChange die Flanke d.h. den Übergang von True/False und False/True zählt. Die Schleife wird sooft durchlaufen, wie es der Spin Control nudRunden.Value (ex nudFaktor)

angibt. Das bringt zum einen eine angebbare Rundenzahl, zum anderen die Möglichkeit, die aktuelle Rundennummer auch auszugeben.

---

## Schritt 5 : Überkreuz Beladen

```
Private Sub cmdAction_Click( ...
    Dim i As Integer
    Try
        ft.OpenInterface(Port.COM2)
        Do
            lblStatus.Text = "--- Ein- und Aussteigen ---"
            For i = 1 To 3
                Beladen(1, i * 2 - 1)
                Beladen(3, i * 2)
            Next
            lblStatus.Text = "Fahrbetrieb : Quittungs-Taster"
        .....
    End Sub

Sub Beladen(ByVal Position As Integer, ByVal Runde As Integer)
    Dim n%
    ft.SetMotor(mRadMotor, Dir.Links)
    For n = 1 To Position
        ft.WaitForHigh(eRadPos)
    Next
    ft.Pause(1500)
    ft.SetMotor(mRadMotor, Dir.Aus)
    lblStatus.Text = "Gondel : " & Runde
    ft.WaitForLow(eQuittung)
End Sub
```

In der Praxis werden bei einem Riesenrad die Gondeln selten in ihrer direkten Reihenfolge "beladen". Das Beladen übernimmt hier ein gleichnamiges Unterprogramm. Das Unterprogramm selber entspricht weitgehend dem bisherigen Belade-Code, lediglich die Ansteuerung der Position findet jetzt in einer Schleife statt.

Anstelle des Belade-Codes in cmdAction\_Click findet man dort eine For ... Next Schleife, die Beladen aufruft. Der erste Parameter gibt die relative Nummer der nächsten Beladeposition an (Anzahl Gondeln, die zu überspringen sind, + 1). Der zweite Parameter gibt an die wievielte Gondel zu beladen ist, sie wird aus der äußeren For .. Next Schleife abgeleitet.

Beladen wird in der Reihenfolge 1 – 4 – 5 – 2 – 3 – 6. Also die gegenüberliegende Gondel und deren Nachbar. Denkbar sind natürlich auch noch andere Beladepläne.

Man könnte auch noch durch Anbau eines zusätzlichen Tasters samt Nocken die Gondel Nr. 1 identifizieren. Dann kann man auch noch die einzelnen Gondeln mit der zugehörigen Nummer beschriften, so kann die man über die Position jede Gondel genau Buchführen.

# Referenz

---

## Programmrahmen

### Projekt in der VS.NET Version

Anlegen eines neuen Projektes mit Menü :

Datei | Neu | Projekt | VisualBasic-Projekte | Windows Anwendung

- Auswahl des Verzeichnisses in das das Projekt gespeichert werden soll
- Festlegen des Projektnamens

Ergebnis :

Eine Projektmappe (\*.SLN) mit dem Projekt (\*.VBPROJ) und der Windows-Form Form1.VB. Zusätzlich noch weitere Dateien, die hier nicht weiter interessieren.

- Beim eben angelegten Projekt (im Projektmappen Explorer) wird ein Verweis auf die Assembly FishFa30.DLL angelegt :

Projekt | Verweise | Verweise hinzufügen | Durchsuchen

FishFa30.DLL auswählen. FishFa30.DLL kann in einem beliebigen (zentralen)Verzeichnis liegen.

### Projekt in der SharpDevelop Version

Anlegen eines neuen Projektes mit Menü :

Datei | Neu | Combine

VBNET | VB Windows Forms Anwendung

- Auswahl des Verzeichnisses in das das Projekt gespeichert werden soll
- Festlegen des Projektnamens

Ergebnis :

Eine Projektmappe (\*.CMBX) mit dem Projekt (\*.PRJX) und der Windows-Form MainForm.VB.

Zusätzlich noch weitere Dateien, die hier nicht weiter interessieren.

- Beim eben angelegten Projekt (im Projektmappen Explorer) wird ein Verweis auf die Assembly FishFa30.DLL angelegt :

Projekte | combine | ..project.. | Referenzen | Referenz hinzufügen

.NET AssemblyBrowser | Browse

FishFa30.DLL auswählen. FishFa30.DLL kann in einem beliebigen (zentralen)Verzeichnis liegen.

## Die Source Form1.VB | MainForm.VB

Als erste Zeilen :

```
Option Explicit On
Option Strict Off
Import FishFa30
```

eintragen. Die Options könnten entfallen, da sie default sind – aber man weiß ja nie. Explicit On verlangt eine Deklaration aller Variablen, Strict Off : implizierte Typenkonversion ist zulässig (d.h. Zuweisung eines Integers an einen String ist zulässig, z.B.)

Hinter dem Kasten "Von Windows Form Designer generierter Code" eine Instanz der Klasse FishFace anlegen (Bei SharpDevelop am Ende des bisher generierten Codes) :

```
Dim ft As New FishFace()
```

Wenn man die Analog-Eingänge EX/EY nutzen will :

```
Dim ft As New FishFace(True, False, 0)
```

Die Parameter bedeuten : mit Auswertung der Analog-Eingänge, kein Extension Module, Standard-Werte für das PollInterval.

Anlegen eines Start-Buttons auf der Form und der dazu gehörenden Routine für das Click-Ereignis (Durch DoppelKlick auf den Button in der Form).

In der Ereignis-Routine wird dann der Programmcode untergebracht :

```
Private Sub Button1_Click(...) ... | ButtonClick(...)
' --- Variablen Deklarationen
Try
    ft.OpenInterface(Port.COM1)
' --- Programmcode
Catch eft As FishFaceException
    MsgBox(eft.Message)
Finally
    ft.CloseInterface()
End Try
End Sub
```

Der Try – Block fängt mögliche Fehler aus der Klasse FishFace ab (aber nur diese), sie werden ggf. in einer MessageBox angezeigt.

Mit OpenInterface wird die Verbindung zum Interface hergestellt (hier natürlich, den eigenen Anschluß angeben). ft.CloseInterface() hebt die Verbindung wieder auf.

Auf Basis dieses Programmrahmens kann man nette kleine Testprogramme erstellen, z.B. zum probieren mit den Beispielen der Referenz. Bei richtigen Anwendungen sollte man sich etwas mehr einfallen lassen. Der Programmrahmen sollte aber eingehalten werden.

**ACHTUNG** : auch hier gibt es Unterschiede zwischen VS.NET und SharpDevelop. Gravierend Generierung der Class Form1 bzw. Class MainForm. Eher harmlos die Form der Button Click Routine. Im praktischen Betrieb fällt das aber weiter nicht auf. Die Übernahme von VS.NET Forms.VB ist möglich, jedoch sind die System Imports nachzutragen, die bei VS.NET teilweise implizit gemacht werden.

---

## Verwendete Variablenbezeichnungen

Die Variablen sind durchweg vom Typ Integer. Parallel dazu gibt es eine Aufrufvariante (overload), die Enums verwendet. Hier werden zur Beschreibung des Wertebereichs einer Variablen die beschreibende Namen angegeben und in Klammern das entsprechende Enum bzw. der Datentyp.

Die Parameter-Angaben erfolgen – soweit nicht extra notiert - ByVal

AnalogNr	Nummer eines Analog-Einganges (Nr) EX = 0, EY = 1
Counter	Wert eines ImpulsCounters (Integer)
Direction	Drehrichtung eines Motors (Dir) Aus = 0, Ein = 1, Links = 1, Rechts = 2
InputNr	Nummer eines E-Einganges (Nr) E1 = 1 ... E16 = 16
InputStatus	Aktueller Status aller E-Eingänge. Jeweils ein bit pro Ausgang, begonnen bei 0 (0 = E1, 1 = E2 .... 15 = E16)
LampNr	Nummer eines "halben"-M-Ausganges (Integer) Werte 1 - 16
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) M1 = 1 ... M8 = 8
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 (Integer)
OnOff	Ein/Ausschalten eines M-Ausganges (Dir) Off = 0, On = 1
PortNr	Nummer der wählbaren Ports (Port) LPT = 0, COM1 = 1 ... COM8 = 8
Position	Positionsangabe in Impulsen (Integer)
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (Speed) Off = 0, 1 – 15 (Full)
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) E1 = 1 ... E16 = 16
Value	allgemeiner Integer-Wert

---

## Enums

Verwendung zur Eingaben von Parametern bei den FishFace-Methoden

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 (Integer) Angaben gemacht werden. Dazu siehe "Verwendete Variablenbezeichnungen".

---

## Klasse FishFace

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

FishFace ist die Basisklasse der Assembly FishFa30.

## Konstruktor

### **FishFace()**

Mit default-Werten für AnalogScan = False, Slave = False,  
PollInterval = 0, LPTAnalog = 0, LPTDelay = 0

### **FishFace(AnalogScan As Boolean, Slave As Boolean, PollInterval As Integer)**

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen  
LPTAnalog = 0, LPTDelay = 0

### **FishFace(AnalogScan As Boolean, Slave As Boolean, PollInterval As Integer, LPTAnalog As Integer, LPTDelay As Integer)**

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen  
LPTAnalog = 0 Bestimmung durch OpenInterface, >0 wird übernommen  
LPTDelay = 0 Bestimmung durch OpenInterface, >0 wird übernommen

AnalogScan : mit/ohne Auswerten der Analog-Eingänge EX/EY

Slave : mit/ohne Slave/Extension Module

PollInterval : Zeitinterval in MilliSekunden in dem das Interface abgefragt und upgedated werden soll. Wert = 0 : Der Wert wird intern beim OpenInterface bestimmt. Der aktuelle Wert kann (nach OpenInterface) mit der Eigenschaft PollInterval abgefragt werden und ggf. bei der nächsten Instanzierung – vorsichtig – verändert werden. ACHTUNG zu kleine Werte führen zum "Aufhängen" des Rechners.

LPTAnalog : Skalierfaktor für Analogwerte.

LPTDelay : Ausgabeverzögerung bei Universal(LPT) Interface (Werte 50 – 1000)

LPTAnalog / LPTDelay werden nur der Vollständigkeit halber aufgeführt, da sie nur auf Win9x-Systemen (mit Direktzugriff auf den LPT-Port) zum Einsatz kommen können. Und da gibt kein .NET Framework.

## Eigenschaften

Boolean	<b>AnalogScan</b> (readonly) Angabe ob auch die Analogeingänge gescannt werden sollen (default = False)
Integer	<b>AnalogsEX</b> (readonly) Lesen EX-Wert
Integer	<b>AnalogsEY</b> (readonly) Lesen EY-Wert
Integer	<b>Inputs</b> (readonly) Lesen der Werte aller E-Eingänge
Integer	<b>LPTAnalog</b> (readonly) Lesen Analogskalierung
Integer	<b>LPTDelay</b> (readonly) Lesen Ausgabeverzögerung
Boolean	<b>NotHalt</b> Anmelden eines Abbruchwunsches (Default = False).
Integer	<b>Outputs</b> Lesen/Schreiben der Werte aller M-Ausgänge
Integer	<b>PollInterval</b> (readonly) Interval (in MilliSekunden) in dem der Status des Interfaces abgefragt(gepollt) und aufgefrischt (refresh) wird.
string	<b>PortName</b> (readonly) Aktueller PortName
Boolean	<b>Slave</b> (readonly) Mit/ohne Extension Module
string	<b>Version</b> (readonly, static) Version der DLL

readonly : der Wert kann nur ausgelesen, nicht aber verändert werden.

## Methoden

### ClearCounter

Löschen (0) des angegebenen Counters

ft.**ClearCounter**(InputNr)

Siehe auch : ClearCounters, GetCounter, SetCounter

### ClearCounters

Löschen (0) aller Counter

ft.**ClearCounters**()

Siehe auch : ClearCounter, GetCounter, SetCounter

### ClearMotors

Abschalten aller M-Ausgänge

ft.**ClearMotors**()

Exception : InterfaceProblem, KeinOpen

Siehe auch : SetMotor, SetMotors, SetLamp Outputs

### CloseInterface

Schließen der Verbindung zum Interface

ft.**CloseInterface**()

Siehe auch : OpenInterface

### Finish

Feststellen eines Endewunsches (NotHalt, Escape, E-Eingang(optional))

Boolean = ft.**Finish**(Optional InputNr)

Exception : InterfaceProblem, KeinOpen. DoEvents

Siehe auch : GetInput, GetInputs, Inputs

Beispiel :

```
Do  
    . . . .  
Loop Until ft.Finish(Nr.E1)
```

Die Do .. Loop-Schleife wird solange durchlaufen, bis entweder ft.NotHalt = True, die ESC-Taste gedrückt oder E1 = True wurde.

### GetAnalog

Feststellen eines Analogwertes(EX / EY).

Es wird der intern vorliegende Wert ausgegeben. Der Parameter AnalogScan muß bei der Instanzierung muß auf True gesetzt werden.

Value = ft.**GetAnalog**(AnalogNr)

Exception : InterfaceProblem, KeinOpen

Siehe auch : GetAnalogDirect, AnalogsEX, AnalogsEY, AnalogScan, Instanziierung

Beispiel

```
lblAnalog.Text = ft.GetAnalog(Nr.EX)
```

Dem Label lblAnalog wird der aktuelle Wert von EX zugewiesen.

## GetAnalogDirect

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 (bei der Instanziierung) bleiben.

Value = ft.**GetAnalogDirect**(AnalogNr)

Exception : InterfaceProblem, KeinOpen

Siehe auch : GetAnalog, AnalogsEX, AnalogsEY, AnalogScan, Instanziierung

Beispiel

```
lblAnalog.Text = ft.GetAnalogDirect(Nr.EY)
```

Dem Label lblAnalog wird der aktuelle Wert von EY zugewiesen.

## GetCounter

Auslesen des Wertes des angegebenen Counters

Value = ft.**GetCounter**(InputNr)

Siehe auch : SetCounter, ClearCounter, ClearCounters

Beispiel

```
lblPosTurm.Text = ft.GetCounter(Nr.E2)
```

Dem Label lblPosTurm wird der aktuelle Zählerstand, der dem E-Eingang E2 zugeordnet ist, zugewiesen.

## GetInput

Auslesen des Wertes des angegebenen E-Einganges

Boolean = ft.**GetInput**(InputNr)

Exception : InterfaceProblem, KeinOpen. DoEvents

Siehe auch : GetInputs, Inputs, Finish, WaitForInput

Beispiel

```
If ft.GetInput(Nr.E1) Then  
    ...  
Else  
    ...  
End If
```

Wenn der E-Eingang E1 (Taster, PhotoTransistor, Reedkontakt ...) = True ist, wird der Then-Zweig durchlaufen. Bei Not ft.GetInput(Nr.E1) wird der Else-Zweig durchlaufen. Möglich ist auch If ft.GetInput(Nr.E1) = False Then ...

## GetInputs

Auslesen der Werte aller E-Eingänge

InputStatus = ft.**GetInputs**()

Exception : InterfaceProblem, KeinOpen; DoEvents

Siehe auch : GetInputs, Inputs, Finish, WaitForInputs

### Beispiel

```
Dim e14%
    e14 = ft.GetInputs()
    If (e14 And &H1) Or (e14 And &H4) Then ...
```

Wenn die E-Eingänge E1 oder E3 True sind, wird der Then-Zweig ausgeführt.

## OpenInterface

Setzen von FishFace-Eigenschaften, Herstellen der Verbindung zum Interface

**ft.OpenInterface**(PortNr, Optional DoEvents As Boolean)

DoEvents : mit/ohne DoEvents (die Mehrzahl der Methoden kann wahlweise intern durch den Befehl DoEvents unterbrochen werden um eine Unterbrechbarkeit der Anwenderoberfläche sicherzustellen. Default : True.

Exception : InterfaceProblem

Siehe auch : CloseInterface

### Beispiel

```
Try
    ft.OpenInterface(Port.COM2)
Catch eft As FishFaceException
    lblStatus.Text = eft.Message
    ...
End Try
```

Herstellen der Verbindung zum Interface an COM2, DoEvents wird ausgeführt. Im Fehlerfall wird im Label lblStatus der Text 'InterfaceProblem.Open' ausgegeben

## Pause

Anhalten des Programmablauf für mSek MilliSekunden

**ft.Pause**(mSek)

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForTime

### Beispiel

```
ft.SetMotor(Nr.M1, Dir.Links)
ft.Pause(1000)
ft.SetMotor(Nr.M1, Dir.Aus)
```

Der Motor am M-Ausgang M1 wird für eine Sekunde (1000 MilliSekunden) eingeschaltet.

## SetCounter

Setzen des Counters für den angegebenen E-Eingang

**ft.SetCounter**(InputNr, Value)

Siehe auch : GetCounter, ClearCounter, ClearCounters

## SetLamp

Setzen eines 'halben' M-Ausganges. Anschluß einer lampe oder eines Magneten ... an einen Kontakt eines M-Ausganges und Masse.

**ft.SetLamp**(LampNr, OnOff)

Exception : InterfaceProblem, KeinOpen

Siehe auch : SetMotors, SetMotors, ClearMotors

### Beispiel

```
Const lGruen = 1, lGelb = 2, lRot = 3
```

```
ft.SetLamp(lGruen, Dir.Ein)
ft.Pause(2000)
ft.SetLamp(lGruen, Dir.Aus)
ft.SetLamp(lGelb, Dir.Ein)
```

Die grüne Lampe an M1-vorn und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an M1-hinten.

## SetMotor

Setzen eines M-Ausganges (Motor). Die Motordrehzahl kann gewählt werden (Default = Full), ebenso die Fahrstrecke in Anzahl Impulsen. Siehe auch "Anmerkungen zu den Rob-Funktionen.

ft.**SetMotor**(MotorNr, Direction, Optional Speed, Counter)

Exception : InterfaceProblem, KeinOpen; DoEvents; Counter (bei Parameter Counter)

Siehe auch : SetMotors, ClearMotors, SetLamp, Outputs.

### Beispiel 1

```
ft.SetMotor(Nr.M1, Dir.Rechts, Speed.Full)
ft.Pause(1000)
ft.SetMotor(Nr.M1, Dir.Links, Speed.Half)
ft.Pause(1000)
ft.SetMotor(Nr.M1, Dir.Aus)
```

Der Motor am M-Ausgang M1 wird für 1000 Millisekunden rechtsdrehend, volle Geschwindigkeit eingeschaltet und anschließend für 1000 MilliSekunden linksdrehend, halbe Geschwindigkeit.

### Beispiel 2

```
ft.SetMotor(Nr.M1, Dir.Links, 12, 123)
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am E-Eingang E2 oder E1 = True mit Geschwindigkeitsstufe 12 eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Siehe Auch Beispiel WaitForMotors.

## SetMotors

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]) Siehe auch "Anmerkungen zu den Rob-Funktionen"

ft.**SetMotors**(MotorStatus, Optional SpeedStatus, ModeStatus)

Exception : InterfaceProblem, KeinOpen; DoEvents, Counter(bei Parameter Counter)

Siehe auch : ClearMotors, SetMotors, SetLamp, Outputs

### Beispiel

```
ft.SetMotors(&H1 + &H80)
ft.Pause(1000)
ft.ClearMotors()
```

Der M-Ausgang (Motor) M1 wird auf links geschaltet und gleichzeitig M4 auf rechts. Alle anderen Ausgänge werden ausgeschaltet. Nach 1 Sekunde werden alle Ausgänge abgeschaltet.

## WaitForChange

Warten auf NrOfChanges Impulse an InputNr oder TermInputNr = True

Intern wird der zu InputNr gehörende Counter genommen, der dazu zu Beginn zurückgesetzt wird.

**ft.WaitForChange**(InputNr, NrOfChanges, Optional TermInputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar.

Siehe auch : WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh.

Beispiel

```
ft.SetMotor(Nr.M1, Dir.Links)
ft.WaitForChange(Nr.E2, 123, Nr.E1)
ft.SetMotor(Nr.M1, Dir.Aus)
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an E-Eingang E2 oder E1 = True gewartet, der Motor wird abgeschaltet. Solange wird der Programmablauf angehalten. Siehe auch Beispiel bei SetMotors : dort läuft das Programm weiter.

## WaitForHigh

Warten auf einen False/True-Durchgang an einem E-Eingang

**ft.WaitForHigh**(InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForLow, WaitForChange, WaitForInput.

Beispiel

```
ft.SetMotor(Nr.M1, Dir.Ein)
ft.SetMotor(Nr.M2, Dir.Links)
ft.WaitForHigh(Nr.E1)
ft.SetMotor(Nr.M2, Dir.Aus)
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband aus der Lichtschranke ausgefahren ist (die Lichtschranke wird geschlossen), dann wird abgeschaltet. Die Lichtschranke muß vorher False sein (unterbrochen).

## WaitForInput

Warten, daß der angegebene E-Eingang den vorgegebenen Wert annimmt. (Default = True)

**ft.WaitForInput**(InputNr, Optional OnOff)

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar

Siehe auch : WaitForChange, WaitForLow, WaitForHigh.

Beispiel

```
ft.SetMotor(Nr.M1, Dir.Links)
ft.WaitForInput(Nr.E1)
ft.SetMotor(Nr.M1, Dir.Aus)
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf E-Eingang = True gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer EndPosition.

## WaitForLow

Warten auf einen True/False-Durchgang an einem E-Eingang

**ft.WaitForLow**(InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForChange, WaitForInput, WaitForHigh.

Beispiel

```
ft.SetMotor(Nr.M1, Dir.Ein)
ft.SetMotor(Nr.M2, Dir.Links)
```

```
ft.WaitForLow(Nr.E1)
ft.SetMotor(Nr.M2, Dir.Aus)
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an E-Eingang E1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband in die Lichtschranke einfährt (sie unterbricht), dann wird abgeschaltet. Die Lichtschranke muß vorher True sein (nicht unterbrochen).

## WaitForMotors

Warten auf ein MotorReadyEreignis oder den Ablauf von Time

WaitWert = ft.**WaitForMotors**(Time, MotorNr, ...)

Time (Integer) : Zeit in MilliSekunden. Bei Time = 0 wird endlos gewartet.

MotorNr(Nr) : Liste von M-Ausgängen in beliebiger Reihenfolge auf die gewartet werden soll. Gewartet wird auf MotorStatus = Aus für die betreffenden M-Ausgänge gewartet.

MACKE : hier müssen echte numerische Werte verwendet werden (1, 2 ... 8)

WaitWert(Wait) : Grund warum die Methode beendet wurde

Wait.Ende : Alle betroffenen M-Ausgänge = Dir.Aus

Wait.Time : Die vorgegebene Wartezeit ist abgelaufen

Wait.NotHalt : Die Eigenschaft NotHalt = True, alle betroffenen Motoren wurden angehalten

Wait.ESC : Die ESC-Taste wurde betätigt, alle betroffenen Motoren wurden angehalten.

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : SetMotor

### Beispiel

```
ft.SetMotor(Nr.M4, Dir.Links, Speed.Half, 50)
ft.SetMotor(Nr.M3, Dir.Rechts, Speed.Full, 40)
Do
  lblPos.Text = ft.GetCounter(Nr.E6) & " - " & ft.GetCounter(Nr.E8)
Loop While ft.WaitForMotors(100, 4, 3) = Wait.Time
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehen mit voller Geschwindigkeit für 40 Impulse. Die Do .. Loop-Schleife wartet auf das Ende der Motoren (ft.WaitForMotors). Alle 100 MilliSekunden wird in der Schleife die aktuelle Position angezeigt ( 100 ... = Wait.Time). Wenn die Position erreicht ist (<> Time), ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet. Achtung hier wurde nicht auf NotHalt, oder ESC abgefragt, es könnte also auch vor Erreichen der Zielposition abgebrochen worden sein. In der Schleife wird auf Label lblPos die aktuelle Position angezeigt.

## WaitForPositionDown

Warten auf Erreichen einer vorgegebenen Position durch Abwärtszählen von der aktuellen

ft.**WaitForPositionDown**(InputNr, ByRef Counter, Position, Optional TermInputNr)

Ausgegangen wird von der aktuellen Position, die in Counter gespeichert ist, es werden solange Impulse von Counter abgezogen, bis der in Position angegebenen Stand erreicht ist. Counter enthält zusätzlich die dann tatsächlich erreichte position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = True beendet. Counter und Position mpssen immer positive Werte (einschl. 0 ) enthalten.

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForPositionUp, WaitForChange

### Beispiel

```
Dim Zaehler%
Zaehler = 12
ft.SetMotor(Nr.M1, Dir.Links)
ft.WaitForPositionDown(Nr.E2, Zaehler, 0, Nr.E1)
```

```
ft.SetMotor(Nr.M1, Dir.Aus)
```

Die aktuelle Position ist 12 (Zaehler), der Motor an M1 wird linksdrehend gestartet. WaitForPositionDown wartet dann auf Erreichen der Position 0, der Motor wird dann ausgeschaltet. Wenn vorher E1 = True wird, wird ebenfalls abgeschaltet.

## WaitForPositionUp

Warten auf Erreichen einer vorgegebenen Position durch Aufwärtszählen von der aktuellen.

ft.**WaitForPositionUp**(InputNr, ByRef Counter, Position, Optional TermInputNr)

Ausgegangen wird von der aktuellen Position in Counter, es werden solange Impulse auf Counter aufaddiert, bis der in Position angegebene Stand erreicht ist. Counter enthält zusätzlich die dann tatsächlich erreichte Position (kann einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = True beendet. Counter und Position müssen immer positive Werte (einschl. 0) enthalten.

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForPositionDown, WaitForChange

### Beispiel

```
Dim Zaehler%  
Zaehler = 0  
ft.SetMotor(Nr.M1, Dir.Rechts)  
ft.WaitForPositionUp(Nr.E2, Zaehler, 24)  
ft.SetMotor(Nr.M1, Dir.Aus)
```

Die aktuelle Position ist 0 (Zaehler), der Motor an M1 wird rechtsdrehend gestartet. WaitForPositionUp wartet dann auf Erreichen der Position 24, der Motor wird dann ausgeschaltet. Siehe auch Beispiel zu WaitForPositionDown, hier wird die Gegenrichtung gefahren.

## WaitForTime

Anhalten des Programmablaufes für mSek MilliSekunden.

ft.**WaitForTime**(mSek)

Synonym für Pause

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : Pause

### Beispiel

```
Do  
ft.SetMotors(&H1)  
ft.WaitForTime(555)  
ft.SetMotors(&H4)  
ft.WaitForTime(555)  
Loop Until ft.Finish()
```

In der Schleife Do .. Loop Until ft.Finish() wird erst M-Ausgang (Lampe) M1 eingeschaltet und alle anderen abgeschaltet (binär : 0001), dann gewartet, M2 (Lampe) eingeschaltet (Rest aus, binär 0100) und gewartet. Ergebnis ein Wechselblinker. Ende der Schleife durch ESC-Taste.

## Anmerkungen

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 zugehörigen **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 (Aus) 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 Programmablauf an, bis das Waitziel (Ablauf Zeit, erreichte Position, Tasterstellung ...) erreicht ist d.h. sie synchronisieren den Programmablauf wieder.

Die längerlaufenden Methoden sind abbrechbar. Das geschieht manuell durch Drücken der ESC-Taste oder im Programm durch Setzen der Eigenschaft NotHalt = True (z.B. über einen Button).

Bei der Beschreibung der Methoden wird das unter dem Stichwort Exception angegeben.

---

## Klasse FishRobot

Die Klasse FishRobot ist von FishFace abgeleitet und bietet zusätzlich zu den FishFace Eigenschaften und Methoden eine Reihe von Methoden, die speziell für den Betrieb von Robots des Typs "Industry Robot" geeignet sind. Charakteristika : Der Antriebsmotor treibt auf einer geeigneten Welle noch ein zusätzliches Impulsrad an, das einen Taster betätigt. Die Schaltvorgänge (Einschalten, Ausschalten separat) werden ab Robot Null gezählt. Das Robot Null wird durch einen weiteren Taster markiert, der bei einer Linksdrehung (Dir.Links) vom Robot angefahren wird. Die Taster sind dem M-Ausgang fest zugeordnet (M1 : E1 Endtaster, E2 Impulszähler). Siehe auch Abschnitt Anmerkungen | Rob-Funktionen.

## Test-Programmrahmen

Erstellung des Projekts wie bei FishFace als Windows.Forms Projekt. Die Source sieht in Anlehnung an FishFace wie folgt aus :

```
Option Explicit On
Option Strict Off
Imports FishFa30

" Vom Windows Form Designer generierter Code "
Dim WithEvents ft As New _
    FishRobot(New Integer(,) {{3, 222}, {4, 88}})

Private Sub cmdAction_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdAction.Click
    Try
        ft.OpenInterface(Port.COM2)
        lblStatus.Text = "Home Position E5 / E7 drücken"
        ft.MoveHome()
        ft.MoveTo(34, 56)
        lblStatus.Text = "Finito"
    Catch eft As FishFaceException
        lblStatus.Text = eft.Message
    Finally
        ft.CloseInterface()
    End Try
End Sub

Private Sub ft_PositionChange(ByVal sender As Object, _
    ByVal ActPositions() As Integer) Handles ft.PositionChange
    lblStatus.Text = ActPositions(0).ToString() & " - " & _
        ActPositions(1).ToString()
End Sub
```

Verändert hat sich die Instanzierung, jetzt mit FishRobot und der Liste der beteiligten Motoren an M3 und M4 mit einer Fahrstrecke von max 222 bzw. 88 Impulsen. Zusätzlich wird mit WithEvents angegeben, daß Ereignisroutinen verwendet werden sollen und die Ereignisroutine ft\_PositionChange selber. In der Ereignisroutine wird die aktuelle Robot-Position angezeigt.

Die Ereignisroutine ist nicht zwingend.

## Konstruktor

**FishRobot**(MotList As Integer(,))

MotList : Integer(,) Liste der für den Robot-Betrieb eingesetzten Motoren. Jeweils Nummer des M-Ausganges und max. Fahrweg ab Endtaster in Impulsen.

Mit default-Werten für AnalogScan = False, Slave = False,

PollInterval = 0, LPTAnalog = 0, LPTDelay = 0

```
Dim WithEvents ft As New FishRobot(New Integer(,){{3,123},{4,456}})
```

oder :

```
Dim MotList(,) As Integer = {{3, 123}, {4, 456}}
```

```
Dim WithEvents ft As New FishRobot(MotList)
```

In beiden Fällen werden die Motoren an M3 (mit E5 Endtaster, E6 Impulstaster) und M4(mit E7 Endtaster, E8 Impulstaster) in den Robbetrieb einbezogen. Der Fahrweg beträgt 123 bzw. 456 Impulse ab Endtaster. Zu beachten ist, daß die anzufahrenden Positionen bei den Methoden MoveTo/MoveDelta in dieser Reihenfolge anzugeben sind.

**FishRobot**(MotList As Integer(,) , AnalogScan As Boolean, Slave As Boolean, \_  
PollInterval As Integer)

MotList : wie oben

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTAnalog = 0, LPTDelay = 0

**Fishrobot**(MotList As Integer(,) , AnalogScan As Boolean , Slave As Boolean , \_  
PollInterval As Integer, LPTAnalog As Integer, LPTDelay As Integer)

MotList : wie oben

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTAnalog = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTDelay = 0 Bestimmung durch OpenInterface, >0 wird übernommen

AnalogScan : mit/ohne Auswerten der Analog-Eingänge EX/EY

Slave : mit/ohne Slave/Extension Module

PollInterval : Zeitintervall in MilliSekunden in dem das Interface abgefragt und upgedated werden soll. Wert = 0 : Der Wert wird intern beim OpenInterface bestimmt. Der aktuelle Wert kann (nach OpenInterface) mit der Eigenschaft PollInterval abgefragt werden und ggf. bei der nächsten Instanzierung – vorsichtig – verändert werden. ACHTUNG zu kleine Werte führen zum "Aufhängen" des Rechners.

LPTAnalog : Skalierfaktor für Analogwerte.

LPTDelay : Ausgabeverzögerung bei Universal(LPT) Interface (Werte 50 – 1000)

LPTAnalog / LPTDelay werden nur der Vollständigkeit halber aufgeführt, da sie nur auf Win9x-Systemen (mit Direktzugriff auf den LPT-Port) zum Einsatz kommen können. Und da gibt es kein .NET Framework.

## Eigenschaften

### MotCntl

Liste mit den Daten der bei der Instanzierung übergebenen Motordaten

ft.**MotCntl**(n).Nr Nummer des zugehörigen M-Ausganges

ft.**MotCntl**(n).maxPos Maximaler Fahrweg in Impulsen

ft.**MotCntl**(n).actPos Aktuelle Position ab Home (0) in Impulsen

Alle Werte vom Typ int. Zusammengefaßt in der struct MotWerte.

n bezieht sich auf die Motor-Position in der MotList der Instanzierung.

## Methoden

### MoveDelta

Simultanes Anfahren einer vorgegebenen Position relativ zur aktuellen.

ft.**MoveDelta**(DeltaList() As Integer)

DeltaList : Liste der anzufahrenden Positionen gezählt ab der aktuellen Positionen, bei negativen Werten wird die Fahrrichtung umgekehrt. Die Werte können als Aufzählung einzelner Werte oder alternativ als Array angegeben werden. Während der jeweils letzten 6 Impulse wird gebremst.

Exception : InterfaceProblem; DoEvent, Abbrechbar

Ereignis : PositionChange.

Siehe auch : MoveTo

Beispiel

```
Dim ft As New FishRobot(New Integer(,){3,234}, {4,123});  
.....  
ft.MoveDelta(34, -12);
```

Der Motor an M3 fährt 34 Impulse nach rechts, der Motor an M4 fährt 12 Impulse nach links. Alternative Schreibweise siehe Beispiel MoveTo

### MoveHome

Simultanes Anfahren der Home Position

ft.**MoveHome**()

Betroffen sind die Motoren, die bei der Instanziierung angegeben wurden. Es wird nach links (Dir.Links) gefahren, bis der zugehörige Endtaster erreicht wird. Die erreichte Position wird auf 0 gesetzt.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Beispiel : siehe MoveTo

### MoveTo

Simultanes Anfahren einer vorgegebenen Position bezogen auf die Home Position.

ft.**MoveTo**(PosList() As Integer)

PosList : Liste der anzufahrenden Positionen gezählt ab Home Position (zug. Endtaster bei Dir.Links). Wertebereich 0 – max. (Angabe bei der Instanziierung). Die Werte können als Aufzählung einzelner Werte oder alternativ als Array angegeben werden. Während der jeweils letzten 6 Impulse wird gebremst.

Exception : InterfaceProblem; DoEvent, Abbrechbar

Ereignis : PositionChange.

Siehe auch : MoveDelta

Beispiel

```
Dim ft As New FishRobot(New Integer(,){3,234}, {4,123});  
Dim PosList() As Integer = {100, 200};  
.....  
ft.MoveHome();  
ft.MoveTo(PosList);
```

Der gesamte Robot fährt mit den Motoren an M3 und M4 zunächst die Home Position an und dann die durch MoveTo vorgegebene. Der Motor an M3 fährt auf Position 100, der an M4 auf Position 123, da er bei Erreichen der max. zul. Position gestoppt wird.

Alternative Schreibweise :

```
Dim ft As New FishRobot(New Integer(,){3,234}, {4,123});  
.....  
ft.MoveTo(100, 200);
```

Schreibweise 1 ist sinnvoll, wenn die Werte bestehenden Tabellen entnommen werden können (Abarbeiten einer Instruktionsliste). Schreibweise 2, wenn Einzelwerte vorliegen (z.B. beim TeachIn).

## Ereignisse

### PositionChange

Aufruf bei einer Veränderung der Position eines Motors durch die Methoden MoveTo / MoveDelta (Bei der Instanzierung ist ein WithEvents erforderlich)

Beispiel

```
Dim WithEvents ft As New FishRobot(New Integer(,){3,222}, {4,88});  
.....  
Private Sub ft_PositionChange(ByVal sender As Object, _  
    ByVal ActPositions() As Integer) Handles ft.PositionChange  
    lblStatus.Text = ActPositions(0).ToString() & " - " &  
        ActPositions(1).ToString()  
End Sub
```

sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt. ActPositions eine Liste mit den aktuellen Positionen der laut Instanzierung übergebenen MotList. Die Positionen zählen in Impulsen ab Home positiv. Die Ereignisroutine wird über die Auswahllisten des Editorfensters ausgewählt.

---

## Klasse FishStep

Die Klasse FishStep ist von FishFace abgeleitet und bietet zusätzlich zu den FishFace Eigenschaften und Methoden eine Reihe von Methoden, die speziell für den Betrieb von Schrittmotoren geeignet sind. Dabei wird zwischen dem Betrieb einzelner Schrittmotoren (Anschlußbelegung : zwei aufeinanderfolgende M-Ausgänge, Methoden Step...) und dem Simultanbetrieb zweier zusammenhängender Schrittmotoren im XY-Verbund (Anschlußbelegung : drei aufeinanderfolgende M-Ausgänge für die beiden Motoren, Methoden Plot) unterschieden. Die Positionierung erfolgt in Zyklen (in der Regel vier Schaltvorgänge von 7,5°). Zu den M-Ausgängen gehören fest vorgegeben E-Eingänge zur Feststellung der Home Position. Dazu siehe auch "Anmerkungen zu den Step-Funktionen" am Ende des Dokumentes.

## Test-Programmrahmen

Erstellung des Projekt wie bei FishFace als Windows.Forms Projekt. Die Source sieht (in Anlehnung an FishFace) wie folgt aus :

```
Option Explicit On
Option Strict Off
Imports FishFa30
```

" Vom Windows Form Designer generierter Code "

```
Dim WithEvents ft As New _
    FishStep(New Integer(,) {{1, 456}, {3, 456}})
```

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdAction.Click
    Try
        ft.OpenInterface(Port.COM2)
        lblStatus.Text = "ManRefStep gestartet"
    Catch eft As FishFaceException
        lblStatus.Text = eft.Message
    Finally
        ft.CloseInterface()
    End Try
End Sub
```

```
Private Sub ft_StepChange(ByVal sender As Object, _
    ByVal MotNr As Integer, ByVal ActPosition As Integer) _
    Handles ft.StepChange
    lblStatus.Text = "Akt. Step-Position : " & ActPosition.ToString()
End Sub
```

```
Private Sub ft_PlotChange(ByVal sender As Object, _
    ByVal MotNr As Integer, _
    ByVal Xact As Integer, _
    ByVal Yact As Integer) Handles ft.PlotChange
    lblStatus.Text = "Akt. Plot-Position : " & Xact.ToString() _
        & " / " & Yact.ToString()
End Sub
```

Verändert hat sich die Instanzierung, jetzt mit FishStep und der Liste der beteiligten Motoren an M1/M2 (Endtaster E1) und M3/M4 (Endtaster E5) bei Nutzung durch Step-Methoden bzw. M1/M2 (Endtaster E1) und M3/M1 (Endtaster E5) bei Nutzung durch Plot-Methoden. Jeweils mit einer Fahrstrecke von 456 Zyklen.

Der gezeigte Testrahmen läßt nur den alternativen Betrieb mit Step- bzw. Plot-Methoden zu. Dementsprechend können die jeweils nicht benötigten Ereignisse entfallen. Bei Einsatz eines Extension-Modules (Slave) können Step- und Plot-Methoden aber gemeinsam genutzt werden, wenn sie bei der Instanziierung entsprechend auf die Interfaces verteilt werden.

Die Ereignisroutinen können auch ganz entfallen.

## Konstruktor

**FishStep**(MotList As Integer(,))

MotList : Liste der für den Step-Betrieb eingesetzten Motoren. Jeweils Nummer des M-Ausganges und max. Fahrweg ab Endtaster in Zyklen.

Mit default-Werten für AnalogScan = False, Slave = False,

PollInterval = 0, LPTAnalog = 0, LPTDelay = 0

```
Dim ft As New FishStep(New Integer(,){{1,123},{3,456}})
```

oder :

```
Dim MotList(, ) As Integer = {{1,123},{3,456}}
```

```
Dim WithEvents ft As New FishStep(MotList)
```

In beiden Fällen werden die Motoren an M1/M2 (Endtaster E1) und M3/M4(Endtaster E5) in den Betrieb mit Step-Methoden einbezogen. Der Fahrweg beträgt 123 bzw. 456 Zyklen ab Endtaster. Beim Betrieb mit Plot-Methoden im XY-Verbund werden die Motoren an M1/M2 (Endtaster E1) und M3/M1(Endtaster E5) in den Betrieb einbezogen. Eine fließende Verteilung auf Interface und Extension Module ist möglich.

**FishStep**(MotList As Integer(,), AnalogScan As Boolean, Slave As Boolean, PollInterval As Integer)

MotList : wie oben

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTAnalog = 0, LPTDelay = 0

**FishStep**(MotList(,) As Integer, AnalogScan As Boolean, Slave As Boolean, PollInterval As Integer, LPTAnalog As Integer, LPTDelay As Integer)

MotList : wie oben

PollInterval = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTAnalog = 0 Bestimmung durch OpenInterface, >0 wird übernommen

LPTDelay = 0 Bestimmung durch OpenInterface, >0 wird übernommen

AnalogScan : mit/ohne Auswerten der Analog-Eingänge EX/EY

Slave : mit/ohne Slave/Extension Module

PollInterval : Zeitintervall in MilliSekunden in dem das Interface abgefragt und upgedated werden soll. Wert = 0 : Der Wert wird intern beim OpenInterface bestimmt. Der aktuelle Wert kann (nach OpenInterface) mit der Eigenschaft PollInterval abgefragt werden und ggf. bei der nächsten Instanziierung – vorsichtig – verändert werden. ACHTUNG zu kleine Werte führen zum "Aufhängen" des Rechners.

LPTAnalog : Skalierfaktor für Analogwerte.

LPTDelay : Ausgabeverzögerung bei Universal(LPT) Interface (Werte 50 – 1000)

LPTAnalog / LPTDelay werden nur der vollständigkeithalber aufgeführt, da sie nur auf Win9x-Systemen (mit Direktzugriff auf den LPT-Port) zum Einsatz kommen können. Und da gibt es kein .NET Framework.

## Eigenschaften

### MotCntl

Liste mit den Daten der bei der Instanzierung übergebenen Motordaten

- ft.**MotCntl**(n).maxPos    Maximaler Fahrweg in Zyklen (Integer)
- ft.**MotCntl**(n).actPos    Aktuelle Position ab Home (0) in Zyklen (Integer)
- ft.**MotCntl**(n).outPos    Angabe, ob einer der Motoren die actPos überschritten hat  
nur bei PlotTo/PlotDelta (Boolean)

n bezieht sich auf die Motor-Position in der MotList der Instanzierung.

## Methoden

### PlotDelta

Fahren eines Motorenpaares im XY-Verbund um Xrel / Yrel Zyklen bezogen auf die aktuelle Position

ft.**PlotDelta**(MotNr, Xrel As Integer, Yrel As Integer)

MotNr : Nummer (Nr. oder Integer) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.

Xrel / Yrel : Increment in Zyklen. Positive Werte rechtsdrehend (weg vom Endtaster, begrenzt durch den Wert der maxPos), negative linksdrehend (in Richtung Home Position / Endtaster).

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Ereignis : PlotChange.

Siehe auch : PlotTo

Beispiel :

```
ft.PlotDelta(Nr.M1, 100, -50)
```

Von der aktuellen Position wird um 100 Zyklen in X- und um 50 Zyklen (hin zum Endtaster) Y-Richtung gefahren.

### PlotHome

Anfahren der HomePosition für einen XY-Verbund von zwei Schrittmotoren.

ft.**PlotHome**(MotNr)

MotNr : Nummer (Nr. oder Integer) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.

Gefahren wird in Richtung der zugeordneten Endtaster. Nach Erreichen der Endtaster wird um zwei Zyklen in Gegenrichtung freigefahren.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Beispiel :

```
Dim ft As New FishStep(New Integer(,){{1,123},{3,456}})  
ft.Home(Nr.M1)
```

Bei der Instanzierung werden die M-Ausgänge, die von den Motoren genutzt werden sollen, und die max. Fahrwege angegeben. Hier wird der X-Motor an M1/M2 (Endtaster E1, Fahrweg 123 Zyklen) und der Y-Motor an M3/M1(Endtaster E5, Fahrweg 456 Zyklen) angeschlossen. Anschließend wird auf die Home Position gefahren.

## PlotTo

Fahren eines Motorenpaares im XY-Verbund auf die Position Xabs / Yabs.

ft.**PlotTo**(MotNr, Xabs As Integer, Yrel As Integer)

MotNr : Nummer (Nr. oder int) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.

Xabs / Yabs : Position ab Home Position (0) in Zyklen. Begrenzung durch Endtaster.

Exception : InterfaceProblem; DoEvents, Abbrechbar.

Ereignis : PlotChange

Beispiel :

```
ft.PlotTo(Nr.M1, 150, 333)
```

Gefahren wird auf Position 123 / 273, wenn die Instanzierung (max 123 / 456)des Beispiels von PlotHome angenommen wird. Die Fahrwegbegrenzung hat hier also zugeschlagen. Der Fahrbefehl wird mit Erreichen von Xmax abgebrochen, daraus ergibt sich dann die erreichte Y-Position (123/150 \* 333 = 273).

## StepDelta

Fahren eines einzelnen Schrittmotors um Xabs Zyklen bezogen auf die aktuelle Position

ft.**StepDelta**(MotNr, Xabs As Integer)

MotNr : Nummer (Nr. oder Integer) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Bei positiven Werten wird weg vom Endtaster gefahren bei negativen Werten hin zum Endtaster. Fahrwegbegrenzung bzw. Endtaster werden beachtet.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Ereignis : StepChange.

Beispiel :

```
ft.StepDelta(Nr.M5, 123)
```

Der Schrittmotor an M5/M6 fährt von der aktuellen Position rechtsdrehend (weg vom Endtaster E9) 123 Zyklen.

## StepHome

Anfahren der Home Position des über MotNr angegebenen Schrittmotors.

ft.**StepHome**(MotNr)

MotNr : Nummer (Nr. oder Integer) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Gefahren wird in Richtung des zugeordneten Endtasters (E9).

Exception : InterfaceProblem; DoEvents, Abbrechbar:

Beispiel :

```
Dim ft As New FishStep(new int[],{{1,123},{3,456}})
ft.StepHome(Nr.M3)
```

Der Motor an M3/M4 wird in Richtung des Endtasters (E9) gefahren, die aktuelle Position wird auf 0 gesetzt.

## StepTo

Fahren eines einzelnen Schrittmotors auf Position Xabs.

ft.**StepTo**(MotNr, Xabs As Integer)

MotNr : Nummer (Nr. oder Integer) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Endtaster und Fahrwegbegrenzung werden beachtet.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Ereignis : StepChange.

Beispiel :

```
Const mAufzug = 1  
ft.StepTo(mAufzug, 123)
```

Der Motor an M1/M2 fährt auf Position 123.

## Ereignisse

### PlotChange

Aufruf bei einer Veränderung der Position des Motorenpaars des XY-Verbundes durch die Methoden PlotTo/PlotDelta.

Beispiel :

```
Dim WithEvents ft As New FishStep(New Integer(,){1,123},{3,456})  
....  
Private Sub ft_PlotChange(ByVal sender As Object, _  
    ByVal MotNr As Integer, _  
    ByVal Xact As Integer, _  
    ByVal Yact As Integer) Handles ft.PlotChange  
    lblStatus.Text = "Akt. Plot-Position : " & Xact.ToString() _  
        & " / " & Yact.ToString()  
End Sub
```

sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt, MotNr die Nummer des auslösenden Motors, Xact / Yact die aktuelle Position des XY-Verbundes.

### StepChange

Aufruf bei einer Veränderung der Position eines einzelnen Schrittmotors durch die Methoden StepTo/StepDelta.

Beispiel :

```
Dim WithEvents ft As New FishStep(New Integer(,){1,123},{3,456})  
....  
Private Sub ft_StepChange(ByVal sender As Object, _  
    ByVal MotNr As Integer, _  
    ByVal ActPosition As Integer) Handles ft.StepChange  
    lblStatus.Text = "Akt. Step-Position : " & ActPosition.ToString()  
End Sub
```

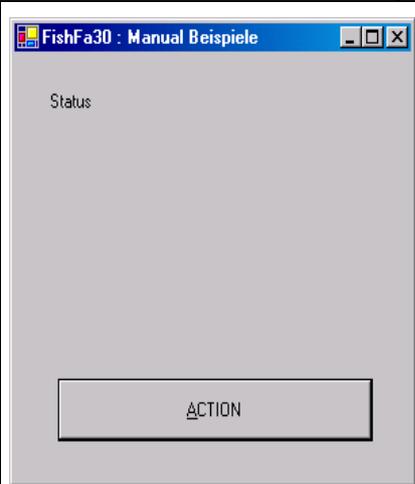
sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt, MotNr die Nummer des auslösenden Motors, ActPosition die aktuelle Position des Schrittmotors.

Die Ereignisroutine wird über ie Auswahllisten des Editorfensters ausgewählt.

# Tips & Tricks

## Programmrahmen

Die im Kapitel Techniken angeführten Programmausschnitte benötigen einen Programmrahmen innerhalb dessen sie ablaufen können. Er wird im Kapitel Techniken dann nicht mehr extra angegeben.

	<pre>Option Explicit On Option Strict Off Imports FishFa30  Public Class frmMain     Inherits System.Windows.Forms.Form      " Vom Windows Form Designer generierter Code "      Dim ft As New FishFace(True, False, 0)      Private Sub cmdAction_Click( _         ByVal sender As System.Object, _         ByVal e As System.EventArgs) _         Handles cmdAction.Click         ' --- Beispiel Deklarationen ---         Try             ft.OpenInterface(Port.COM2)             ' --- Beispiel Code ---         Catch eft As FishFaceException             lblStatus.Text = eft.Message         Finally             ft.CloseInterface()         End Try     End Sub End Class</pre>
<p>Elemente :</p> <p>Label lblStatus</p> <p>Button cmdAction</p> <p>ACHTUNG : Der Programmrahmen unterscheidet sich ein wenig bei mit VB.NET bzw. SharpDevelop erstellen Forms. Hier die VS.NET Schreibweise. Zu SharpDevelop siehe Einführung.</p>	

Erstellung des Programmrahmens erfolgt wie gewohnt durch Anlegen eines neuen WindowsForm-Projektes. Zusätzlich ist dann ein Verweis auf FishFa30.DLL in der Projektmappe anzulegen. Auf diesen bezieht sich dann der `Import FishFa30`. Die `Option Explicit On` verlangt die Deklaration aller Variablen. Die `Option Strict Off` ermöglicht den Verzicht auf Typumwandlungs-Angaben (z.B. Integer -> String).

Mit `Dim ft As New FishFace(True, False, 0)` wird eine Instanz der Klasse `FishFace` der Assembly `FishFa30.DLL` angelegt. `True` : Zugriff auf die Analog-Eingänge EX / EY ist möglich, `False` : kein Extension Module, `0` : Standard PollInterval.

Für den Button `cmdAction` wird eine Click-Routine angelegt. Sie enthält mit `ft.OpenInterface(Port.COM2)` und `ft.CloseInterface` die Methoden, die die Verbindung zum Interface kontrollieren. Sie sind in einen `Try ... Catch ... Finally`-Block eingebettet um auftretende InterfaceProbleme abzufangen und in `lblStatus` anzuzeigen.

---

# Techniken

## Blinker/Schleife

Lampe an M1 blinkt im Sekundentakt :

```
Const mGelb = 1
Do
  ft.SetMotor(mGelb, Dir.Ein)
  ft.Pause(555)
  ft.SetMotor(mGelb, Dir.Aus)
  ft.Pause(444)
Loop Until ft.Finish()
```

Die Parameter für die FishFace Methoden sollten benannt werden. Für einige Standardwerte gibt es bereits Namen : Nr.M1 ... Nr.E16, Dir.Ein, Dir.Aus, Dir.Links, Dir.Rechts ... und Wait.Ende, Wait.Time, Wait.ESC für die Methode WaitForMotors. Weitere sollte man selber erfinden.

Meistens enthält das Programm ein große Schleife in der alle Befehle wiederholt durchlaufen werden. Hier ist das `Do .. Loop Until Finish()` : Die Methode Finish prüft, ob ein Abbruchwunsch vorliegt und meldet dann True zurück.

Beenden der Schleife durch ESC-Taste. Es kann auch zusätzlich ein E-Eingang angegeben werden : `ft.Finish(Nr.E8)`. Beendigung durch ESC-Taste oder E8.

## WechselBlinker

Lampen an M1 und M2 blinken im Wechsel.

Alternative 1 :

```
Do
  ft.SetMotor(Nr.M2, Dir.Aus)
  ft.SetMotor(Nr.M1, Dir.Ein)
  ft.Pause(444)
  ft.SetMotor(Nr.M1, Dir.Aus)
  ft.SetMotor(Nr.M2, Dir.Ein)
  ft.Pause(444)
Loop Until ft.Finish()
```

Alternative 2 kompakter:

```
Do
  ft.SetMotors(&H1)
  ft.Pause(444)
  ft.SetMotors(&H4)
  ft.Pause(444)
Loop Until ft.Finish()
```

Hier werden alle M-Ausgänge gleichzeitig geschaltet. Jeweils zwei bit pro M-Ausgang. Also 00000001 für M1 Ein und 00000100 für M2 ein. Alle anderen Ausgänge sind Aus.

## Abfrage eines E-Einganges

Wenn E1 geschaltet ist lblStatus "---EIN---" sonst "---AUS---" :

```
If ft.GetInput(Nr.E1) Then
    lblStatus.Text = "--- EIN ---"
Else
    lblStatus.Text = "--- Aus ---"
End If
```

Man kann auch Vergleichen :

```
If ft.GetInput(Nr.E1) = False Then lblStatus.Text = "off"
```

## Warten auf einen E-Eingang

Wenn E1 geschlossen ist, wird in lblStatus "--- Es geht los ---" ausgegeben :

```
lblStatus.Text = "--- Zum Programmstart E1 drücken ---"
ft.WaitForInput(Nr.E1)
lblStatus.Text = "--- Es geht los ---"
```

## Anzeige des Status der E-Eingänge

Status von E1 :

```
lblStatus.Text = "E1 : " & ft.GetInput(Nr.E1)
```

Laufende Anzeige des Status aller E-Eingänge :

```
Do
    lblStatus.Text = Status der E-Eingänge : " & _
                    Hex(ft.GetInputs())
    ft.Pause(1234)
Loop Until ft.Finish()
```

## Analog-Anzeige

Laufende Anzeige der beiden Analog-Eingänge EX und EY :

```
Do
    lblStatus.Text = "Werte EX : " & ft.GetAnalog(Nr.EX) & _
                    " EY : " & ft.GetAnalog(Nr.EY)
    ft.Pause(1111)
Loop Until ft.Finish()
```

**ACHTUNG** : Hier muß beim Instanzieren das AnalogScan eingestellt werden (siehe Programmrahmen). Außerdem ist zu beachten, daß ein größeres PollInterval erforderlich ist. Hier wird ein Defaultwert eingesetzt. Kann man das nicht gebrauchen, kann man man GetAnalogDirect benutzen. Das greift jedesmal direkt auf das Interface zu (und hält das Programm solange an, der Parameter AnalogScan beim OpenInterface ist also nicht erforderlich) :

```
lblStatus.Text = "Wert von EY : ", ft.GetAnalog(Nr.EY)
```

## Fahren für eine bestimmte Zeit

Der Motor an M3 soll 3,5 Sekunden nach links laufen :

```
ft.SetMotor(Nr.M3, Dir.Links)
ft.Pause(3500)
ft.SetMotor(Nr.M3, Dir.Aus)
```

## Fahren zum Endtaster

Der Motor an M3 soll den Endtaster E5 anfahren und dann abschalten :

```
ft.SetMotor(Nr.M3, Dir.Links)
Do : Loop Until ft.GetInput(Nr.E5)
ft.SetMotor(Nr.M3, Dir.Aus)
```

Das Beispiel ist umständlich und nicht durch ESC-Taste abbrechbar.  
besser :

```
ft.SetMotor(Nr.E3, Dir.Links)
ft.WaitForInput(Nr.E5)
ft.SetMotor(Nr.M3, Dir.Aus)
```

## Fahren um eine vorgegebene Anzahl von Schritten

### WaitForChange

Motor an M3 mit Impulstaster an E6 soll um 12 Impulse fahren.

```
ft.SetMotor(Nr.M3, Dir.Links)
ft.WaitForChange(Nr.E6, 12)
ft.SetMotor(Nr.M3, Dir.Aus)
```

### WaitForPositionDown

Motor an M3 fährt von IstPosition = 12 auf ZielPosition = 0,  
Impulszählung an E6 in Richtung 0 (Endtaster = E5) :

```
Dim IstPosition% = 12
ft.SetMotor(Nr.M3, Dir.Links)
ft.WaitForPositionDown(Nr.E6, IstPosition, 0, Nr.E5)
```

Die tatsächlich erreichte Position (kann um einen Impuls von der Vorgabe abweichen) steht nach dem Vorgang in IstPosition. Wird der Endtaster E5 vorher True, wird vorzeitig abgebrochen.

### WaitForPositionUp

Motor an M3 fährt von IstPosition = 12 auf ZielPosition = 24,  
Impulszählung an E6 in Richtung weg von Endtaster :

```
Dim IstPosition% = 12
ft.SetMotor(Nr.M3, Dir.Rechts)
ft.WaitForPositionUp(Nr.E6, IstPosition, 24)
```

Die tatsächlich erreichte Position (kann um einen Impuls von der Vorgabe abweichen) steht nach dem Vorgang in IstPosition.

## WaitForMotors

Der Motor an M3 fährt für 12 Impulse an E6 mit verminderter Geschwindigkeit nach Links.

```
ft.SetMotor(Nr.M3, Dir.Links, Speed.L5, 12)
ft.WaitForMotors(0, 3)
```

Es wird gewartet, bis das Ziel erreicht wurde. Es geht auch ohne WaitForMotors, wenn das Programm anderweitig beschäftigt ist (Die Motoren schalten bei Erreichen der Zielposition selbsttätig ab). Siehe auch "Anmerkungen zu den Rob-Funktionen".

Zwei Motoren simultan mit laufender Positionsanzeige

Zwei Motoren (M3, M4) fahren simultan (gleichzeitig), die Impulszählung erfolgt an E6 und E8 (siehe auch Rob-Funktionen). Parallel dazu wird die aktuelle Position angezeigt :

```
ft.SetMotor(Nr.M3, Dir.Links, Speed.Full, 121)
ft.SetMotor(Nr.M4, Dir.Rechts, Speed.L8, 64)
Do
  lblStatus.Text = "Position M3 - M4 : " & _
                  ft.GetCounter(Nr.E6) & " - " & _
                  ft.GetCounter(Nr.E8)
Loop While ft.WaitForMotors(300, 3, 4) = Wait.Time
lblStatus.Text = "Position M3 - M4 : " & _
                  ft.GetCounter(Nr.E6) & " - " & _
                  ft.GetCounter(Nr.E8) & "--- Final ---"
```

Motor M3 fährt mit voller Geschwindigkeit um 121 Impulse nach Links

Motor M4 fährt mit halber Geschwindigkeit um 64 Impulse nach Rechts

WaitForMotors wartet auf beide, alle 0,3 Sekunden wird die aktuelle Position angezeigt. Zum Schluß wird die tatsächlich erreichte Position angezeigt. Zur Positionsanzeige wird mit GetCounter die aktuelle Position ausgelesen.

## Lampen

Lampen werden meistens genauso behandelt wie Motoren (mit zwei Polen an einem M-Ausgang, z.B. `ft.SetMotor(1, ft.Ein)`), da sie aber nur ein oder ausgeschaltet werden können, ist auch die Schaltung an einem Pol eines M-Ausganges und Masse möglich man kann so bis zu acht Lampen an ein Interface anschließen :

```
ft.SetLamp(1, Dir.Ein)
ft.SetLamp(4, Dir.Ein)
ft.Pause(1000)
ft.SetLamp(1, Dir.Aus)
ft.SetLamp(4, Dir.Aus)
```

Die Lampen an Pin 1 und 4 (M1 vorn, M2 hinten) werden für 1 Sekunde eingeschaltet. Hinweis : vernünftig geht das nur mit dem parallelen Interface.

## Lichtschranken

### Warten auf Lichtschranke

Lampe an M1, Phototransistor an E1. Es wird auf eine Unterbrechung der Lichtschranke gewartet :

```
Const mLicht = 1, ePhoto = 1
ft.SetMotor(mLicht, Dir.Ein)
```

```
ft.Pause(555)
ft.WaitForInput(ePhoto, False)
```

Lampe wird eingeschaltet, danach 0,5 Sekunden Pause um den Phototransistor "anzuwärmen", dann wird auf eine Unterbrechung der Lichtschranke gewartet.

## Warten auf Einfahrt in eine Lichtschranke

Lampe an M1, Förderbandmotor an M3, Phototransistor an E1 :

```
Const mBand = 2, ePhoto = 1

ft.SetMotor(mBand, Dir.Link)
ft.WaitForLow(ePhoto)
ft.SetMotor(mBand, Dir.Aus)
```

Der Motor M1 läuft solange bis ein Teil auf dem Band in die vorher nicht unterbrochene Lichtschranke einfährt. Die Lichtschranke wurde bereits vorher eingeschaltet.

## Warten auf Ausfahrt aus einer Lichtschranke

Lampe an M1, Förderbandmotor an M3, Phototransistor an E1 :

```
Const mBand = 2, ePhoto = 1

ft.SetMotor(mBand, Dir.Link)
ft.WaitForHigh(ePhoto)
ft.SetMotor(mBand, Dir.Aus)
```

Der Motor M1 läuft solange bis ein Teil auf dem Band, das die Lichtschranke unterbricht, aus der Lichtschranke herausgefahren ist.

## Gleichzeitiges Schalten aller M-Ausgänge

Mit SetMotors können alle M-Ausgänge mit einem Befehl geschaltet werden. Dazu muß der Parameter MotorStatus entsprechend besetzt werden. Im MotorStatus sind pro M-Ausgang jeweils 2bit reserviert : 00 00 00 00 (bei Einsatz des Extension Modules nochmal 4). 00 bedeutet ausgeschaltet, 01 Drehrichtung links bzw. Ein, 10 Drehrichtung rechts. 00 01 00 00 demnach M3 links und 01 00 00 00 M4 links.

## Einfache Ampel

Ein einfaches Ampelspiel sieht so aus : Grün – Gelb – Rot – RotGelb.  
Die Lampen dazu M1 : Grün, M2 : Gelb, M3 : Rot und die Konstanten dazu :  
mGruen = 00 00 00 01, mGelb = 00 00 01 00, mRot = 00 01 00 00,  
dezimal = 1, 4, 16.

```
Const mGruen = 1, mGelb = 4, mRot = 16

Do Until ft.Finish()
  ft.SetMotors(mGruen)
  ft.Pause(1000)
  ft.SetMotors(mGelb)
  ft.Pause(250)
  ft.SetMotors(mRot)
  ft.Pause(1000)
  ft.SetMotors(mRot+mGelb)
  ft.Pause(250)
Loop
```

## Listengesteuerte Ampel

Wenn man einen festen Ampeltakt vorgibt, kann man den Ablauf auch listengesteuert machen :

```
Const mGruen = 1, mGelb = 4, mRot = 16
Dim Phase() As Integer = {mGruen, mGruen, mGruen, mGruen, _
    mGelb, mRot, mRot, mRot, mRot, mRot+mGelb}
Dim p%

Do Until ft.Finish()
    For Each p in Phase
        ft.SetMotors(p)
        ft.Pause(250)
    Next p
Loop
```

Hier wird mit einer festen Taktung von 250 MilliSekunden gearbeitet. Das Verfahren lohnt bei komplexeren Steuerungen.

---

## Betrieb eines Robots

Die Klasse FishRobot ist speziell auf den Betrieb von Robot-Motoren ausgerichtet. Als Robot-Motor wird ein Motor dann bezeichnet, wenn auf einer Motorwelle ein Impulsrad mitläuft, das einen Taster betätigt über den die Umdrehungen der Motorwelle in Form von Impulsen gezählt werden. Hinzu kommt ein Endtaster zur Bestimmung der Home-Position. Endtaster und Impulstaster sind dem jeweiligen M-Ausgang fest zugeordnet. Außerdem kann der max. Fahrweg vorgegeben werden (s.a. Anmerkungen zu den Rob-Funktionen). Es können bis zu vier (mit Extension Module bis zu acht) Motoren simultan (gleichzeitig) betrieben werden.

Die Testroutine ist eine Windows.Form, sie entspricht der im Kapitel Programmrahmen angeführten, die Instanzierung wird auf FishRobot umgestellt.

## Robot-Fahren

Das Robot-Fahren geschieht über die Methoden MoveTo und MoveDelta. Als Parameter enthalten sie eine Liste der anzufahrenden Positionen (absolut von Home oder relativ zur aktuellen Position). Die Liste der zugehörigen M-Ausgänge und die Begrenzung wird bei der Instanzierung festgelegt. Die Home-Position (die Position an den Endtastern) wird mit MoveHome angefahren. Die Motoren müssen so gepolt sein, daß sie linksdrehen (Dir.Links). Bei Erreichen der Endposition werden die aktuellen Motorpositionen auf 0 gesetzt.

```
Dim ft As New FishRobot(New Integer(,){3,222},{4,88})
.....
Private Sub cmdAction_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdAction.Click
    Try
        ft.OpenInterface(Port.COM2)
        lblStatus.Text = "Home Position E5 / E7 drücken"
        ft.MoveHome()
        lblStatus.Text = "M3 auf Pos : " & _
            ft.MotCntl(0).actPos.ToString()

        ft.Pause(1234)
        ft.MoveTo(34, 56)
        ft.MoveDelta(-13, 6)
        ft.MoveTo(50, 80)
    Catch eft As FishFaceException
        lblStatus.Text = eft.Message
    Finally
        ft.CloseInterface()
    End Try
End Sub
```

Bei der Instanzierung wird die Roboter-Konfiguration festgelegt : Motoren an M3 und M4 mit Fahrwegbegrenzung auf 222 bzw. 88 Impulse.

Nach dem OpenInterface wird die Home-Position angefahren und die aktuelle Position auf 0 gesetzt. Das wird kurz angezeigt.

Anschließend wird auf die Position M3 = 23 und M4 = 34 gefahren. Dann wird M3 um 13 Impulse zurück auf Position 10 und M4 um 6 Positionen vor auf Position 40 gefahren. Zum Schluß wird die Position 50/80 angefahren.

## Positionsanzeige

Die aktuelle Position kann nach einer Move-Methode den entsprechenden Werten von MotCntl entnommen werden, wie im vorhergehenden Beispiel geschehen. Die aktuelle Position kann aber auch während der Ausführung der Methoden MoveTo/MoveDelta über eine Ereignis-Routine angezeigt werden.

Dazu ist die Instanzierung mit einer WithEvents Klausel zu sehen :

```
Dim MotList(,) As Integer = {{3, 123}, {4, 456}}
```

Die Ereignisroutine sieht dann so aus :

```
Private Sub ft_PositionChange(ByVal sender As Object, _  
    ByVal ActPositions() As Integer) Handles ft.PositionChange  
    lblStatus.Text = ActPositions(0).ToString() & " - " & _  
        ActPositions(1).ToString()  
End Sub
```

Die Ereignisroutine zeigt die aktuelle Position der beiden Motoren in lblStatus an. Natürlich könnten hier auch noch weitere Aufgaben wahrgenommen werden.

---

# Betrieb von Schrittmotoren

Die Klasse FishStep ist von FishFace abgeleitet und unterstützt zusätzlich den Einsatz von Schrittmotoren. Mit den Methoden StepHome / StepTo / StepDelta werden einzelne Schrittmotoren, die an zwei aufeinander folgende M-Ausgänge angeschlossen sind, unterstützt. Und mit den Methoden PlotHome / PlotTo / PlotDelta der Betrieb von zwei Schrittmotoren im XY-Verbund unterstützt. Die Schrittmotoren belegen 3 aufeinanderfolgende M-Ausgänge. Jedem Schrittmotor ist ein Endtaster fest zugeordnet.

Die hier verwendete Testroutine ist eine einfache Windows.Form, sie entspricht der im Kapitel Programmrahmen angeführten, die Instanziierung wird auf FishStep umgestellt.

## Einzelner Schrittmotor

Beispiel : Fahrstuhl aus einem Schrittmotor mit einer (langen) Schneckenwelle senkrecht nach oben und einem "Korb" an der Schneckenmutter. Der Endtaster liegt auf E1.

```
Dim WithEvents ft As New FishStep(New Integer(,){1, 456})
Const mFahrstuhl = 1
```

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles cmdAction.Click
    Try
        ft.OpenInterface(Port.COM2)
        lblStatus.Text = "Fährt in den Keller (Home)"
        ft.StepHome(mFahrstuhl)
        lblStatus.Text = "Fährt auf Etage 4"
        ft.StepTo(mFahrstuhl, 200)
        ft.Pause(1234)
        lblStatus.Text = "Fährt eine Etage tiefer"
        ft.StepDelta(mFahrstuhl, -50)
        lblStatus.Text = "Das war's"
    Catch eft As FishFaceException
        lblStatus.Text = eft.Message
    Finally
        ft.CloseInterface()
    End Try
End Sub
```

Und die Routine zur Positionsausgabe :

```
Private Sub ft_StepChange(ByVal sender As Object, _
    ByVal MotNr As Integer, _
    ByVal ActPosition As Integer) Handles ft.StepChange
    lblStatus.Text = _
        String.Format("Fahrstuhl auf Etage : {0} | {1:00}", _
            Int(ActPosition / 50), (ActPosition Mod 50))
End Sub
```

Bei der Instanziierung der Instanziierung wird der Fahrstuhlmotor mFahrstuhl (M1/M2) mit einem max. Fahrweg von 456 Zyklen der Instanz zugeordnet.

In der Klick-Routine für den ACTION-Button läuft das Steuer-Programm :

- Nach OpenInterface : Anfahren der Home Position (StepHome)
- Fahren zu Etage 4 (StepTo)
- Eine Etage tiefer mit StepDelta
- Laufende Anzeige der Position mit Routine ft\_StepChange.

## Zwei Motoren im XY-Verbund : Plotten

Plotter mit zwei Schrittmotoren an M1 – M3 und den Endtastern E1 und E5. Für Testzwecke reichen die nackten Motoren mit Scheibenrädern drauf, damit man etwas sehen kann.

Instanziierung :

```
Dim WithEvents ft As New FishStep( _  
                                New Integer(,) {{1, 456}, {3, 456}})  
Const mPlotter = 1
```

Das Steuer-Programm in der ACTION-Button Klickroutine :

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _  
                            ByVal e As System.EventArgs) Handles cmdAction.Click  
    Try  
        ft.OpenInterface(Port.COM2)  
        lblStatus.Text = "Fährt in linke untere Ecke (Home E1/E5)"  
        ft.PlotHome(mPlotter)  
        lblStatus.Text = "Anfahren 50/50"  
        ft.PlotTo(mPlotter, 50, 50)  
        lblStatus.Text = "Zeichnen eines Quadrats"  
        Vieleck(New Integer(,) {{50, 0}, {0, 50}, {-50, 0}, {0, -50}})  
        lblStatus.Text = "Das war's"  
    Catch eft As FishFaceException  
        lblStatus.Text = eft.Message  
    Finally  
        ft.CloseInterface()  
    End Try  
End Sub
```

Die laufende Positionsanzeige :

```
Private Sub ft_PlotChange(ByVal sender As Object, _  
                          ByVal MotNr As Integer, _  
                          ByVal Xact As Integer, _  
                          ByVal Yact As Integer) Handles ft.PlotChange  
    lblStatus.Text = _  
        String.Format("Akt. Plot-Position : {0} / {1}", Xact, Yact)  
End Sub
```

Zeichnen des Quadrats :

```
Private Sub Vieleck(ByVal RelList(,) As Integer)  
    Dim i%  
    For i = 0 To UBound(RelList)  
        ft.PlotDelta(mPlotter, RelList(i, 0), RelList(i, 1))  
    Next i  
End Sub
```

Diesmal sind die Kommentare in der Source.

# Anmerkungen zum Verständnis

---

## 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 internen 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 zugehörige 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) 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.

---

## Anmerkungen zu den Counters

Ein wesentliches Element zur Positionsbestimmung sind die Counter. Sie sind den E-Eingängen zugeordnet. In den Countern wird von einer zentralen Routine in umFish30.DLL jede Veränderung des Zustandes der E-Eingänge gezählt. Also z.B. das Öffnen oder auch das Schließen eines Tasters, der z.B. durch ein Impulsrad betätigt wird.

Die Counter sind Teil eines internen Kontrollblocks. Sie können mit entsprechenden Methoden gesetzt und abgefragt werden. Die Counter werden auch intern von einigen Funktionen/Methoden (z.B. SetMotor mit Parameter Counter und den meisten Wait-Methoden) genutzt, es kann also nicht damit gerechnet werden, daß sie über den Programmablauf Bestand haben.

---

## 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

Hier werden allgemeine Anmerkungen zu den Rob-Funktionen und deren Nutzung durch Methoden der Klasse FishFace gemacht. Die spezielle Klasse FishRobot wird separat beschrieben.

Die Rob-Funktionen laufen in einem besonderen 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 SetMotor oder alle gemeinsam über SetMotors geschaltet werden. Das Argument Counter gibt die Anzahl der zu fahrenden Impulse an. Die Argumente

ActPosition und ZielPosition beschreiben den Fahrauftrag. GetCounter /SetCounter greifen direkt auf den intern verwendeten Counter zu.

Die Motoren können auch alle mit einem Befehl geschaltet werden : SetMotors. Dazu müssen vorher die Parameter aufbereitet 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 : SetMotors(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. Die zugehörenden Counter sind vorher mit SetCounter 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 SetMotor 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 die Rob-Funktionen ausgelösten Aktionen wieder zu synchronisieren ist ein WaitForRobMotor(s) erforderlich.

---

## Anmerkungen zu den Step-Funktionen

Der Betrieb von Schrittmotoren über ein fischtechnik Interface ist möglich. Dazu ist die Klasse FishStep der Assembly FishFa30.DLL vorgesehen.

Schrittmotoren können mit FishStep **einzeln** oder im **XY-Verbund** paarweise betrieben werden. In beiden Fällen werden sie synchron betrieben, d.h. das Programm wartet, bis die vorgegebene Position erreicht ist. Im Falle des XY-Verbundes werden die beiden dazugehörenden Schrittmotoren simultan (gleichzeitig) betrieben.

Die Schrittmotoren erfordern zum Anschluß zwei aufeinanderfolgende M-Ausgänge (Einzel-Motoren) bzw. drei aufeinanderfolgende M-Ausgänge (XY-Verbund). Die M-Ausgänge können sich über Master und Slave (Extension Module) erstrecken. Der Betrieb erfolgt in Zyklen zu vier Schritten. Ein Zyklus ist damit auch die Positioniereinheit für einen Motor. Da die Motoren pro Schritt eine 7,5° Drehung machen, ergeben 48 Schritte eine volle Umdrehung. Das entspricht dann 12 Zyklen.

Diese Betriebsart wurde besonders in Hinblick auf die beschränkte Anzahl von M-Ausgängen am Interface gewählt. So ist ein Plotterbetrieb mit nur einem (Master) Interface möglich. Der freie M-Ausgang wird hier für den Stift-Antrieb genutzt. Da führt zu einem "Zittern" des Motors, der gerade nichts zu tun hat (Vor-/Rückschritt im Wechsel). Dieses Zittern stört den Betrieb aber nicht weiter, da es von dem üblichen Spiel (Schneckenantrieb) des Modells aufgefangen wird.

### Zeiten

Bei Schrittmotoren werden häufig Schnecken zum Modellantrieb genutzt. Die 'große' Schnecke hat eine Steigung von ca. 4,77 mm. d.h. bei einer Umdrehung der Motorwelle (mit der aufgesteckten Schnecke) legt die Schneckenmutter einen Weg von 4,77 mm zurück. Bei 12 Zyklen/Umdrehung sind das pro Zyklus 0,4 mm.

Bei einem Win2000 Rechner mit 1700 MHz und einem eingestellten PollInterval von 10 MilliSekunden ergeben sich folgenden Zeiten und Geschwindigkeiten :

200 Zyklen : 12,5 Sekunden. Pro Zyklus also 62.5 MilliSekunden.

100 mm Weg entsprechen 250 Zyklen. Das sind dann ca. 15 Sekunden für die 100 mm.

## Anschluß der Schrittmotoren

Die verwendeten Schrittmotoren haben vier Kabelanschlüsse : rot, grün, schwarz, grau.

### Zwei Schrittmotoren im XY-Verbund

		vorn	hinten
Motor X-Achse	Ma	rot	schwarz
	Mb	grün	grau
Motor Y-Achse	Ma	rot	schwarz
	Mc	grün	grau

vorn heißt die Stiftreihe an der Außenkante.

Mit Ma-Mc sind aufeinanderfolgende M-Ausgänge gemeint.

z.B. M1-M3. Aber M4-M6 sind auch möglich.

Die Motoren drehen bei PlotHome in Richtung 0 auf den zugehörigen Endtaster (Schließer, Kontakte 1 und 3). Für X ist der zu Ma gehörende, für Y der zu Mc gehörende z.B. M1 : E1, M3 : E5. (Alle von M1 : E1, E3, E5, E7, E9, E11, E13, E15 (für M8)).

### Ein einzelner Schrittmotor

		vorn	hinten
Motor A	Ma	rot	schwarz
	Mb	grün	grau

vorn heißt die Stiftreihe an der Außenkante.

Mit Ma-Mb sind aufeinanderfolgende M-Ausgänge gemeint.

z.B. M1-M2. Aber M4-M5 sind auch möglich.

Der Motor dreht bei StepHome in Richtung 0 auf den zugehörigen Endtaster (Schließer, Kontakte 1 und 3). Das ist der zu Ma gehörende.

z.B. M1 : E1 (Alle von M1 : E1, E3, E5, E7, E9, E11, E13, E15 (für M8)).