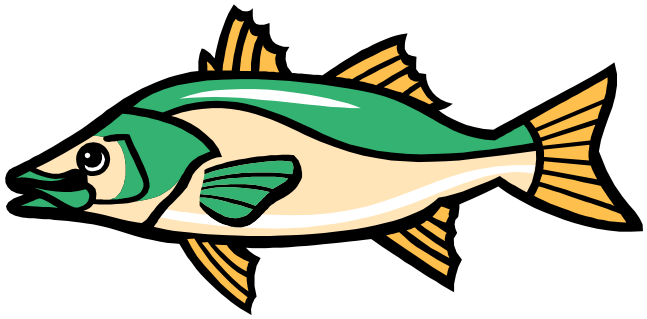
ftComputing

FishFa40AX.CLS

Handbuch zu Version 4.0

Ulrich Müller



Inhaltsverzeichnis

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

Dokumentname : FishFace40VB.DOC. Druckdatum : 10.06.2005
titelbild : Einfügen | Grafik | AusDatei | Office | Fisch12.WMF

Einführung

Allgemeines

FishFa40AX.CLS ist ein Visual Basic-Klassenmodul der als zentrale Klasse FishFace enthält. FishFa40AX basiert auf der in VC++ 6.0 geschriebenen umFish40.DLL. FishFa40AX selber liegt als Source und in übersetzter Form als ActiveX.DLL FishFa40AX.DLL vor.

Dies Dokument soll eine Einführung in die Programmierung von ftComputing-Modellen mit Visual Basic 6 und FishFace sein. Es berücksichtigt dabei besonders die Programmieranfänger. Einstiegsvoraussetzung ist der erfolgreiche Betrieb eines (selber) erweiterten "HelloWorld". Das heißt erste Kenntnisse werden vorausgesetzt. Es ist auch nicht beabsichtigt, Visual Basic Lehrbücher überflüssig zu machen.

Zum Einstieg empfiehlt sich ein Durcharbeiten des Dokumentes in der Reihenfolge der Kapitel :

- Installation mit vbFish40Setup.EXE
- Test der erfolgreichen Installation mit dem Interface Panel
- Erstes Projekt : StartAmpel.VBP mit eigenen Modifikationen.
- Aufbau des TestModells
- Durcharbeiten des Beispielprojektes mit eigenen Modifikationen. Bei LLWin-Erfahrung vielleicht parallel dazu auch eine Erprobung der LLWin-Beispiele.
    - Blinken : Programmschleife und der Abbruch, Analogwerte
    - Fahren : Warten auf I-Eingänge, Motorbetrieb
    - Warten : Das Warten geht weiter, Abbruch
    - Positionieren : Anfahren frei vorgegebener Positionen
- Bauen und Programmieren eigener Modelle
    - HändeTrockner : Templates, ActiveX.DLL
    - FußgängerAmpel : Listen mit Anweisungen abarbeiten
- Wenn Kasten Industry Robots vorhanden : Durcharbeiten der Robots-Beispiele
    - EinzelAktionen
    - Tätigkeitsablauf
    - Lernen, Speichern und Ausführen
- Weitere Modelle und FishFace-Erweiterungen können bei Bedarf konsultiert werden
    - Analog.CTL : Analog-Anzeige von EX und EY
    - KurveS40 : Erfassen von Meßwerten als Kurven
- Eine Referenz der Eigenschaften und Methoden bildet den Abschluß des Handbuchs.

Installation

Vorausgesetzt wird ein Windows System ab Windows 98 mit einem installierten Visual Basic Version 6 (ab Einstiegs Version).
Verwendet wurde Windows 2000 SR2 und Visual Basic 6 SP5.

Das Setup-Programm vbFish40Setup.EXE (www.ftcomputing.de/zip/vbfish40setup.exe) enthält alles was man zum Arbeiten mit FishFace benötigt
{app} gewählter Installationspfad
(default : C:\Programme\ftComputing), {sys} Windows\System-Verzeichnis :

- {app} : dieses Dokument (**FishFace40VB.PDF**, auch über das Start-Menü erreichbar) und ein **vbFish40.TXT** (ReadMe).
- {app} : Das Interface Panel **umFishDP40.EXE**
- {app} : die **FishFa40AX.DLL** ActiveX.DLL einschl. Helpdatei.
- {app}\FishFace40\VB6\ : die **FishFa40AX.CLS** Source
- {app}\FishFace40\VB6\Sample : das Beispielprojekt **FiFa40.VBP**
- {app}\FishFace40\LLWin30: die **LLWin 3.0** Beispiele dazu
- {app}\Templates40\VB6 : Das Template **ftComputing40.VBP**
- {app}\Modelle40\VB6 : die **Modellprogramme**, begonnen mit der **StartAmpel.VBP**
- {sys} : **umFish40.DLL**

Das Setup-Programm läuft, wie üblich, weitgehend automatisch. Installationspfad, anzulegende Desktop-Icons, Einträge ins Start-Menü können gewählt werden. Ebenso die Installation eines Treibers für das (alte) Universal Interface an einem LPT-Port.

Eine Deinstallation kann über Start | Einstellungen | Systemsteuerung | Software erfolgen.

Umstellung von FishFa30.DLL auf FishFa40AX.DLL

Eine Umstellung vorhandener FishFa30-Programme auf FishFa40AX ist erforderlich, wenn man ein ROBO Interface an USB betreiben will.

Intelligent Interfaces können ebenfalls mitFishFa40AX.DLL betrieben werden, eine Umstellung lohnt sich aber nur, wenn sie alternativ zu ROBO Interfaces betrieben werden sollen. Neue Eigenschaften für die Intelligent Interface werden nicht angeboten.

Das Universal Interface (an LPT) wird nicht mehr unterstützt, hier muß also FishFa30.DLL weitergenutzt werden.

Die Umstellung selber ist in den meisten Fällen recht einfach .

- Ändern des Verweises auf FishFa30 in FishFa40AX.DLL
  Menü Projekt Verweise : FishFace : FishFa40AX.DLL markieren.
- Umstellen der OpenInterface Methode auf OpenInterfaceUSB bzw. OpenInterfaceCOM
- Bei Verwendung von Enums sind die ftiNr Enums auf ftiOut und ftiInp umzustellen.
- Die Namen der Digital-Eingänge haben sich von 'E' auf 'I' geändert,
- Die Namen der Analog-Eingänge von EX / EY auf AX / AY. Außerdem wurde die numerische Bezeichnung von 0 / 1 auf 1 / 2 (5) geändert.

Neu hinzugekommen

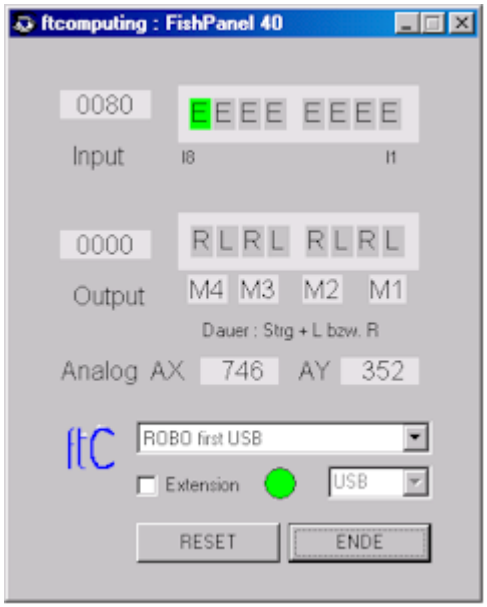
sind die Spannungs-Eingänge A1, A2, AV am Interface (Methode GetVoltage),

die Möglichkeit das Interface mit dem IR-Sender zu steuern (Methoden GetIRKey, FinishIR und WaitForInputIR)

sowie die Möglichkeit das Interface drahtlos ober das ROBO RF Datalink zu betreiben. Diese Funktion kann ein Download in das Interface (das mit FishFa40AX.DLL nicht geht) ersetzen und bietet gleichzeitig eine Bedienoberfläche über VB-Forms.

Bei der Methode SetMotors haben sich die Parameter ModeStatus geändert (nur noch 2bit pro Motor) und der Parameter SpeedStatus16 ist hinzugekommen (kann auf 0 gesetzt werden, wenn man keine 2. oder 3. Extension angeschlossen hat) .

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 kann die Verbindung zum Interface eingestelllt werden :

- zuerst ist das Interface auszuwählen. Bei ROBO Interface an USB gibt es nur die Möglichkeit : erstes ROBO Interface an USB. Beim Betrieb über einen COM-Port kann das Interface und dessen Betriebsart gewählt werden. zusätzlich kann angegeben werden, ob ein Extensionmodule angezeigt werden soll.

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

Die Input-Zeile zeigt den Status aller I-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 (ftiLinks, ftiLeft) 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.

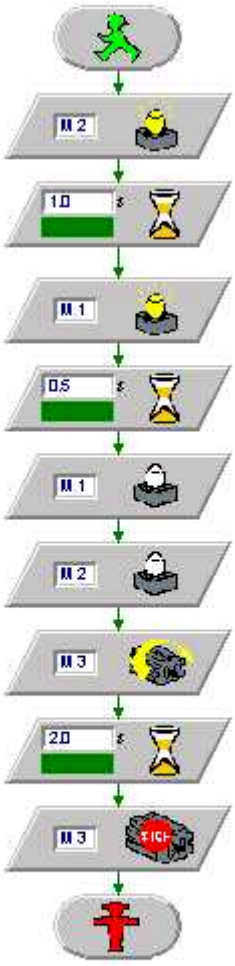
Die Analog-Zeile zeigt die (dezimalen) Werte an die an den Eingängen AX und AY gemessen werden. Die anderen Eingänge fehlen z.Zt. noch.

Die StartAmpel

So geht's los :

- Interface anschließen, Funktion mit dem Interface Panel testen
- An das Interface anschließen M1 : gelbe Lampe, M2 rote Lampe, M3 MiniMotor
- im Verzeichnis {app}\FiFa40 z.B. C:\ftComputing\FiFa40 auf StartAmpel.VBP klicken.
- die Source von frmMain (StartAmpel.FRM) kurz ansehen (es sind nur ein paar Zeilen)

- F5 Drücken : das Programm läuft an
- Kontrollieren, ob COM1 der richtige Anschluß ist
  (es muß der Wert vom Interface Panel sein)
- START Drücken : Es geht los – und das wars denn auch schon
- Und nochmal jetzt aber Shift+F8-Drücken – START – Drücken
- Und dann weiter mit Shift+F8 durch das Programm im Einzelschritt. Bei ft.Pause dauerts ein wenig
- man kann auch F8 allein drücken, dann gerät man aber in FishFa40AX.CLS, das ist für den Anfang zuviel des Guten.



StartAmpel

```
Private Sub cmdAction_Click()
  Dim ft As New FishFace
  ft.OpenInterfaceUSB _
       ftiROBO_first_USB, 0
  ft.SetMotor ftiM2, ftiEin
  ft.Pause 1000
  ft.SetMotor ftiM1, ftiEin
  ft.Pause 500
  ft.SetMotor ftiM1, ftiAus
  ft.SetMotor ftiM2, ftiAus
  ft.SetMotor ftiM3, ftiLinks
  ft.Pause 2000
  ft.SetMotor ftiM3, ftiAus
  ft.CloseInterface
  Unload Me
End Sub
```

Das Projekt StartAmpel.VBP besteht aus der gleichnamigen Projekt-Datei und der zugehörenden Datei StartAmpel.VBW mit weiteren Projektdaten.

Das Programm ist in AmpelStart.FRM enthalten. Auch die Daten, die visuell ZusammengeKlickt wurden, man kann sie mit NotePad ansehen, sind aber jetzt nicht weiter von Interesse.

Sub cmdAction_Click ist die einzige Nutzroutine des Programmes, die die M-Ausgänge schaltet.

Und der Klassenmodul FishFa40AX.CLS mit den Interface-Funktionen.

Zu den Elementen :

- Dim ft As New FishFace : anlegen einer neuen Instanz der Klasse FishFace (Teil von FishFa40AX.CLS) mit dem Name ft. Unter diesem Namen werden dann die Methoden (Funktionen) der Klasse angesprochen.
- ft.OpenInterfaceUSB ftiROBO_first_USB, 0 : Herstellenfür einer Verbindung für das erste ROBO Interface an USB.

- ft.SetMotor ftiM2, ftiEin : Einschalten der roten Lampe
  Die Methoden von FishFace bieten meist einen Auswahlliste (Enum) möglicher Parameterwerte. Hier aus der Aufzählung ftiOut und ftiDir. Es können aber auch einfache Zahlen oder eigene Konstanten angegeben werden.
- ft.Pause : Das Programm wird für 1000 MilliSekunden (1 Sekunde) angehalten.
- ft.SetMotor ftiM1, ftiEin die gelbe Lampe wird für 500 MilliSekunden zugeschaltet. und dann werden beide aus und der Motor an M3 wird für 2000 MilliSekunden angeschaltet
- und der Ordnung halber : ft.CloseInterface, die Verbindung zum Interface gekappt und die (und damit das Programm) mit Unload Me entladen. Ganz ordentliche könnten hier noch ein Set ft = Nothing dazwischen schieben um die FishFace Instanz zu entfernen, das geschieht hier aber auch automatisch.

Das wars denn auch schon für den Anfang. Etwas ausführlicher geht’s dann mit dem Beispielprojekt weiter. Vorher sollte man aber noch etwas mit dem Programm "herumspielen".

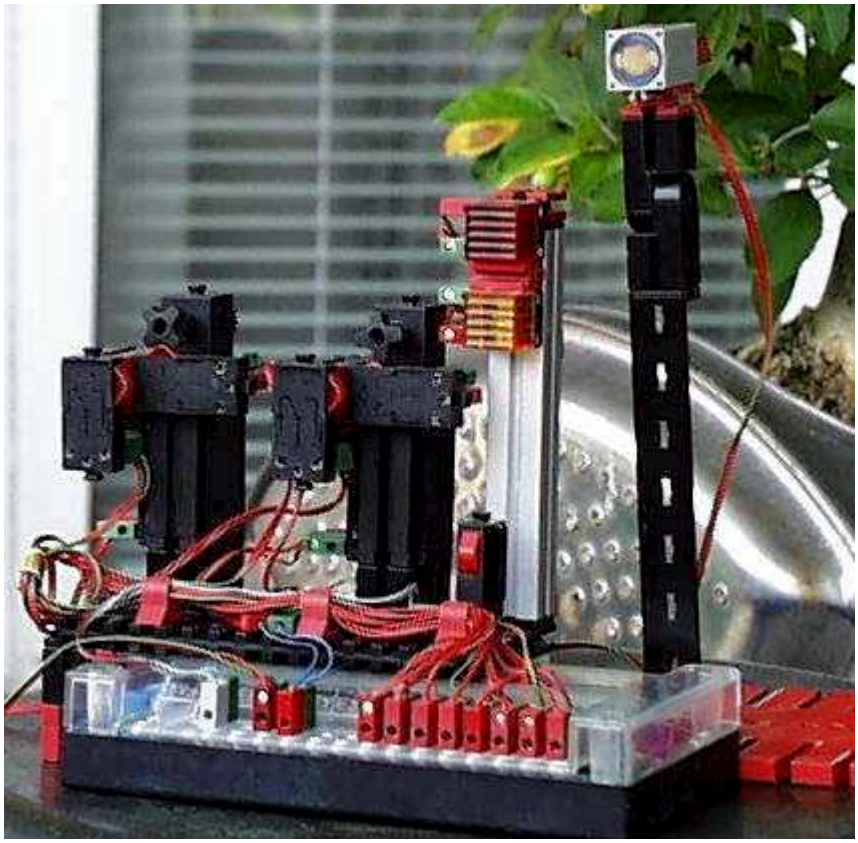
FiFa40AX.VBP : Das Beispielprojekt

Allgemeines

Der Einsatz von FishFace-Methoden und Eigenschaften wird anhand eines Beispielprogrammes vorgestellt. Die Methoden werden in mehreren Subs zu Gruppen zusammengefaßt und einer LLWin3.0-Lösung gegenübergestellt. Das erleichtert den Umstieg von LLWin und bietet gleichzeitig noch ein Ablauf-Diagramm. Die LLWin-Lösungen wurden bewußt knapp gehalten, insbesondere wurde auf den Einsatz des Terminalbausteins verzichtet, der in manchen Punkten der Bedienoberfläche des VB-Programms entspricht, aber letztlich vom Thema ablenkt. Das sollte LLWin-Kenner nicht davon abhalten, die gezeigten LLWin-Fragmente entsprechend zu erweitern (z.B. durch Abbildung der lblStatus-Ausgaben auf entsprechende des Terminal-Bausteins).

Das Beipielprojekt benötigt ein einfaches Testmodell :

Das Testmodell



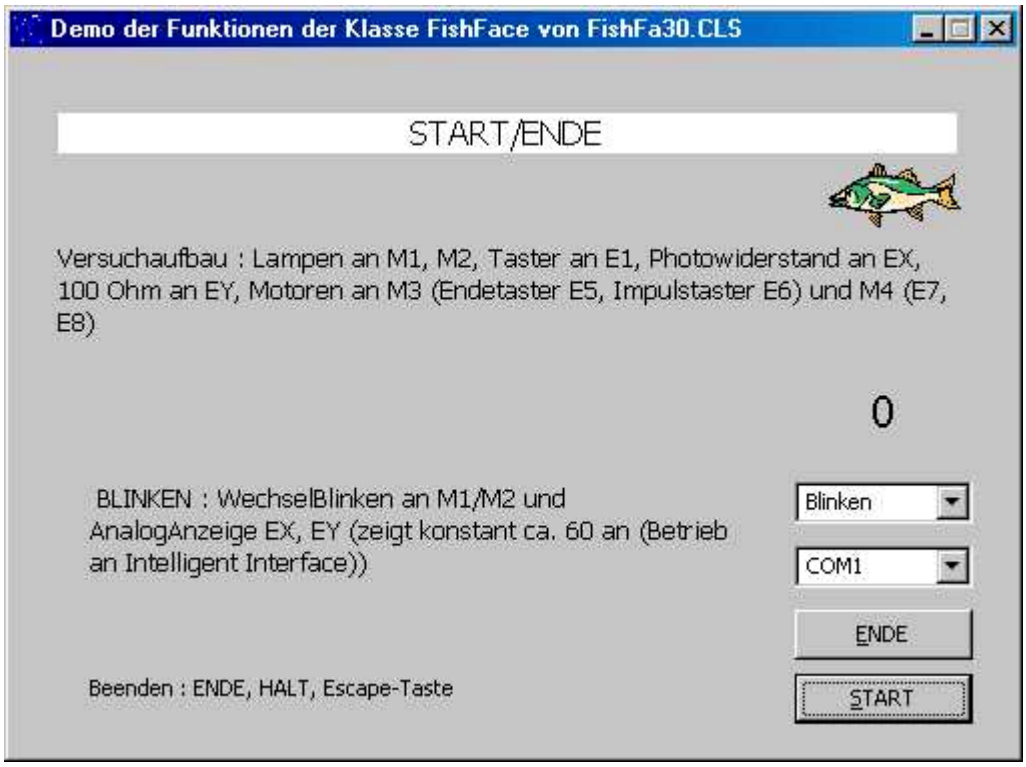
Das Testmodell hat nur die Aufgabe, die FishFace-Methode zu demonstrieren, es führt keine bestimmten Funktionen aus. Es besteht aus folgenden Elementen (von links) :

- Motor mit Getriebe und Impulsrad an M3 mit Endtaster an I5 und Impulstaster an I6
- Motor mit Getriebe und Impulsrad an M4 mit Endtaster an I7 und Impulstaster an I8
- Taster an I1
- Rote Lampe an M2 und Gelbe an M1
- Photowiderstand an AX (schwenkbar auf bewegbarer Säule)

- Festwiderstand 100 Ohm an AY (zu Vergleichszwecken, sollte beim Intelligent Interface Werte von ca. 60 anzeigen). Ist entbehrlich.

Alle Taster werden als Schließer geschaltet (Kontakte 1 und 3, Betätigung : Schließen)

FiFa40AX.VBP : Das UserInterface

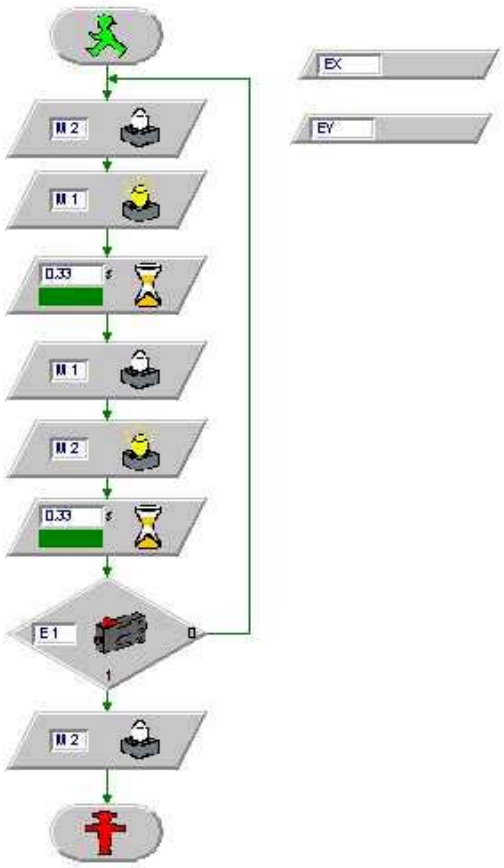


Das ist die einzige Form des Projektes FiFa40AX.VBP. Sie bietet die Kommunikation mit dem Benutzer und hat folgende Elemente :

- lblStatus : die weiße Zeile oben, in ihr werden der aktuelle Status des Programms und Anweisungen an den Benutzer angezeigt
- lblAnalog : Anzeige der aktuellen Werte von AX – AY und auch von Positionswerten.
- cboProgName : ComboBox mit den verfügbaren Demo Sub's. Eine Kurzbeschreibung wird jeweils links daneben angezeigt.
- cboPortName : ComboBox mit den zulässigen PortNamen, gleichzeitig Anzeige des aktuell eingestellten. Bei PortName USB wird das erste ROBO Interface an USB ausgewählt, bei COMx ein Intelligent Interface an diesem Port.
- ENDE/HALT Button (cmdEnde) zum Beenden des Programms oder einer gerade laufenden DemoRoutine. Die Beschriftung wechselt entsprechend. DemoRoutinen können auch durch die Escape-Taste angehalten werden.
- START Button (cmdAction) zum Start der aktuell angezeigten DemoRoutine
  Bei laufender DemoRoutine ist der START Button verriegelt (Enabled = False) um einen erneuten Start der DemoRoutine zu verhindern. (man könnte auch noch cboProgName / cboPortName sperren).

Die FishFace Beispiel Routinen werden hier mit entsprechenden LLWin 30 Routinen verglichen, das wurde von FishFa30 übernommen. Die entsprechenden ROBO Pro Level 1 Routinen sehen aber ganz ähnlich aus.

Sub Blinken : Wechselblinken mit zwei Lampen



Blinken : M1/M2 bis E1 gedrückt

```
Private Sub Blinken()
  Do
    ft.SetMotor ftiM2, ftiAus
    ft.SetMotor ftiM1, ftiEin
    lblAnalog = ft.GetAnalog(ftiAX) &
             " - " & ft.GetAnalog(ftiAY)
    ft.Pause 333
    ft.SetMotor ftiM1, ftiAus
    ft.SetMotor ftiM2, ftiEin
    lblAnalog = ft.GetAnalog(ftiAX) & _
             " - " & ft.GetAnalog(ftiAY)
    ft.Pause 333
  Loop Until ft.Finish(ftiI1)
End Sub
```

Geblinkt wird hier mit zwei Lampen (M2 Rot und M1 Gelb) im Wechsel in jeweils 333 MilliSekunden Abstand und das in einer Endlosschleife.

Lampe M2 aus : `ft.SetMotor ftiM2, ftiAus` .... auch hier als Parameter die Enums ftiOut.ftiM2 ...

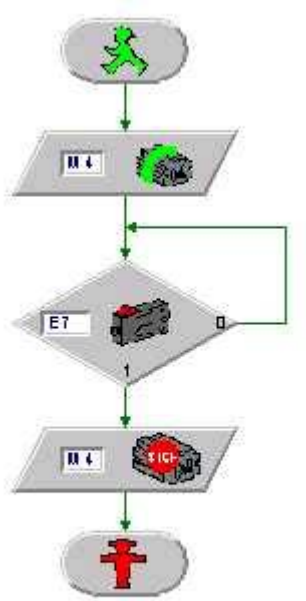
ft.Pause 333 : Programm für 0,3 Sekunden anhalten.

Neu ist hier die Schleife `Do ... Loop Until ft.Finish(ftiI1)` : die Befehle dazwischen werden solange (Until) durchlaufen bis die Methode `ft.Finish` True zurückgibt. Das tut sie hier wenn auf die Esc-Taste gedrückt wurde oder der Taster an I1 geschlossen wurde (bei dem TestModell manuell). Außerdem wird die Eigenschaft NotHalt ausgewertet, True bedeutet auch Abbruch. Im Beispiel wird `ft.NotHalt` bei HALT gesetzt.

Bei der LLWin Codierung wurde anstelle von Finish der Eingang-Baustein verwendet, es wird hier nur I1 abgefragt, mehr ist da auch nicht erforderlich, da die Laufzeitumgebung genügend Abbruchmöglichkeiten bietet. Die reine Abfrage eines I-Einganges geschieht bei FishFace mit `ft.GetInput`.

Außerdem werden mit `ft.GetAnalog(ftiAX)` und `ft.GetAnalog(ftiAY)` die aktuellen Analogwerte abgefragt und in `lblAnalog` angezeigt. LLWin macht die Abfrage in der Laufzeitumgebung und zeigt sie mit dem WerteAnzeigen-Baustein an (rechts oben).

Sub Fahren : Fahren eines Motors zu einem Endtaster



Fahren : Motor läuft bis E1 gedrückt wird

```
Private Sub Fahren()
lblStatus = "Beenden : I7 drücken"
  ft.SetMotor ftiM4, ftiRight
  ft.WaitForInput ftiI7
End Sub
```

Eine ganz einfache Übung : Fahren bis das Ende in Form eines geschlossenen I7 Tasters kommt.

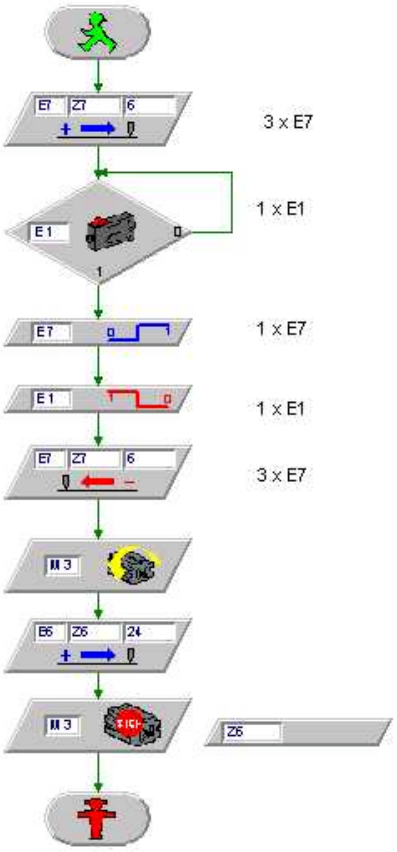
Hier gibt es im Gegensatz zu Blinken keine Schleife, da es hier nichts zu Wiederholen gibt.

Zur Erkennung von I7 = True wird `ft.WaitForInput` eingesetzt, der die gleichen "Nebenwirkungen" wie `ft.Finish` hat.

Bei LLWin wieder der Eingabe-Baustein. und danach folgt dann noch ein Motor Aus, bei VB6 aber nicht, Grund im rufenden `cmdAction_Click` gibt es nach dem Aufruf für Fahren noch ein `ft.ClearMotors` (Gilt natürlich auch für Blinken).

in lblStatus wird eine Bediener-Anweisung "Beenden : I7 drücken" angezeigt

Sub Warten



Warten : Demo der Warte-Befehle

```
Private Sub Warten()

Dim Zahler&

  lblStatus = "WaitForChange : 3 x I7" & _
                    "drücken & loslassen"
  ft.WaitForChange ftiI7, 6

  lblStatus = "WaitForInput : I1 drücken"
  ft.WaitForInput ftiI1

  lblStatus = "WaitForHigh : I7 drücken"
  ft.WaitForHigh ftiI7

  lblStatus = "WaitForLow : I1 drücken"
  ft.WaitForLow ftiI1

  lblStatus = "WaitForPositionDown : " & _
                              "3x I7 drücken"
  Zahler = 6
  ft.WaitForPositionDown ftiI7, Zahler, 0
  lblAnalog = Zahler

  lblStatus = "WaitForPositionUp : mit" & _
                     " Motorkraft 24 Impulse"
  ft.SetMotor ftiM3, ftiLeft, ftiHalf
  ft.WaitForPositionUp ftiI6, Zahler, 24
  ft.SetMotor ftiM3, ftiOff
  lblAnalog = Zahler

End Sub
```

Die Sub Warten zeigt restlichen Möglichkeiten auf etwas zu warten (zwei sind ja schon bekannt : WaitForInput, Pause/WaitForTime). Gewartet wird immer auf eine oder mehrere Veränderungen an einem der I-Eingänge. Hinweis : Mit WaitForInputIR ist ein Warten auf eine Taste des IR-Senders möglich.

WaitForChange wartet hier auf 6 Impulse (Pegelwechsel) an I7, dazu muß man im Beispielprogramm 3 mal den Taster an I7 drücken (und wieder loslassen).
Nutzen kann man diese Methode z.B. zum Verfahren eines Modells um eine festgelegte Anzahl von Impulsen, die Impulse werden dann über ein Impulsrad durch den Motor ausgelöst. Siehe auch WaitForPosition.

WaitForInput wartet auf I1 True
z.B. beim Anfahren einer Endposition

WaitForHigh wartet auf einen False/True Durchgang an I7. Das heißt : I7 muß vor dem Drücken des Taster offen gewesen sein.
Beispiel : Ausfahren aus einer Lichtschranke.

WaitForLow dito : aber True/False Durchgang
Beispiel Einfahren in eine Lichtschranke.

WaitForPositionDown : ähnlich WaitForChange, es wird aber mit den tatsächlichen Positionen gearbeitet, Zahler enthält die aktuelle Position und der Parameter Position (hier 0) ist die ZielPosition, Zahler enthält nach Ende der Methode die tatsächlich erreichte Position, dabei werden die festgestellten Impulse von Zahler abgezogen.

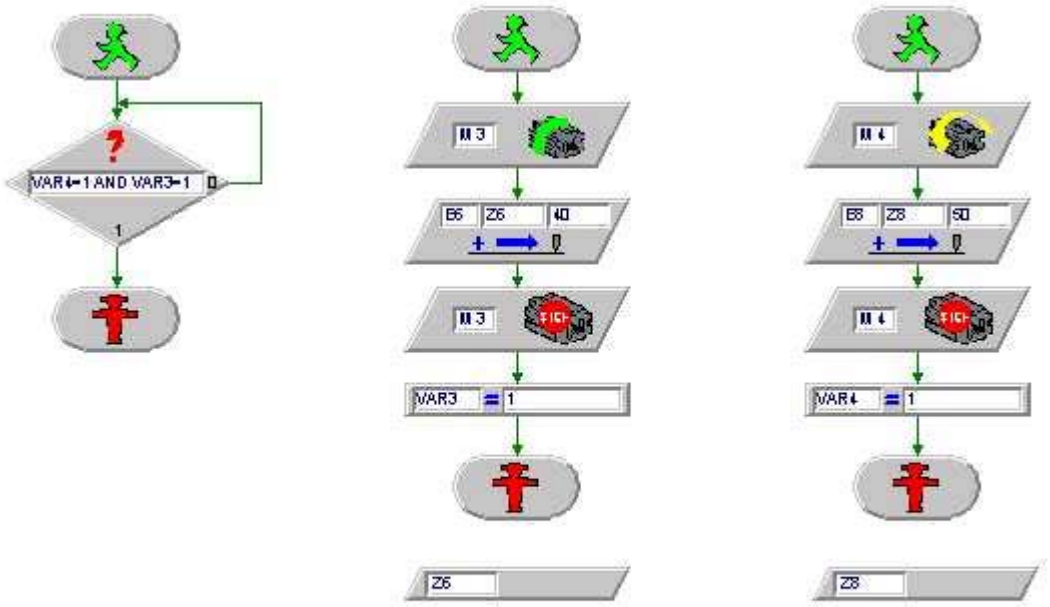
WaitForPositionUp zur Abwechslung mit Motorkraft und aufwärts ZielPosition ist hier 24, AusgangsPosition 0, da Zahler seit WaitForPosiionDown nicht verändert wurde.
In `lblAnalog` wird in beiden Fällen die tatsächlich erreichte Position angezeigt.

Optionale Parameter : Werte, die der Methode wahlweise übergeben werden können, die Methode ändert dadurch ihr Verhalten. ft.SetMotor ftiM3, ftiLeft, ftiHalf. ftiHalf steht als Wert für den Parameter Speed, also mit halber Geschwindigkeit, default ist ftiFull.

Die WaitFor... Methoden können alle vorzeitig durch die Esc-Taste oder die Eigenschaft ft.NotHalt beendet werden. WaitForChange, WaitForPositionUp/Down kennen außerdem noch die TermInputNr(optionaler Parameter), die Nummer eines I-Einganges, der bei True ebenfalls ein vorzeitiges Ende bewirkt, üblicherweise ein EndTaster.

lblStatus enthält immer die entsprechende Bedienanweisung

Sub Positionieren : Fahren zu einer vorgegebenen Position



Simultanes Anfahren von Position 40 (M3) und 50 (M4)

```
Private Sub Positionieren()

  lblStatus = "Motor an M4 fährt 50 links & Motor an M3, 40 rechts"
  ft.SetMotor ftiM4, ftiLeft, ftiHalf, 50
  ft.SetMotor ftiM3, ftiRechts, ftiFull, 40

  Do
    lblAnalog = ft.GetCounter(ftiI6) & " - " & ft.GetCounter(ftiI8)
  Loop While ft.WaitForMotors(100, ftiM4, ftiM3) = ftiWait.ftiTime

lblAnalog = ft.GetCounter(ftiI6) & " - " & ft.GetCounter(ftiI8)

End Sub
```

Hier fahren zwei Motoren gleichzeitig (simultan) unterschiedliche Ziele an. Der Motor an M4 fährt mit halber Geschwindigkeit 50 Impulse nach links und der Motor an M3 mit voller Geschwindigkeit 40 Impulse nach rechts. Dazu hat die bekannte Methode `ft.SetMotor` einen weiteren (optionalen) Parameter Counter der die angegebenen Werte enthält. Die Methode ist asynchron, d.h. der Motor läuft unabhängig vom übrigen Programm weiter. Das ist ansich nichts neues, denn die anderen Varianten von ft.SetMotor verhalten sich ebenso,

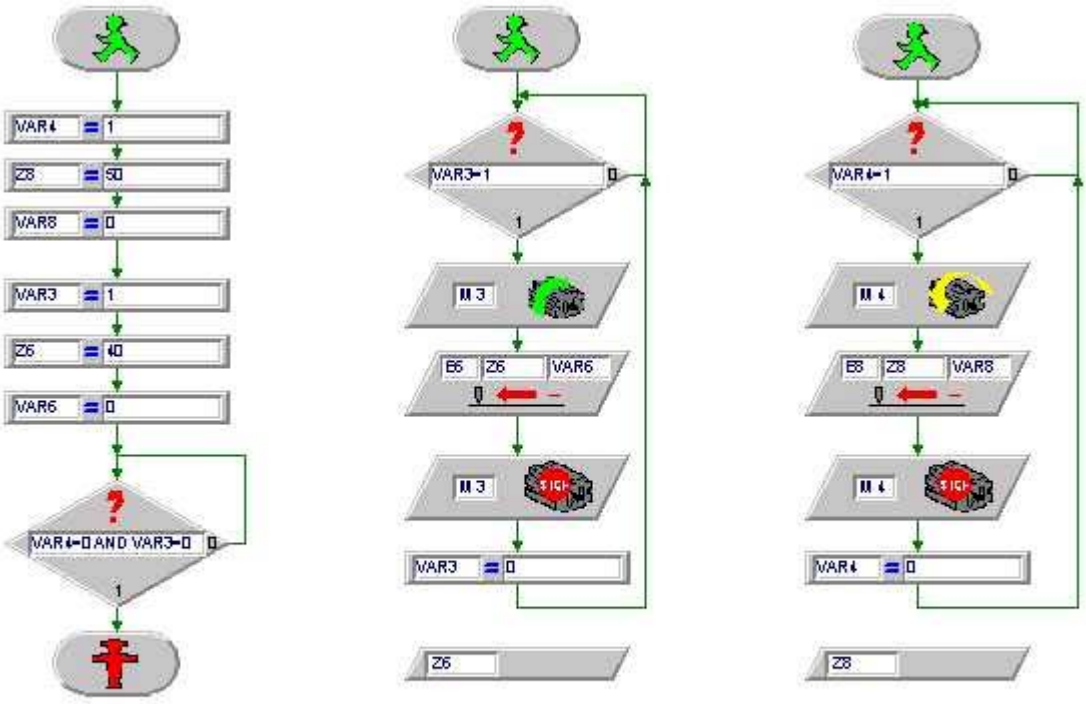
neu ist hier, das die Motoren intern bei Erreichen der vorgegebenen Impulszahl abgeschaltet werden. Das Programm, das sie über die ft.SetMotor Methoden gestartet hat, muß deswegen (bei Gelegenheit) nachfragen, ob die Zielwerte schon erreicht sind, in der Zwischenzeit kann etwas anderes erledigt werden.

Die Methode ft.WaitForMotors kann das tun. Der erste Parameter hier 100, gibt die Zeit in Millisekunden an, die jeweils gewartet werden soll, spätestens dann kehrt sie zurück und das rufende Programm fragt ab, warum. Die zweiten Parameter sind eine Liste von M-Ausgängen (Motoren) auf die gewartet werden soll.

Hier wurde eine Do Loop While – Schleife verwendet in der alle 100 MilliSekunden (der Warte Parameter) der aktuelle Counterstand angezeigt wird. Die Schleife wird solange durchlaufen, wie der Rückkehrwert = ftiTime ist d.h. die Motoren sind noch nicht am Ziel. weitere Rückkehrwerte sind ftiEnde : Ziel erreicht, ftiEsc und ftiNotHalt : vorzeitiges Ende.

Damit die ganze Zählerei funktioniert ist ein fester ModellAufbau erforderlich. Zu jedem Motor gehört ein Endtaster und ein Impulstaster (M1 / I1 / I2, M2 / I3 / I4, M3 / I5 / I6 ....). Der Endtaster dient zum einen als fester Bezugspunkt für die Modell Home-Position und ist eine "Notbremse" bei dessen Erreichen der entsprechende Motor ggf. vorzeitig abgeschaltet wird. Aus Gründe der Verfügbarkeit gibt es nur für eine Drehrichtung einen Endtaster, der Endtaster wird bei Linkslauf ausgewertet.

LLWin kennt keinen entsprechenden simultanen Befehl, hat dafür aber die Möglichkeit (simultaner) Task, sprich mehrere "Grüner Männchen" in einem Projekt. Oben eine knappe Lösung mit drei Grünen Männchen. Unten eine Lösung, die auch Wiederholungen zuläßt.



Simultanes Anfahren von Position 40 (M3) und 50 (M4)

FiFa40AX.VBP : Der Programmrahmen

Hinweise

Solange Programme innerhalb ihrer Entwicklungsumgebung (IDE – Interactive Development Environment) gestestet werden, können auch deren Möglichkeiten zu Start und Abbruch genutzt werden (Bei Visual Basic : Das Ausführen Menü bzw. die entsprechenden Icons) . Bei der Ausführung eines übersetzten Programmes über ein Desktop oder Explorer-Icon steht sie nicht mehr zur Verfügung, in diesem Fall muß das Programm selber entsprechende Möglichkeiten bieten.

Methoden, die den Programmablauf anhalten oder steuern (Finish, WaitForxxx, Pause) sind abbrechbar. Dazu werten sie die Esc-Taste und die Eigenschaft NotHalt aus. die Methoden beenden sich dann ohne besondere Hinweise. Die Esc-Taste kann jederzeit durch den Bediener betätigt werden, die Eigenschaft NotHalt muß im Programm z.B. beim Betätigen des HALT-Buttons gesetzt werden

Ebenso muß ein wiederholter Start bzw. ein Schließen der Form während das Programm noch (in einer Schleife) läuft (und auch nach Schließen der Form weiterläuft) verhindert werden. Das geschieht hier durch Verriegeln der entsprechenden Buttons.

cmdAction : Steuern des Modellbetriebes

```
Private Sub cmdAction_Click()

On Error GoTo ftiFehler

If cboPortName.ListIndex = 0 Then _
    ft.OpenInterfaceUSB ftiROBO_first_IF, 0 _
Else ft.OpenInterfaceCOM ftiIntelligent_IF, _
                         cboPortName.ListIndex, 12

cmdAction.Enabled = False
cmdEnde.Caption = "&HALT"
lblStatus.Caption = "Läuft"

Select Case cboProgName.ItemData(cboProgName.ListIndex)
Case 1
  Blinken
Case 2
  Fahren
Case 3
  Warten
Case 4
  Positionieren
End Select

ft.ClearMotors
ft.CloseInterface

cmdAction.Enabled = True
cmdEnde.Caption = "&ENDE"
lblStatus.Caption = "START/ENDE"
cmdEnde.SetFocus
Exit Sub

ftiFehler:
  lblStatus = Err.Number & " : " & Err.Source & "." & _
                                       Err.Description
```

```
  cmdEnde.Caption = "&ENDE"
  ft.CloseInterface
End Sub
```

`Dim ft As New FishFace` (im globalen Bereich zu Programmanfang) erzeugt eine **FishFace Instanz**. Sie wurde in den globalen Bereich gestellt, da ft von mehreren Subs (die ja alle ihren eigenen Gültigkeitsbereich für dort deklarierte Variablen haben) zugegriffen wird.

`ft.OpenInterface...` stellt **Verbindung** zu einem Interface her. Wahlweise einem ROBO Interface an USB oder einem Intelligent Interface an COM. Die Analogeingänge des Intelligent Interface werden abgefragt (AnalogZyklen = 12), es ist kein Extension Module vorhanden (default). Die Verbindung wird durch ft.CloseInterface wieder beendet.

Während der Laufzeit des Programmes können **Fehler** auftreten, sie sollten abgefangen werden. Das geschieht hier durch die Klammer, die durch `On Error GoTo ftiFehler` und `ftiFehler`: gebildet wird. Alle Statements, die logisch dazwischen stehen – also auch die in Subs -, verzweigen bei Auftreten eines Fehlers zum Label ftiFehler. Die Fehlerbehandlung besteht hier in der Anzeige des Fehler mit Nummer und Text, einem Schließen der Interface-Verbindung und einem Verlassen der Sub cmdAction_Click.

Hier sind die Fehler, die von FishFace erkannt und signalisiert (**Err.Raise**) werden, besonders interessant. Das sind 30001 : Interface Problem und 30002 : Kein Open. 30001 ist ein allgemeiner Fehler, der z.B. auftritt, wenn im laufenden Betrieb der Strom ausfällt (Vergessen vor dem ft.OpenInterfacexxx Strom einzuschalten und das Kabel zu stecken. Oder auch : Notaus durch Stecker ziehen). 30002 ist meist ein Programmierfehler, es wurden ft.Methoden vor einem ft.OpenInterfacexxx (oder nach einem ft.CloseInterface) aufgerufen.

`Select Case cboProgName.ItemData(cboProgName.ListIndex)` ist lediglich ein **Sprungverteiler** über den die durch die KomboBox cboProgName ausgewählte Sub aufgerufen wird. cboProgName.ListIndex liefert die aktuellen Listen Position (mit 0 beginnend), in cboProgName.ItemData steht dann die Case Ziffer

`cmdAction.Enabled = False` **sperrt** den erneuten Aufruf von START (cmdAction_Click). Nach dem Ablauf einer Demo-Sub wird es dann wieder entsperrt : `cmdAction.Enabled = True`. Parallel dazu wechselt die Beschriftung von cmdEnde von ENDE in HALT und wieder in ENDE.

Nach dem Ablauf einer Demo-Sub werden außerdem alle M-Ausgänge (Motoren) **ausgeschaltet** und die Interface-Verbindung geschlossen, deswegen können sie am Ende der Demo-Subs fehlen.

cmdEnde : Beenden des Modellbetriebes

```
Private Sub cmdEnde_Click()
  If cmdEnde.Caption = "&HALT" Then ft.NotHalt = True _
   Else Unload Me
End Sub

Private Sub Form_QueryUnload(Cancel As Integer, _
        UnloadMode As Integer)
  If cmdEnde.Caption = "&HALT" Then Cancel = 1
End Sub
```

Während des **Ablaufs** einer Demo-Sub ist der cmdEnde-Button mit HALT beschriftet, das wird hier abgefragt : `If cmdEnde.Caption = "&HALT" Then`. Das Programm kann an dieser Stelle nicht einfach beendet werden, da das Modell noch läuft, in diesem Fall wird die Eigenschaft `ft.NotHalt` gesetzt, um einen Ende-Wunsch anzumelden (er wird z.B. von ft.Finish() erkannt), Nach Ende einer Demo-Sub wechselt die Beschriftung wieder auf ENDE und die Form kann entladen und damit das Programm beendet werden.

Da über das **x** rechts oben im Rahmen der Form auch ein Programmende herbeigeführt werden kann,wird das konsequenterweise in Form_QueryUnload bei cmdEnde = "&HALT" verhindert.

Form_Load : Startwerte

```
Private Sub Form_Load()
  cboPortName.ListIndex = 0
  cboProgName.ListIndex = 0
End Sub
```

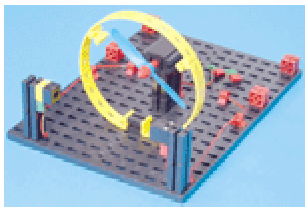
Gleich beim Start des Programm, dem Laden der Form werden die in den ComboBoxen anzuzeigenden Texte bestimmt : `cboPortName.ListIndex = 0` (USB) und `cboProgName.ListIndex = 0`. Besonders cboPortName sollte hier auf den eigenen Anschluß angepaßt werden.

Modelle & FishFace-Erweiterungen

Kleine Modelle

Eine Folge kleiner Modelle, die primär Visual Basic-Programmiertechniken und den Umgang mit FishFace zeigen sollen.

Händetrockner : Templates, ActiveX.DLLs


Modell aus dem Computing Starter Kit 16 553 (Handbuch einzeln : 30 434). Aufgabe laut Handbuch : "Der Händetrockner soll so programmiert werden, daß, sobald die Lichtschranke unterbrochen wird, der Lüfter ein- und nach 5 Sekunden wieder ausgeschaltet wird."

Templates

Bisher war das Programmgerippe eine "Einzelanfertigung" zu der die FishFa40AX.CLS und die eigentliche Anwendung hinzugefügt wurde. Jetzt soll ein Programmgerippe "von der Stange" genutzt werden, ein sogenanntes Template. Das ist ein vorgefertigtes Programmgerippe in das nur noch die eigentliche Anwendung einzufügen ist. In diesem Fall geht es um ftComputing40.VBP. Das ist ein lauffähiges Projekt mit START und ENDE Button, FishFa40AX und den erforderlichen ft.OpenInterfacexxx / CloseInterface Methoden-Aufrufen.

Zu finden ist es im Verzeichnis {app}\Template40\VB6. Der Inhalt sollte nach C:\Programme\Microsoft Visual Studio\VB98\Template\Projects (bei Standard Installation) verschoben werden. Anschließend kann es von dort als normales Projekt geöffnet und begutachtet werden. Man wird im Laufe der Zeit einige Anpassungen machen wollen. z.B. kann auf die KomboBox mit den PortNamen verzichtet werden, wenn nur ein Interface vorhanden ist, es wird beim ft.OpenInterfacexxx dann fest der entsprechende PortName eingetragen. Bei den Projekteigenschaften findet sich ebenfalls einiges, daß noch persönlicher gestaltet werden kann : Testen – Speichern – SoWeit – SoGut.

Nutzen kann man Template dann beim Anlegen eines neuen Projektes : (Menü : Datei | Neues Projekt) in der Auswahlmaske erscheint jetzt auch ein Projekt ftCompting40, das ist dann zu wählen. Am besten speichert man es dann gleich unter dem Namen des neuen Projektes in ein eigenes Verzeichnis.

ActiveX.DLLs

Bei den bisherigen Projekten wurde der Klassenmodul FishFa40AX.CLS genutzt, man kann es sich auch leichter machen und auf die kompilierte Version von FishFa40AX.CLS : FishFa40AX.DLL zugreifen. FishFa40AX.DLL muß dazu registriert sein, das ist bei Installation von vbFish40Setup.EXE der Fall. (Manuell kann man das im Menü : Projekt | Verweise einfach über Durchsuchen machen). Man muß dann nur noch einen Verweis in die

Projektdatei eintragen : Menü : Projekt | Verweise : "FishFace : FishFa40AX.DLL" ankreuzen. Wenn man dann noch das Template ftComputing40 nutzt, ist das schon geschehen.

Da die Source von FishFa40AX.DLL vorliegt, kann man es natürlich auch selber ändern. Neues Projekt vom Typ ActiveX.DLL anlegen, den KlassenModul Class1 wieder entfernen und den Klassenmodul FishFa40AX.CLS hinzufügen, die Eigenschaft Instancing auf 5 – Multiuse setzen und in Projekt | Eigenschaften Projektname und Projektbeschreibung mit sinnvollen Namen benennen. Projektname wird der Name der neuen Datei, Projektbeschreibung erscheint nach der Kompilierung dann in der Verweisliste. Die Registrierung erfolgt automatisch. Man sollte noch bei den Projekteigenschaften auf der Tab Komponente nach der Versionskompatibilität sehen, anfangs sollte dort Projektkompatibiltät stehen, später (wenn man die Methoden und Eigenschaften nicht mehr ändern will) dann Binärkompatiblilität. Mit einem Verweis auf die aktuelle DLL oder eine alte zu der sie kompatibel sein soll. Dieser Absatz hat aber Zeit für später. Jetzt aber :

Die Modellfunktionen

Wenn das Template ftComputing genutzt wird (und das ist bei dem mitgelieferten Programm in {app}\HandTrockner der Fall) gibt es da eine **Private Sub Action** :

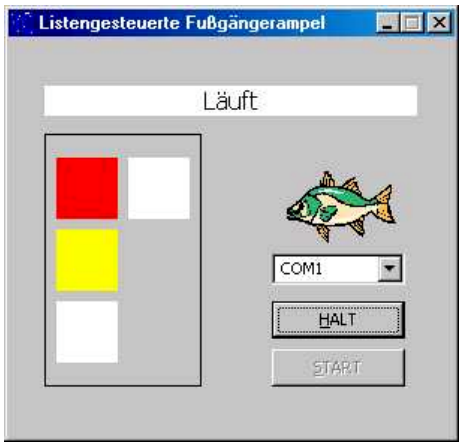
```
Private Sub Action()
' --- Routine zum Betrieb des Händetrockners ---
Const mVentilator = 1, mLampe = 2
Const ePhototransistor = 1

  ft.SetMotor mLampe, ftiEin
  ft.WaitForTime 1000
  Do
    If Not ft.GetInput(ePhototransistor) Then
      ft.SetMotor mVentilator, ftiEin
      lblStatus = "--- trocknet ---"
      tmrAction.Enabled = True
      ft.WaitForTime 5000
      ft.SetMotor mVentilator, ftiAus
      lblStatus = "--- bereit ---"
      tmrAction.Enabled = False
      imgFish.Height = 735
    End If
  Loop Until ft.Finish()

End Sub
```

Viel passiert da nicht mehr : Die Do – Loop Until Schleife ist bekannt. Vorher wird noch die Lampe für die Lichtschranke angewärmt. In der Schleife gibt es nur ein If Then – End If in der abgefragt wird, ob die Lichtschranke offen ist (Not ft.GetInput(ePhototransistor)), dann wird der Ventilator angeworfen, die Nachricht "trocknet" ausgegeben und der Fisch angestellt und 5 Sekunden gewartet. Anschließend das Ganze rückwärts. Wenn man den Fisch in Schwung halten will braucht man auch noch einen Timer :

```
Private Sub tmrAction_Timer()
  If imgFish.Height = 735 Then imgFish.Height = 555 _
   Else imgFish.Height = 735
End Sub
```

Die Höhe des Image-Controls wird hier wechselnd verändert. Beim Testen sollte man den Fisch aber lieber abstellen tmrAction.Enabled = False, er stört da nur. Aber bei einer Demo muß man schon sichtbar machen, dass das Programm läuft, bei dem Krach, den der Ventilator macht --- .

FussAmpel : Über den Umgang mit SetMotors (SetLamps) und Listen



Mal wieder eine Ampel, diesmal als Fußgängerampel mit vier Lampen an den O-Ausgängen und einem Taster an I1.

Bei gleichzeitiger Steuerung mehrerer Motoren oder Lampen über SetMotor/SetLamp ist eine Folge vom Befehlen zum Ein- und Aussschalten erforderlich. Mit der Methode SetMotor**s*** können alle M-Ausgänge und damit auch alle O-Ausgänge auf einmal geschaltet werden. Und wenn man dann noch die einzelnen AmpelTakte in eine Tabelle stellt, kommt miteinmal dabei ein richtig interessantes Programm heraus :

SetMotors (SetLamps)

Die Methode SetMotors hat in seiner einfachsten Form nur den Parameter MotorStatus in dem alle M-Ausgänge (M1 –M4) bzw. die "halben" M-Ausgänge (O1 – O8) geschaltet werden können. Bei angeschlossenen Extensions entsprechend mehr. Dazu wird der Parameter Direction von SetMotor für jeden M-Ausgang als 2bitWert (ftiLinks = 01, ftiRechts = 10, ftiEin = 01, ftiAus = 00) in MotorStatus abgestellt. Beim Schalten von O-Ausgängen gelten dann 1bit-Werte für Ein und Aus.

MotorStatus ist eine Dezimalzahl mit Vorzeichen und hat eine Länge von 32bit. Im Rechner wird sie aber nur als bitFolge gesehen. Im Anwendungsprogramm kann man sie wahlweise als Dezimalzahl, als Hexa- oder Binärwert verwenden. Um die Direction-Wert Anordnung zu sehen, wird erstmal die Binärdarstellung genutzt

O8	O7	O6	O5	O4	O3	O2	O1

ganz rechts ist das bit 0. O1 ftiEin, Rest ftiAus heißt dann 0000000000000001 oder schlicht dezimal 1. O2 hat seine Position 1 bit weiter links ftiEin hieße da 0000000000000010 oder dezimal 2, O1 und O2 ftiEin : 0000000000000011 oder dezimal 2 + 1 = 3. O3 ist dann 0000000000000100 oder dezimal 8, das ist Hexa auch 8 (&H8&).

Für die Lampen der Ampel werden jetzt solche (Hexa) Konstanten angelegt :

```
Const lGruen = &H4&, lGelb = &H2&, lRot = &H1&, lFuss = &H8&
```

entsprechend der Belegung der O-Ausgänge am Interface.

Ein `ft.SetMotors lRot + lFuss` schaltet dann die rote Autoampel und die grüne Fußgängerampel ein, alle anderen Ampeln (O-Ausgänge) werden ausgeschaltet (auf den entsprechenden Positionen stehen ja Nullen).

* Einen besonderen Befehl SetLamp**s** gibt es nicht, dem SetMotors ist es egal wie seine bits gesetzt werden. Der Vollständigkeithalber : man kann mit dem zusätzlichen SpeedStatus Parameter auch noch die Helligkeit der Lampen steuern.

Die FussTabelle

Man könnte jetzt eine Folge von ft.SetMotors schreiben, die die Fußgängerphase bilden, etwa so:

```
ft.SetMotors lGruen
ft.SetMotors lGelb
ft.SetMotors lRot
ft.SetMotors lRot + lFuss
```

```
ft.SetMotors lRot
ft.SetMotors lRot + lGelb
```

und dazwischen noch jeweils eine ft.Pause. Das ist schon eine rechte Vereinfachung gegenüber dem Einsatz von ft.SetMotor. Eleganter wird es noch, wenn diese Werte in eine Tabelle gestellt werden, dann kann man sie in einer einfachen Schleife abarbeiten (und könnte die Tabelle auch flexibel aus einer Datei lesen).

```
Private Sub Form_Load()
ReDim FussTabelle(1 To 6)
  FussTabelle(1).AmpelStatus = lGruen
  FussTabelle(1).Dauer = 1000
  FussTabelle(2).AmpelStatus = lGelb
  FussTabelle(2).Dauer = 500
  FussTabelle(3).AmpelStatus = lRot
  FussTabelle(3).Dauer = 500
  FussTabelle(4).AmpelStatus = lRot + lFuss
  FussTabelle(4).Dauer = 2000
  FussTabelle(5).AmpelStatus = lRot
  FussTabelle(5).Dauer = 500
  FussTabelle(6).AmpelStatus = lRot + lGelb
  FussTabelle(6).Dauer = 500
End Sub
```

Die FussTabelle enthält als Elemente eine Struktur (Type) mit den Werten für einen Ampeltakt.

```
Private Type FussSatz
  AmpelStatus As Long
  Dauer As Long
End Type
```

Die Tabelle selber ist im globalen Bereich zu Beginn des Programm deklariert worden und wird hier nochmal mit Bekanntsein der tatsächlichen Werteanzehl redimensioniert.

FussPhase

Die Schleife für die Abarbeitung der Takte einer Fußgängerphase befindet sich in

```
Private Sub FussPhase()
Dim i%
  For i = 1 To UBound(FussTabelle)
    ft.SetMotors FussTabelle(i).AmpelStatus
    Anzeige FussTabelle(i).AmpelStatus
    ft.Pause FussTabelle(i).Dauer
  Next i
End Sub
```

Da gibt es dann den (einzigen) ft.SetMotors Aufruf und eine ft.Pause in einer Schleife, die bis UBound(FussTabelle) also bis zur aktuellen Länge der FussTabelle geht.

Damit es nicht zu einfach wird :

Anzeige

Eine Anzeige des aktuellen Ampeltaktes in entsprechenden Labels auf der Form :

```
Private Sub Anzeige(AmpelStatus&)
If (AmpelStatus And lGruen) > 0 Then lblGruen.BackColor = vbGreen _
  Else lblGruen.BackColor = vbWhite
If (AmpelStatus And lGelb) > 0 Then lblGelb.BackColor = vbYellow _
  Else lblGelb.BackColor = vbWhite
```

```
If (AmpelStatus And lRot) > 0 Then lblRot.BackColor = vbRed _
  Else lblRot.BackColor = vbWhite
If (AmpelStatus And lFuss) > 0 Then lblFuss.BackColor = vbGreen _
  Else lblFuss.BackColor = vbWhite
End Sub
```

Hier wird der AmpelStatus wieder auseinandergenommen und dann auf die einzelnen Label farblich verteilt.

AmpelStatus And lGruen ist ein bitweises Verarbeiten einer Variable (AmpelStatus) und einer Maske (lGruen), wobei im Ergebnis alle bits stehen bleiben, die sowohl in Variable wie auch Maske eine Endsprechung haben :

```
00001111   ' hier wären die O-Ausgänge O1-O4 an
And
00000100
=
00000100
```

Abgefragt wird das Ergebnis auf größer Null, weil nur interessiert, ob lGruen an ist. Es gibt übrigens auch noch Or und Xor ... einfach auf F1-Hilfe drücken um mehr zu erfahren.

Action

Für die Hauptroutine Action bleibt dann nicht mehr viel über :

```
Private Sub Action()
Do
    ft.SetMotors lGruen
    Anzeige lGruen
    If ft.GetInput(eFussWunsch) Then FussPhase
    ft.Pause 111
  Loop Until ft.Finish()
End Sub
```

Eine Endlosschleife in der auf Fußgänger und deren Wünsche gewartet wird. In der Schleife wird erstmal die Autoseite auf Grün geschaltet (und nur die : ftSetMotor**s**), das wird dann angezeigt, danach die Abfrage des FussWunsch-Taster mit dem Aufruf der FussPhase, genau alle 111 MilliSekunden.

Industry Robots

Einige Beispielprogramme, die – aufeinander aufbauend – den Einsatz von FishFace bei den Industry Robots zeigen sollen.

RobStep : Steuerung eines Robots durch Einzelschritte

Bändigen eines ganz normalen Industry Robots (Säulen- oder Knickarm-Robot). Aufbau nach Handbuch.

Das Programm bringt, neben dem bekannten Rahmen des Templates ftComputing40, eine Reihe von Buttons, über die die Robotfunktionen abgerufen werden können, am besten von unten nach oben :

- Anfahren der Home-Position (muß die erste Funktion sein).
- Fahren zur Position A (dem Magazin)
- Greifen eines Teils
- Fahren zur Position B (der Ablage)
- und Ablegen
- HALT/ENDE hat noch eine besondere Parkfunktion, das Anfahren einer Mittelstellung in der man den Robot gut wegstellen kann.

Die ausgeführten Funktionen greifen ins "Leere", d.h. es wurde keine festinstalliertes Magazin bzw. eine besondere Ablage vorgesehen. Hier ist Phantasie und Basteltalent gefragt

Die Sub Home

Zur Positionierung des Robots sind Bezugspunkte erforderlich, das ist in diesem Fall die Position an den Endtastern, die durch ein Fahren in Richtung ftiLinks erreicht werden. Von hier werden dann die weiteren Positionen bestimmt.

```
Private Sub Home()
   lblStatus = "Home"
  ft.SetMotor mSaule, ftiLinks, ftiFull, 999
  ft.SetMotor mArmV, ftiLinks, ftiFull, 999
  ft.SetMotor mArmH, ftiLinks, ftiFull, 999
  ft.SetMotor mGreifer, ftiLinks, ftiFull, 999
  ft.WaitForMotors 0, mSaule, mArmV, mArmH, mGreifer
  ft.SetMotor mSaule, ftiRechts, ftiFull, 10
  ft.WaitForMotors 0, mSaule
End Sub
```

Mit ft.SetMotor werden alle Motoren in Richtung ftiLinks mit voller Geschwindigkeit um 999 Impulse gestartet. Die 999 Impulse entsprechen etwa drei Umdrehungen der Säule, werden also nie erreicht. Erreicht wird aber der Endtaster, der ebenfalls den ft.SetMotor beendet. Die Motoren fahren alle gleichzeitig (simultan) bis sie ihre vorgegebene Position erreicht haben, hier ist es eigentlich eine Ersatzposition : die jeweiligen Endtaster.

Mit ft.WaitForMotors wird (= 0 endlos) auf das Erreichen der Home-Position gewartet. Anschließend wird die Position B angefahren (Säule 10 Schritte nach rechts).

Relative Positionierung

Die Methode ft.SetMotor erwartet bei dem Parameter Counter eine Angabe um wieviele Impulse nach ftiRechts oder ftiLinks (Angabe im Parameter Direction) zu fahren ist, immer bezogen auf die aktuelle Position. Es ist hier also eine Buchführung erforderlich mit der die absolute Position (gerechnet ab Endtaster) nachgehalten wird. In diesem Kapitel geschieht das mit Papier und Bleistift, das ändert sich im nächsten.

```
Private Sub NachA()
  lblStatus = "Nach A"
  ft.SetMotor mSaule, ftiRechts, ftiFull, 100
  ft.WaitForMotors 0, mSaule
  ft.SetMotor mArmV, ftiRechts, ftiFull, 45
  ft.SetMotor mArmH, ftiRechts, ftiFull, 80
  ft.WaitForMotors 0, mArmV, mArmH
End Sub
```

Nach dem Klick auf Home steht die Säule auf Position 10, die anderen Komponenten auf Position 0 (Arm hinten/oben, Zange offen). Hier fährt die Säule jetzt um 100 Schritte nach ftiRechts auf Position 110, es wird endlos (Parameter 0) auf Erreichen der Position gewartet.Anschließend fährt der Arm um 45 Schritte nach ftiRechts auf Posititon 45 und um 80 Schritte ftiRechts nach vorn auf Position 80. Anschließen wird auf das Erreichen der Positionen gewartet. Man könnte alle Befehle in einem Wait zusammen "abwarten". Aber manchmal stößt man mit zuviel Simultanität die zur greifenden Teile vom Sockel.

Das Greifen in Sub Greifen ist dann auch nur noch eine reines Zupacken (Schließen des Greifers um 24 Impulse).

Dann geht's zur Ablage (Sub NachB), auch wieder erst heben und dann ftiFull nach ftiLinks wieder um 100 Schritte auf Position 10.

Das (Sub) Ablegen ist dann auch nur ein Greifer aufreißen (`ft.SetMotor mGreifer, ftiLinks, ftiFull, 999`) bis zum Anschlag.

Der Programmrahmen

Das Programm basiert auf dem Template ftComputing30.VBP, ist also bekannt. Die Sub Action dient hier als Funktionsverteiler :

```
Private Sub Action()
  Do                          ' --- Warten auf Aktionen
    Select Case NrAction
    Case 1                    ' --- Home
      NrAction = 0
      Home
    Case 2                    ' --- Nach A
      NrAction = 0
      NachA
    Case 3                    ' --- Greifen
      NrAction = 0
      Greifen
    Case 4                    ' --- Nach B
      NrAction = 0
      NachB
    Case 5                    ' --- Ablegen
      NrAction = 0
      Ablegen
    Case Else
      ft.Pause 111
```

```
    End Select
  Loop Until ft.Finish()
End Sub
```

Gesetzt wird die FunktionsNummer (NrAction) durch den entsprechenden Button. Hier wird sie gleich wieder gelöscht, um einen erneuten Aufruf durch die Schleife zu verhindern. Das Sperren der Buttons könnte noch verbessert werden.

Sub cmdAction : Das Ende

Der Programmrahmen des Templates wurde beibehalten, deswegen auch die Konstruktion mit der Schleife in Sub Action. Nach dem Action-Aufruf wurde aber noch ein ft.NotHalt = False und der Aufruf von Sub Parken hinzugefügt.

Bei Klick auf den HALT-Button wird ft.NotHalt auf True gesetzt, das beendet die Schleife in Action bei der nächsten Auswertung von ft.Finish(), denn ft.Finish meldet bei ft.NotHalt = True seinerseits True. Damit ist Action beendet. ftNotHalt bleibt aber weiterhin True.

Außer Finish werten noch alle Wait-Befehle (einschl. Pause) NotHalt und ebenso die ESC-Taste aus und beenden sich im positiven Fall schlagartig. Das Modell kommt so recht schnell zum Stand.

Soll nun nach einem solchen NotHalt nochmal mit Wait-Befehlen gearbeitet werden (Sub Parken tut das), so ist ft.NotHalt wieder auf False zu setzen (beim START tut das ft.OpenInterface).

```
Private Sub Parken()
  Home
  lblStatus = "Parken"
  ft.SetMotor mSaule, ftiRechts, ftiFull, 100
  ft.WaitForMotors 0, mSaule
End Sub
```

Parken selber fährt erstmal auf Home-Position, um eine gesichterte Ausgangslage zu schaffen (der HALT-Button ist auch während der Funktionsabläufe aktiv) und von da aus den die Säule auf eine platzsparende Mittelposition, reif für die Ablage. Bei den ersten Tests sollte man Parken lieber auskommentieren, es kann auch zuviel des Guten sein.

RobCycle : Robotsteuerung durch Abruf eines Funktionszyklus

Eine Ein-Button-Lösung zu wiederholten Aufruf eines Funktionszyklus.

Das Programm hat einen leicht modifizierten Rahmen auf Basis von Template ftComputing30 und zeigt, wie man sichs mit der absoluten Positionierung leichter machen kann.

Außerdem ein kleiner Exkurs zum Umgang mit Klassen und deren Instanzen.

Und eine Anzeige der aktuellen Position

Die absolute Positionierung

Ist schon etwas vertrackter als die von SetMotor freiHaus gelieferte relative Positionierung. Hier mit MoveTo eine zentrale Routine dafür :

```
Public Sub MoveTo(Optional ByVal P1& = nullPos, _
  Optional ByVal P2& = nullPos, Optional ByVal P3& = nullPos, _
  Optional ByVal P4& = nullPos)

Dim res&

   ' --- Starten aller Motoren, wenn nicht nullPos ---
  Starten ftiM1, P1
  Starten ftiM2, P2
  Starten ftiM3, P3
  Starten ftiM4, P4

  ' --- Warten auf Ready aller Motoren und laufende Positionsangabe
  Do
    RaiseEvent OnFishPosition(WoIst(1, P1), WoIst(2, P2), _
                              WoIst(3, P3), WoIst(4, P4))
    res = ftI.WaitForMotors(80, ftiM1, ftiM2, ftiM3, ftiM4)
  Loop While res = ftiWait.ftiTime

  ' --- Aktualisieren von PosTab ----------------------------
  If res = ftiWait.ftiEnde Then
    If P1 <> nullPos Then PosTab(1).aktPosition = _
      IIf(P1 > PosTab(1).aktPosition, P1 + ftI.GetCounter(2), _
        P1 - ftI.GetCounter(2))
    ..........
  Else  ' --- Wenn MoveTo durch ESC / NotHalt abgebrochen wurde
    If P1 <> nullPos Then PosTab(1).aktPosiion = _
      IIF P1 > PosTab(1).aktPosition, P1 - ftI.GetCounter(2), _
        P1 + ftI.GetCounter(2))
    ........
  End If
```

```
  ' --- Abschließende Positionsangabe ----------------
  RaiseEvent OnFishPosition(PosTab(1).aktPosition, _
                            PosTab(2).aktPosition, _
                            PosTab(3).aktPosition, _
                            PosTab(4).aktPosition)
End Sub
```

MoveTo hat bis zu vier Parameter mit den (absoluten, ab Endtaster) Positionsangaben für die Motoren M1 – M4 (das Extension Module erfordert dann einen entsprechenden Ausbau), die letzten, nicht benötigten können weglassen werden. Werden vordere Motoren nicht verfahren, so können sie durch nullPos gekennzeichnet werden.

Ablauf : Starten aller beteiligter Motoren über Sub Starten und Warten auf das Ready aller Motoren. Diesmal aber mit Unterbrechungen (80) um sich auch noch um die aktuelle Position kümmern zu können, sie wird über ein RaiseEvent weitergegeben.

Nach dem **Warten** das Aufräumen (hier fehlt etwas vom Code) : Feststellen der Abschlußpostition in einer PosTab. Dabei wird auch ein eventueller zusätzlicher Impuls mitgezählt. Für die Fälle normales Ende (ftiEnde) und Abbruch (ftiESC, ftiNotHalt). Das wird dann auch noch mal über RaiseEvent nach außen gegeben.

PosTab ist eine Tabelle, die für jeden Motor einen Eintrag aktPosition und maxPosition enthält (abs. Positionen, Deklaration über einen Type zu Anfang der Source). Starten rechnet nun die abs. Positionen in die relativen um und starten dann den jeweiligen Motor :

```
Private Sub Starten(ByVal MotNr As ftiNr, ByVal zielPos&)
Dim ct&
  If zielPos <> nullPos Then
    If zielPos > PosTab(MotNr).maxPosition Then zielPos = _
                 PosTab(MotNr).maxPosition
    If zielPos > PosTab(MotNr).aktPosition Then
      ftI.SetMotor MotNr, ftiRechts, ftiFull, _
            zielPos - PosTab(MotNr).aktPosition
    Else
      ct = PosTab(MotNr).aktPosition - zielPos
      If ct > 0 Then ftI.SetMotor MotNr, ftiLinks, ftiFull, ct
    End If
  End If

End Sub
```

Start nur wenn nötig : If zielPos <> nullPos.

Korrektur der zielPos auf maxPosition, wenn sie zu groß ist.

Bei Rechtslauf (zielPos > aktPosition : SetMotor mit zielPos – aktPosition

Sonst : gar nicht, wenn die Differenz zur aktPosition <= 0 ist
bei >0 SetMotor mit aktPosition – zielPos (also linksrum)

Und dann gibt es noch WoIst :

```
Private Function WoIst(ByVal i&, ByVal Pos&) As Long
Dim j&
  j = (i - 1) * 2 + 2
  If Pos = nullPos Then
    WoIst = PosTab(i).aktPosition
  Else
    WoIst = IIf(Pos > PosTab(i).aktPosition, _
      Pos - ftI.GetCounter(j), Pos + ftI.GetCounter(j))
  End If
End Function
```

Da die aktPosiition erst am Ende von MoveTo upgedated wird, muß die derzeitige Position aus der gar nicht so aktuellen aktPosition und dem Counter bestimmt werden, den umFish40AX.DLL aktuell über GetCounter(j) liefert. Dabei ist das IIf eine Kurzschreibweise für If Pos>PosTab Then WoIst = ... Else WoIst = ...

Die Klasse FishRobot im KlassenModul FishRob40.CLS

Wo bringt man eine so schöne Routine nun unter, einfach so in der Form? Da sich ohnehin schon ein paar Routinen angesammelt haben und noch ein paar dazu kommen, tauft man die Routinen jetzt Methoden und bringt sie in einem KlassenModul unter. Damit hat man eine abgeschlossene (gekapselte) Einheit mit allem was dazu gehört und kann es (vielleicht?) im nächsten Projekt wieder verwenden. Man könnte die Routinen natürlich auch in dem KlassenModul von FishFace unterbringen. Aber eine separate Lösung ist für den ersten Versuch übersichtlicher.

```
Public Event OnFishPosition(ByVal Pos1&, ByVal Pos2&, ByVal Pos3&,
ByVal Pos4&)

Private Type PosSatz
  maxPosition as Long
  aktPosition as Long
End Type

Private ftI As FishFace
Private PosTab(1 To 8) As PosSatz

Public Property Set Interface(ft As FishFace)
  Set ftI = ft
End Property
........
Public Property Let maxPosition(ByVal i As ftiNr, ByVal p As Long)
  PosTab(i).maxPosition = p
End Property
.......
Public Sub MoveTo(ByVal P1&, Optional ByVal P2& = nullPos, _
  Optional ByVal P3& = nullPos, Optional ByVal P4& = nullPos)
' --- Die kennen wir ja schon
End Sub
Private Sub Starten(ByVal MotNr As ftiNr, ByVal zielPos&)
 dto....
End Sub

Private Function WoIst(ByVal i&, ByVal Pos&) As Long
 dto.....
End Function

Private Sub Class_Initialize()
   PosTab(1).maxPosition = 180
  PosTab(2).maxPosition = 100
  PosTab(3).maxPosition = 80
  PosTab(4).maxPosition = 24
End Sub
```

Der neue Klassenmodul FishRob40.CLS mit der Klasse FishRobot selber wird durch einfaches Hinzufügen Klassenmodul in das Projekt eingefügt.

Private, also nur im Klassenmodul "sichtbar" der Type für die PosTab, die PosTab selber und eine Variable ftI für die aktuelle Instanz von FishFace. Sie wird über die Eigenschaft (Property) Interface besetzt (WriteOnly).

Die Maximalpositionen des Robots in PosTab können durch die Eigenschaft maxPosition gesetzt werden, brauchen sie aber nicht, da rein zufällig in der privaten Sub Class_Initialize die Werte schon beim Anlegen der Klasse (Instanzieren) passend für den Säulenrobot gesetzt werden. Und sonst gibt es nur noch alte Bekannte (OnFishPosition kommt später).

Jetzt muß die Klasse nur noch genutzt werden. D.h. es muß eine Instanz der Klasse erzeugt werden :

RobCycle : Der Programmrahmen

Ganz oben :

```
Dim ft As New FishFace, WithEvents ftR As FishRobot
```

Anlegen einer Instanz von FishFace wie gehabt und Anlegen einer Variablen für die Klasse FishRobot WithEvents, das Event OnFishPosition soll genutzt werden. Das geht dann nicht in einem Schritt. Schritt 2 steckt dann in Form_Load : `Set ftR = New FishRobot`, da wird instanziert.

Und weils dazu gehört :

```
Private Sub ftR_OnFishPosition(ByVal Pos1&, ByVal Pos2&, _
                               ByVal Pos3&, ByVal Pos4&)
  lblStatus = Pos1 & "-" & Pos2 & "-" & Pos3 & "-" & Pos4
End Sub
```

Eine normale Ereignisroutine, die mit WithEvents ins Spiel kam, mit den Parametern der aktuellen Positionen der vier Motoren des Robots. Und dann einfach nur in lblStatus anzeigen. Ausgelöst wird das Ereignis in MoveTo so in der Mitte und am Ende : `RaiseEvent`

Bei cmdAction und cmdEnd wurde ein wenig umverteilt, da bei Action die Do --- Loop While Schleife entfallen ist. Action enthält dann nun endlich den namensgebenden Zyklus :

Sub Action : Der Zyklus

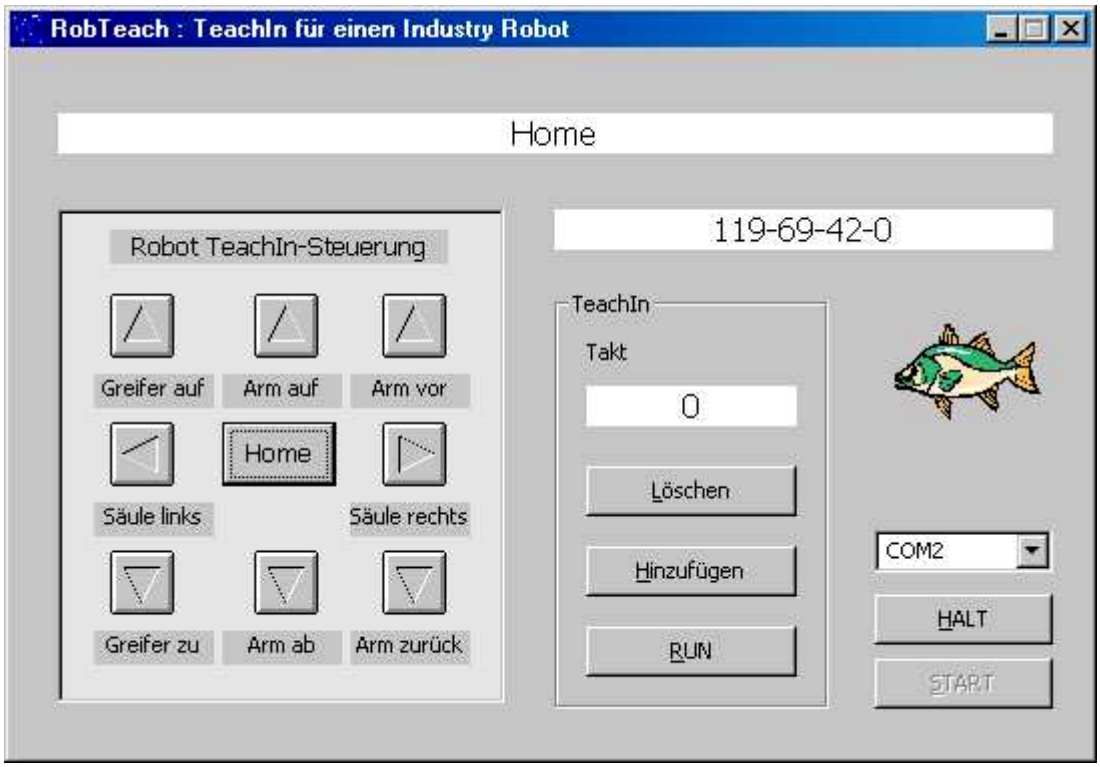
```
Private Sub Action()
  ftR.MoveTo 145
  ftR.MoveTo ftrNull, 45, 45
  ftR.MoveTo ftrNull, ftrNull, ftrNull, 24
  ftR.MoveTo ftrNull, 0
  ftR.MoveTo 45
  ftR.MoveTo ftrNull, 75
  ftR.MoveTo ftrNull, ftrNull, ftrNull, 0
End Sub
```

Immer schön der Reihe nach :

- Zum Magazin auf Position 145 (der Arm wird nicht verfahren)
- Greifen : dazu erstmal Fahren des Arms auf ftrNull (Säule bleibt auf Postion), 45, 45
  Schließen der Greifers, (man könnte hier auch kürzer `ftR.MoveTo P4:=24` schreiben)
  Einziehen des Arms auf Position 0
- Zur Ablage auf Säulenposition 45
- Zum Ablegen Arm wieder ausfahren auf Positon 75
  und Greifer aufmachen.

Ein Zyklus wird mit Button **Zyklus** gestartet.

RobTeach : TeachIn-Programm mit Maus/Tastatur-Steuerung



Die einzelnen Komponenten (Säule, Arm, Greifer) des Robots können alternativ mithilfe des Bedienfeldes bzw. des Zehnerblocks auf eine Positionen im zulässigen Arbeitsraum gesteuert werden.

Jede erreichte Position kann über den HinzuFügen-Button gespeichert werden. Die gespeicherten Positionen können über den RUN-Button nacheinanderabgefahren werden (die erste ist stets die Home-Position). Alle gespeicherten Positionen können über den Löschen-Button gelöscht werden. Die aktuelle Position wird im Positionsfeld angezeigt.

Maus/Keyboard-Steuerung

Die zentrale Routine ist KeyDown, die sowohl bei der Maus- wie auch der Tastatur-Bedienung aufgerufen wird. Der übergebene KeyCode wird über ein Select Case nach für die Bedienung relevanten Codes sortiert. Im positiven Fall wird robKey = True gesetzt und die entsprechende Robbewegung mit der Methode MoveTo gestartet :

```
Private Sub KeyDown(ByVal KeyCode%, ByVal Shift%)
  ftI.NotHalt = False
  robKey = True
  Select Case KeyCode
  Case vbKeyNumpad1            ' --- Greifer zu
    ftR.MoveTo P4:=ftR.maxPosition(ftiM4)
  Case vbKeyNumpad2            ' --- ArmVertikal ab
    ftR.MoveTo P3:=ftR.maxPosition(ftiM3)
  Case vbKeyNumpad3            ' --- ArmHorizontal zurück
    ftR.MoveTo P2:=0
  Case vbKeyNumpad4            ' --- Säule links
    ftR.MoveTo 0
  Case vbKeyNumpad5            ' --- Home
    ftR.MoveHome
  Case vbKeyNumpad6            ' --- Säule rechts
```

```
    ftR.MoveTo ftR.maxPosition(ftiM1)
  Case vbKeyNumpad7            ' --- Greifer auf
    ftR.MoveTo P4:=0
  Case vbKeyNumpad8            ' --- ArmVertikal auf
    ftR.MoveTo P3:=0
  Case vbKeyNumpad9            ' --- ArmHorizontal vor
    ftR.MoveTo P2:=ftR.maxPosition(ftiM2)
  Case Else
    robKey = False
  End Select
End Sub
```

Anmerkung : da hier nur jeweils ein Motor gestartet wird, wird die Möglichkeit genutzt, mit benannten Parametern zu arbeiten, die nicht angegebenen erhalten default-Werte. Da hier nicht bekannt ist, auf welche Position gefahren werden soll (solange wie Taste gedrückt), werden hier – in Abhängigkeit von der Richtung – die maximal Positionen angegeben.

```
Private Sub KeyUp()
   If robKey Then ftI.NotHalt = True
End Sub
```

Bei Loslassen der Maus bzw. Taste wird die unscheinbare Routine KeyUp aufgerufen, die im Fall erkannter Steuerungsfunktionen (robKey) nichts weitermacht, als NotHalt zu setzen. Die in KeyDown gestartete Methode MoveTo wird so beendet, die aktuelle Position wird intern upgedated.

Aufgerufen werden KeyDown / KeyUp von imgFunktion_MouseDown/Up der Bediensymbole und von Form_KeyDown/Up.

FishKey : Benutzersteuerelement

Verpackt werden die Routinen in einem Benutzersteuerelement. Einer Klasse mit Bedienoberfläche (einem Control). Hinzu kommen da noch einige kleinere Eigenschaften und Methoden. Man kann es ganz einfach wie eine Klasse – jetzt aber Benutzersteuerelement – anlegen. Allerdings sind größere Controls nur etwas für Spezialisten. Dies stößt schon an die Grenze dessen, was man so "eben mit" erledigen kann (Eigenschaft EnabledC ist schon Trickserei).
Hinweis : Die Form RobTeach.FRM wird gelegentlich schraffiert dargestellt, das tritt auf, wenn gleichzeitig FishKey geöffnet ist. Also einen von beidne schließen.

Zur Struktur des Gesamtprogramms : RobTeach selber nutzt die Klassen FishFace und FishRobot sowie das Control FishKey.
FishKey seinerseits greift auf FishFace und FishRobot zu (Eigenschaften Robot und Interface mit ftR und ftI). FishRobot greift auf FishFace zu (Eigenschaft Interface mit ftI).. Also schon etwas vertrackt. Wie könnte man das vereinfachen :
1. Zusammenfassen von FishFace, FishRobot und FishKey zu einem Control.
2. Zusammenfassen von FishFace und FishRobot zu einer Klasse & FishKey Control.
3. Vererbung über Polymorphie (echte Vererbung ist in VB6 nicht möglich)
4. Auf das Control verzichten und dessen Elemente in der Form der Anwendung unterbringen.

Zu 1. Das Control hat eine Oberfläche, die meist wohl nicht benötigt wird, man sollte es lieber als Extra-Control lassen.
Zu 2. ist eine praktikable Lösung, die lediglich die Komplexität der Ausgangsklasse FishFace erhöht.
Zu 3. Ist mir schlicht zu umständlich und unübersichtlich, lohnt den Aufwand kaum.
Zu 4. Ist in den Fällen sinnvoll, wo abzusehen ist, das die Zahl der Anwendungsfälle eher gering ist. Da lohnt der Einarbeitungsaufwand nicht. Vorschlag : FishKey-Elemente in Form von FishTeach integrieren, FishRobot so lassen (FishFace bleibt dann klarer). Beim zweiten Programm kann man dann ja bei sich selber abschreiben.

cmdAdd : Speichern von Positionen

FishTeach enthält eine Tabelle `Takte(1 To 100, 1 To 4)`, die pro Zeile die Position aller Komponenten des Robots in der Reihenfolge M1 – M4 speichern kann. Das geschieht schlicht mit der Sub TaktAdd :

```
Private Sub TaktAdd()
   If TaktAnz > UBound(Takte) Then Exit Sub
  TaktAnz = TaktAnz + 1
  Takte(TaktAnz, 1) = ftR.aktPosition(ftiM1)
  Takte(TaktAnz, 2) = ftR.aktPosition(ftiM2)
  Takte(TaktAnz, 3) = ftR.aktPosition(ftiM3)
  Takte(TaktAnz, 4) = ftR.aktPosition(ftiM4)
  lblTaktAnz = TaktAnz
End Sub
```

Dazu wird die entsprechende Eigenschaft aktPosition der Klasse FishRobot herangezogen. Aufgerufen wird TaktAdd über cmdAdd, die aktuelle TaktNr wird in lblTaktAnz angezeigt.

```
Private Sub TaktClear()
   TaktAnz = 1
  Takte(1, 1) = 0
  Takte(1, 2) = 0
  Takte(1, 3) = 0
  Takte(1, 4) = 0
End Sub
```

In der vorliegenden Form ein Löschen durch ein Zurücksetzen von TaktAnz und gleichzeitigem Eintrag der Home-Position als Takt 1, das muß nicht sein. TaktAnz = 0 reicht auch.

cmdRun : Der Roboter tut was in der Tabelle steht

Genutzt wird die Takte-Tabelle in cmdRun_Click :

```
Private Sub cmdRun_Click()
Dim i&
On Error GoTo runFehler

  ft.NotHalt = False
  lblStatus = "--- RUN ---"
  For i = 1 To TaktAnz
    ftR.MoveTo Takte(i, 1), Takte(i, 2), Takte(i, 3), Takte(i, 4)
    lblTaktAnz = i
  Next i
  Exit Sub

runFehler:
  lblStatus = Err.Number & " : " & Err.Source & "." _
              & Err.Description
  ft.CloseInterface
  cmdEnde.Caption = "&Abbrechen"
End Sub
```

Kern ist die For Next i Schleife in der über die Methode ftR.MoveTo von FishRobot eine Position nach der anderen angefahren wird. Dabei wird die aktuelle TaktNr in lblTaktAnz angezeigt.

Drumrum gibt es dann noch ein eigenes On Error Goto, da das On Error von cmdAction hier nicht mehr wirkt. cmdRun und cmdAction stehen auf gleicher Ebene der Aufrufhierarchie (sie rufen sich nicht auf). Genau genommen fehlen noch On Errors für das TeachIn. Außerdem

wieder ein ft.NotHalt = False, um einen vorhergehenden Abbruch (aus der TeachIn-Phase) aufzuheben.

RobTeach : Der Programmrahmen

Der Programmrahmen basiert wieder auf dem modifizierten Template ftComputing30 und ist ähnlich geändert worden (cmdAction -> cmdEnde) wie RobCycle. Es wird hier noch etwas mehr gesperrt um Fehlbedienungen zu vermeiden : das ftK-Control und der Rahmen fraTeachIn als Ganzes.

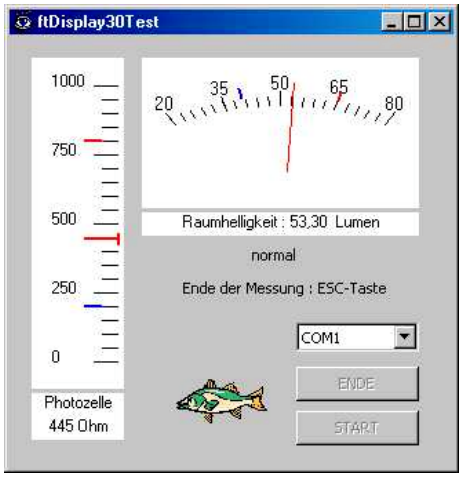
cmdAction enthält die Eigenschaftszuweisung `Me.KeyPreview = True`. Sie ist erforderlich um die TeachIn-Bedienung über den Zehnerblock zu ermöglichen. Der Zehnerblock wird über `Form_KeyDown/Up` mit `ftK.KeyDownControl / ftK.KeyUp_Control` angesteuert

Das Programm ist noch kräftig erweiterungsfähig :

1. Löschen einzelner Positionen anstelle der gesamten Takte-Tabelle
2. Anfahren einzelner Positionen in der Takt-Tabelle.
3. Speichern und Wiedereinlesen der Takte-Tabelle.
4. ---???---

Analog-Instrumente

Analog.CTL : Analog-Anzeige von AX / AY



Analog40.CTL ist ein UserControl mit dem die Werte der Analog-Eingänge des fischertechnik Interfaces flexibel dargestellt werden können. Dazu muß es mit dem Rohwert (rawValue) von AX bzw. AY in regelmäßigen Abständen (z.B. über einen Timer) aufgerufen (**ShowValue**) werden.
Die Darstellung erfolgt über die **DisplayType**n **ftiSegment** (Kreisauschnitt) und **ftiVertical** (senkrechte Skala). Die Anzeigen sind in der Größe flexibel. Am unteren Rand können sie beschriftet (**Description**) werden. Zusätzlich wird hier der aktuelle Meßwert digital angezeigt, die Maßeinheit kann angegeben werden (**Units**).

Die Meßwerte können als Rohwerte, d.h. so wie sie vom Interface kommen, angezeigt werden. Dabei kann der Anzeigebereich gewählt werden (**MinValue**, **MaxValue**). Außerdem können sie auf einen wählbaren Wertebereich linear abgebildet werden (**MinUnit**, **MaxUnit**). Bei Bedarf kann die Anzeigerichtung vertauscht werden (**Exchanged**). Auf diese Weise ist es z.B. möglich, die von einer Photozelle gemessenen Werte im Bereich von 0 bis 1000 auf Helligkeitswerte im Bereich 20 bis 80 Lumen abzubilden. Da die Photozelle bei größerer Helligkeit kleinere Werte anzeigt, wird die Anzeigerichtung vertauscht (Die Beispielwerte wurden mit einer kleinen Photozelle gemessen, die Umrechnung in Lumen ist erfunden).

Bei Erreichen bestimmbarer Meßwerte (**LowArea**, **HighArea**) werden Ereignisse ausgelöst : **GotLow**, **GotNormal**, **GotHigh** bzw. **IsLow**, **IsNormal**, **IsHigh** auf die in den entsprechenden Ereignisroutinen reagiert werden kann. Der Aufruf der Ereignisroutinen kann durch **EventsActivated** gesteuert werden.

Eigenschaften

Description	anzuzeigender Meßwertname
DisplayType	Typ des anzuzeigenden Meßinstruments
EventsActivated	Aktivieren der Ereignisroutinen
Exchanged	Vertauschen der Anzeigerichtung
HighArea	Beginn des Bereichs der hohen Meßwerte
LowArea	Ende des Bereichs der niedrigen Meßwerte
MaxUnit	max. anzuzeigender Wert
MinUnit	min. anzuzeigender Wert
Units	anzuzeigende Meßwerteinheit

Methoden

ShowValue	Anzeige des Instument mit aktuellem Meßwert
ValueToUnit	Umrechnung eines rawValues (Rohwertes, original Meßwertes) in einen unitValue (anzuzeigenden Wert)

Ereignisse

GotHigh	akt. Meßwert in die HighArea eingetreten
GotLow	akt. Meßwert in die LowArea eingetreten
GotNormal	akt. Meßwert in die NormalArea eingetreten
IsHigh	akt. Meßwert lieg in der HighArea
IsLow	akt. Meßwert liegt in der LowArea
IsNormal	akt. Meßwert liegt in der NormalArea

Beispiel : Testrahmen

Im Beispiel werden die Meßwerte eines Photowiderstandes (32 698) zweimal dargestellt : links mit DisplayType = ftiVertical die Originalmeßwerte des Photowiderstandes, rechts die auf eine (fiktive) Raumhelligkeit umgerechneten Werte mit DisplayType = ftiSegment. Unter dem rechten Display wird zusätzlich angezeigt, ob sich der Meßwert im Bereich normal, LOW oder HIGH befindet. Die Grenzwerte sind außerdem in den Skalen mit blau (LOW) und rot (HIGH) markiert. Das eigentliche Programm konzentriert sich auf die Routine cmdAction_Click :

```
Private Sub cmdAction_Click()

On Error GoTo ftiFehler

  ana.MinUnit = 20
  ana.MaxUnit = 80
  ana.MinValue = 0
  ana.MaxValue = 1000
  ana.Exchanged = True

  ft.OpenInterfaceUSB ftiROBO_first_USB, 0

  cmdEnde.Enabled = False
  cmdAction.Enabled = False

  ana.EventsActivated = True
  lblStatus = "Ende der Messung : ESC-Taste"

  ' --- Zyklische Abfrage der Werte von EX, Ende : ESC-Taste ---

   Do
     anb.ShowValue ft.GetAnalog(ftiAX)
     ana.ShowValue ft.GetAnalog(ftiAX)
     ft.Pause 200
  Loop Until ft.Finish(ftiI1)

  ft.CloseInterface
  ana.ShowValue 0
  cmdAction.Enabled = True
  cmdEnde.Enabled = True
  cmdEnde.SetFocus
  Exit Sub

ftiFehler:
  lblStatus = Err.Number & " : " & Err.Source & "." _
       & Err.Description
End Sub
```

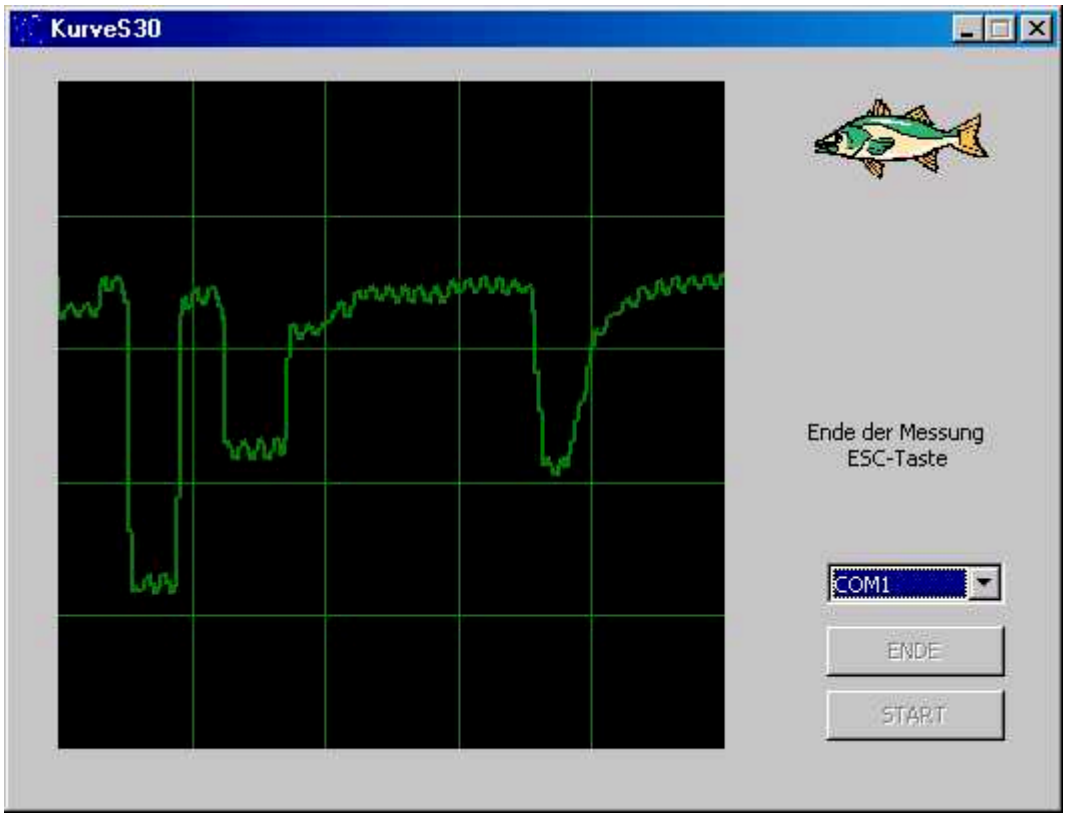
- Das Beispielprogramm ist in einer Programmgruppe (ftDisOCX.vbg) abgelegt mit separaten Projekten für das ftDisplay.OCX (ftDisplay.vbp / Analog.ctl) und den Testrahmen (ftDisplayTest.vbp / ftAnalogTest.FRM). Es kann auch in einem einzigen Projekt ftDisplayTest.vbp mit ftAnalogTest.FRM und Analog.CTL abgelegt werden. Beim Importieren von Analog.CTL wird der Scope dann automatisch auf private umgestellt (VB_Exposed = False).
- FishFa40AX.DLL : Verknüpfung im Menü Projekt herstellen.
- Zwei Instanzen von Analog.ctl auf der Form plazieren (ana und anb) und die Eigenschaften : `Description = "Photozelle", Units = "Ohm", DisplayTypes = 2, LowArea = 200, HighArea = 800` für anb und für ana die Eigenschaften : `MinUnit = 40, MaxUnit = 80, Exchanged = -1 'True, Description = "Raumhelligkeit", Units = " Lumen" HighArea = 65` einstellen.
- ana.MinUnit = 20 .... überschreiben die entsprechenden in der Eigenschaftsliste eingestellten Eigenschaften (nur zu Demozwecken).
- ft.OpenInterfaceUSB ftiROBO_first_USB, 0 ' --- Open des gewünschten Interfaces
- ana.EventsActivated = True Die Eventroutinen werden aktiviert.
- Do ... Loop Until ft.Finish(ftiI1) Endlosschleife zur Abfrage und Anzeige der Analogwerte. Ende der Schleife durch ESC-Taste oder I1 = True

Ereignisroutinen

```
Private Sub ana_GotHigh(ByVal aktValue As Single)
lblArea = "HIGH"
lblArea.BackColor = vbRed
End Sub

Private Sub ana_GotLow(ByVal aktValue As Single)
lblArea = "LOW"
lblArea.BackColor = vbBlue
End Sub

Private Sub ana_GotNormal(ByVal aktValue As Single)
lblArea = "normal"
lblArea.BackColor = &H8000000F
End Sub
```

In den Ereignisroutinen ana_GotHigh, _GotLow, _GotNormal wird nichts weiter gemacht als in lblArea den aktuellen Wertebereich anzuzeigen. Die Ereignisroutinen werden nur aufgerufen, wenn sich der Wertebereich ändert, also nicht bei jedem Meßwert (das wären dann IsHigh ...).

KurveS40 : Erfassen von Meßwerten als Kurven



Erfassen von 1000 Werten des Analogeinganges EX und deren Darstellung als Kurve in Abhängigkeit von der Zeit. Verwendet wurde ein Photowiderstand.
Hinweis : Die Erfassung der Werte erfolgt in einer For ... Next-Schleife. Gezeichnet wird in eine PictureBox. Das Zeichnen des Gitters geschieht in einem separaten Unterprogramm. Interface und Interfaceanschluß werden über eine ComboListe eingestellt.

Die HauptRoutine : cmdAction_Click

```
Private Sub cmdAction_Click()

Dim i&, lastY&, lastX&, actX&, actY&

On Error GoTo ftiFehler

  ft.OpenInterfaceUSB ftiROBO_first_USB, 0

  cmdAction.Enabled = False
  cmdEnde.Enabled = False
  Running = True
  lblStatus = "Ende der Messung ESC-Taste"

' --- Festlegen Maßstab : Beginn X/Y, linke obere Ecke : 0,0
'                         Ende X/Y, rechte untere Ecke : 1000,1000
  picPlot.Scale (0, 0)-(1000, 1000)

' --- Endlosschleife bis ESC-Taste mit 32 MilliSekunden Pause ---
  actWert = 0
  For i = minWert To maxWert
    Werte(i) = maxWert
  Next i
```

```
  ft.Pause 300
  Do
    Werte(actWert) = ft.GetAnalog(ftiAX)
    If Werte(actWert) > 1000 Or Werte(actWert) < 1 Then _
       Werte(actWert) = maxWert
    ShowValues
    actWert = IIf(actWert = maxWert, minWert, actWert + 1)
    ft.Pause 32                 ' --- 32 MilliSekunden Pause
  Loop Until ft.Finish(ftiI1)

  ft.CloseInterface
  cmdAction.Enabled = True
  cmdEnde.Enabled = True
  Running = False
  cmdEnde.SetFocus
  Exit Sub

ftiFehler:
  lblStatus = Err.Number & " " & Err.Source & " " & Err.Description

End Sub
```

Fast das gesamte Programm spielt sich in der Sub cmdAction_Click ab.

1. Im ersten Teil werden die **Variablen** lastX ...definiert, die später beim Zeichnen verwendet werden.
2. Verwendet wird die **FishFa40AX.DLL**, Name : ft.
3. **OpenInterfaceUSB** stellt die Verbindung zum Interface her. Der PortName wird dabei der ComboListe cboPortName entnommen. Schlägt die Verbindung fehl, wird eine entsprechende Nachricht ausgegeben und die Sub abgebrochen.
4. Gezeichnet wird in die **PictureBox** picPlot. Als Vorbereitung dazu wird mit **picPlot.Scale** der Zeichenmaßstab gesetzt. Etwas ungewohnt : Werte für die linke obere und rechte untere Ecke. Anschließend wird noch die Zeichenfläche gelöscht : picPlot.Cls.
5. Das **Gitter** wird dann in einer separaten Sub gezeichnet. Hier werden noch die Startpunkte für die Kurve (lastX = 0, lastY = 0), die Linienstärke festgelegt (picPlot.DrawWidth = 2) und der Startpunkt gezeichnet (picPlot.Line).
6. Dann wird die Schleife **Do ... Loop Until ft.Finish(ftil1)** gestartet in der die Analogwerte erfaßt und ausgegeben werden.
7. Mit **actWert = ft.GetAnalog(ftiAX)** wird der aktuelle Meßwert bestimmt und mit If actWert ... auf den Bereich 1 ... 1000 begrenzt.
8. Das eigentliche Zeichnen erfolgt in ShowValues
9. Nach Schleifen-Ende folgt nur noch ein **CloseInterface** und ein Setzen des Fokus auf den Ende-Button um den Abschluß des Plottens anzuzeigen.

Das war auch schon die Hauptarbeit. Jetzt noch die Routine zum Zeichnen des Gitters :

Das Zeichnen der Werte und des Gitters : ShowValues

```
Private Sub ShowValues()

Dim i&, j&, lastX&, lastY&

' --- Zeichnen Gitter ----------------------
  picPlot.Visible = False
  picPlot.Cls
  picPlot.DrawWidth = 1
  For i = 200 To 800 Step 200
    picPlot.Line (i, 0)-(i, 1000), QBColor(2)
  Next i
```

```
  For i = 200 To 800 Step 200
    picPlot.Line (0, i)-(1000, i), QBColor(2)
  Next i

' --- Zeichnen Werte ------------------------
  i = 0
  lastX = i
  lastY = Werte(actWert)
  picPlot.DrawWidth = 2
  For j = actWert To maxWert
    picPlot.Line (lastX, lastY)-(i, Werte(j)), QBColor(2)
    lastX = i
    lastY = Werte(j)
    i = i + deltaWert
  Next j
  For j = minWert To actWert
    picPlot.Line (lastX, lastY)-(i, Werte(j)), QBColor(2)
    lastX = i
    lastY = Werte(j)
    i = i + deltaWert
  Next j
  picPlot.Visible = True

End Sub
```

1. Strichbreite picPlot.Drawidth = 1
2. Senkrechte Striche
3. Waagerechte Striche jeweils in einer For ... Next-Schleife
4. Das Zeichnen der Meßwert-Linie erfolgt in einer For Next Schleife mit der Methode picPlot.Line. Dabei wird immer der letzte gezeichnete Wert (lastX/Y) mit dem nächsten Punkt aus der Werte-Tabelle verbunden.
5. Anschließend wird lastX/Y gesetzt.

Schrittmotoren

Es ist geplant, eine Schrittmotorenansteuerung in umFish40.DLL zu integrieren. Bis dahin sollte die Beschreibung in FishFa30VB.PDF reichen. Der Umstellungsaufwand ist gering.

Referenz

Allgemeines

Verwendete Parameterbezeichnungen

In der Referenz werden für Parameter und Returnwerte besondere Bezeichnungen verwendet um deren Bedeutung zu charakterisieren. Sie stehen gleichzeitig für einen Variablentyp bzw. alternativ eine Enum.

AnalogNr	Nummer eines Analog-Einganges (Long 0-1, ftiInp)
AnalogWert	Rückgabewert beim Auslesen von AX/AY (AXS1 / AXS2 / AXS3) (0-1024)
AnalogZyklen	Anzahl der Zyklen nach dem die Analogwerte ausgelesen werden (nur Intelligent Interface, Long, typisch 5)
Code	Angabe mit welcher Code-Taste die Eingaben des IR-Senders ausgewertet werden sollen (ftiIRCode).
ComNr	Nummer des COM-Ports für eine Interface-Verbindung (ftiPorts).
Counter	Wert eines ImpulsCounters (Long)
Direction	Schaltzustand eines M-Ausganges (Long 0-2, ftiDir)
ifTyp	Typ des angeschlossenen Interfaces (ftiInterfaces)
InputNr	Nummer eines E-Einganges (Long 0-8(32), ftiInp)
InputStatus	Rückgabewert beim Auslesen aller I-Eingänge (0 - &HFFFFFFFF)
KeyNr	Nummer der vom IR-Sender erwerteten Taste (ftiIRKeys)
LampNr	Nummer eines O-Ausganges ("halber"-M-Ausgang) (Long 0-8(32), ftiOut)
ModeStatus	Status der Betriebsmodi aller M-Ausgänge(Long). Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 normal, 01 RobMode).
MotorNr	Nummer eines M-Ausganges (Long 0-4(16), ftiOut)
MotorStatus	SollStatus aller M-Ausgänge (Long). Jeweils 2 bit pro Ausgang Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts) Bei O-Ausgängen kann jedes bit einzeln gesetzt werden.
mSek	Zeitangabe in MilliSekunden (Long)
NrOfchanges	Anzahl Impulse (Long)
OnOff	Ein/Ausschalten eines M- oder O-Ausganges (ftiDir)
Position	Position in Impulsen ab Endtaster (Long)
SerialNr	Standardseriennummer eines ROBO-Interfaces

Speed	Geschwindigkeitsstufe mit der ein M-Ausgang (Motor) betrieben werden soll (Long 0-7, ftiSpeed)
SpeedStatus	Status der Geschwindigkeiten aller M-Ausgänge (Long) Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1 Werte 0000 (stop) – 0111 (full), für die Motoren M9 – M16 : SpeedStatus16
TermInputNr	Nummer eines E-Einganges mit der die (Wait)Methode beendet werden soll (Long, ftiInp)
Value	Allgemeiner Long Wert
VoltNr	Nummer eines analogen Spannungseinganges (A1, A2, AV, ftiInp)
WaitWert	Rückgabewert von WaitForMotors (Long, ftiWait)

Die Aufrufparameter werden ByVal (Ausnahme WaitForPosition, Parameter Counter) übergeben.

Eine Reihe von Parametern sind optional, sie werden dann meist im Sinne einer Funktions-Überladung (Overload) verwendet. Das wird dann bei der betreffenden Methode besonders beschrieben.

Aufzählungen (Enums)

ftiInterfaces	Bezeichnung der anschließbaren Interfaces
ftiPorts	Bezeichnung der unterstützten Ports
ftiDir	Angabe des Schaltzustandes (Drehrichtung, Ein/Aus)
ftiInp	Angabe der Nummer eines Einganges
ftiOut	Angabe der Nummer eines Ausganges
ftiIRCode	Auswertungsart beim IR-Sender
ftiIRKeys	Getätigte Taste am IR-Sender
ftiSpeed	Angabe einer Geschwindigkeitsstufe
ftiWait	Returnwerte der Methode WaitForMotors

Die Enums können als Parameter parallel zu den numerischen Parameterwerten (Long) angegeben werden.

Eigenschaften

ActDeviceName

Name des angeschlossenen Interfaces

ActDeviceType

Typ des angeschlossenen Interfaces

ActDeviceSerialNr

Standardseriennummer des angeschlossenen Interfaces

ActDeviceFirmware

Firmwarestand des angeschlossenen Interfaces

NotHalt

Anmelden eines Abbruchwunsches, Auswertung durch die Wait-Methoden und Finish

Get | Set, Boolean, Default = False

Outputs

Lesen Werte aller M-Ausgänge.

Get, Long | MotorStatus

Version

Lesen der Version von FishFa40AX.CLS/DLL

Get, String

Methoden

Allgemeines

ft

Bezeichnung für die zugehörende Klasseninstanz

DoEvents

Herstellen der Unterbrechbarkeit durch den Befehl DoEvents. Bei Einsatz von engen Schleifen, z.B. Abfrage I-Eingang, kann die Bedieneroberfläche blockieren (Button-Click wird nicht ausgeführt, Label werden nicht aktualisiert). Um das zu verhindern wurde in die Mehrzahl der Methoden der VB-Befehl DoEvents eingebaut.

Wird bei den betroffenen Methoden extra angegeben.

Abbrechbar

Länger laufende Methoden (Wait...) können von außen durch Setzen der Eigenschaft NotHalt = True oder durch Drücken der Esc-Taste abgebrochen werden.

Wird bei den betroffenen Methoden besonders angegeben.

RaiseEvents

Werden Ausnahmebedingungen festgestellt, wird ein entsprechendes Ereignis ausgelöst (RaiseEvent) :

30002 : "Kein Open", die Methode erfordert ein vorhergehendes erfolgreiches OpenInterface
30001 : "Interface Problem", im laufenden Betrieb trat ein Problem beim interface auf (Stromversorgung, keine Verbindung zum Port).

Das Err-Object enthält außerdem als Herkunftsbezeichnung (Err.Source) "FishFace.methodennamen"

Wird bei den betroffenen Methoden besonders angegeben.

Beispiel

```
On Error Goto ftiFehler

   ft.SetMotor ftiM1, ftiLinks
   ft.WaitForChange ftiI2, 100
   ft.SetMotor ftiM1, ftiAus
   ....

ftiFehler:
lblStatus.Caption = Err.Number & " : " Err.Source & "." & _
     Err.Description
```

Motor an M-Ausgang M1 wird linksdrehend gestartet, es wird auf 100 Impulse an E-Eingang I2 gewartet und dann der Motor wieder abgeschaltet. Tritt in dieser Zeit eine Ausnahme auf (z.B. Ausfall der Spannungsversorgung), wird das in lblStatus angezeigt.

NotHalt

Die Eigenschaft NotHalt (siehe auch "Abbrechbar") wird intern genutzt um langlaufende Funktionen ggf. Abzubrechen. NotHalt wird von OpenInterface auf False gesetzt. Es kann im Programm (z.B. über einen HALT-Button) genutzt werden um den Programmlauf

abzubrechen oder auch eine Endlosschleife (z.B. Do ... Loop Until ft.Finish) zu beenden. Soll das Programm anschließend weiterlaufen, so ist NotHalt wieder auf False zu setzen.

Parameter

Siehe auch "Verwendete Parameterbezeichnungen"

Die Werteübergabe erfolgt ByVal (hier nicht besonders angegeben)

Die Methoden sind meist Subs, bei Functions wird der Ergebniswert vorangestellt.

Als Parameter können im allgemeinen Enum-Werte angegeben werden, das wird bei der Methode beschrieben. Gleichermaßen ist aber die Angabe von Long-Werten möglich. Davon abweichende Typen werden extra angegeben.

Speed

Die Geschwindigkeitssteuerung beruht auf einem zyklischen Ein- und Ausschalten der betroffenen M-Ausgänge (Motoren). Die Geschwindigkeitsstufe wird durch die Parameter Speed (1-7) bzw. SpeedStatus für einen bzw. alle Motoren bei der Methode SetMotor(s) bestimmt.

Counter

Zu jedem I-Eingang wird ein Counter geführt, in dem die Impulse (Umschalten von True auf False und umgekehrt) gezählt werden. Die Counter werden bei OpenInterface auf 0 gesetzt. Sie werden außerdem von einigen Methoden interngenutzt (SetMotor, WaitForxxx). Sie können mit GetCounter abgefragt und mit SetCounter / ClearCounter(s) gesetzt werden. In der Regel werden sie zur Positionsbestimmung eingesetzt.

RobMotoren

Unter RobMotoren wird eine Kombination von einem M-Ausgang und zwei I-Eingängen mit den Funktionen Endtaster / Impulstaster verstanden, die im Betrieb eine Einheit bilden. Dabei muß am M-Ausgang ein Motor angeschlossen sein und an den I-Eingängen Taster mit Schließfunktion (Kontakte 1-3). Auf der Motorwelle muß ein "Impulsrädchen" montiert sein, das den Impulstaster betätigt. Der Motor muß linksdrehend angeschlossen werden. D.h. er läuft bei ftiLinks auf den Endtaster zu. Folgende Kombinationen sind zulässig

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

Bei entsprechend vielen angeschlossenen Extensions, weiter bis Motor 16.

Die RobMotoren können über die Methode
`SetMotor MotorNr, Direction, Speed, Counter`
betrieben werden. Die Methode erlaubt das simultane Anfahren vorgegebener Positionen mit bis zu 8 Motoren. Bei Erreichen einer Position wird der zugehörende Motor abgeschaltet. Die Methode ist asynchron. D.h.Das Erreichen der vorgegebenen Positionen wird von der Methode nicht abgewartet (der Ausführungsteil der Methode läuft in einem separatem Thread). Die Synchronität zum Programm kann durch die Methode WaitForMotors wieder hergestellt werden.

Beispiel

```
ft.SetMotor ftiM1, ftiLeft, ftiHalf, 50
ft.SetMotor ftiM2, ftiRight, ftiFull, 60
ft.WaitForMotors 0, ftiM1, ftiM2
```

Motoren M1 und M2 werden gestartet, anschließend wird auf das Erreichen der Positionen gewartet.

O-Ausgänge am Interface

Die ROBO-Interfaces können die beiden Pole eines M-Ausganges einzeln schalten. Sie werden deswegen auch zusätzlich mit Ox bezeichnet. Geräte für die keine Umpolung im laufenden Betrieb erforderlich ist – Lampen, Magnete, aber auch teilweise Motoren – können an einen O-Ausgang und Masse angeschlossen werden. Dadurch wird die Schaltkapazität eines Interfaces deutlich erhöht. Geschaltet werden sie mit der Methode SetLamp.

Liste der Methoden

ClearCounter

Löschen des angegebenen Counters (0)

ft.**ClearCounter** InputNr

Raise 30001, 30002

Siehe auch ClearCounters, GetCounter, SetCounter

ClearCounters

Löschen aller Counter (0)

ft.**ClearCounters**

Raise 30001, 30002

Siehe auch ClearCounter, GetCounter, SetCounter

ClearMotors

Abschalten aller M-Ausgänge (ftiAus)

ft.**ClearMotors**

Raise 30001, 30002

Siehe auch SetMotor, SetMotors, SetLamp, Outputs

CloseInterface

Schließen der Verbindung zum Interface

ft.**CloseInterface**

Raise 30001

Siehe auch OpenInterfacexxx

Finish

Feststellen eines Endewunsches (NotHalt, Esc-Taste, I-Eingang)

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

Raise 30001, 30002, DoEvents

Siehe auch GetInput, GetInputs, FinishIR

Beispiel

```
Do
   ....
Loop Until ft.Finish(ftiI1)
```

Die Do Loop-Schleife wird solange durchlaufen, bis entweder ft.NotHalt = True, die Esc-Taste gedrückt oder I1 = True wurde.

FinishIR

Feststellen eines Endewunsches (NotHalt, Esc-Taste, IRKey)

Boolean = ft.**Finish**(IRCode, IRKey)

Raise 30001, 30002, DoEvents

Siehe auch GetInput, GetInputs, Finish

Beispiel

```
Do
   ....
Loop Until ft.FinishIR(fticode1, ftiM3L)
```

Die Do Loop-Schleife wird solange durchlaufen, bis entweder ft.NotHalt = True, die Esc-Taste gedrückt oder M3L am IR-Sender gedrückt wurde.

GetAnalog

Feststellen eines Analogwertes (AX, AY (AXS1, AXS2, AXS3)).
Es wird der intern vorliegende Wert ausgegeben. Beim Intelligent Interface ist die AnalogZyklen-Angabe beim OpenInterfaceCOM erforderlich.

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

Raise 30001, 30002

Siehe auch GetVoltage

Beispiel

```
lblAnalog.Caption = ft.Analog(ftiAX)
```

Dem Label lblAnalog wird der aktuellen Wert von EX zugewiesen.

GetCounter

Feststellen des Wertes des angegebenen Counters

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

Raise 30001, 30002

Siehe auch SetCounter, ClearCounter, ClearCounters

Beispiel

Beipiel

```
lblPosTurm.Caption = ft.GetCounter(ftiI2)
```

Dem Label lblPosTurm wird der aktuelle Zählerstand der dem I-Eingang I2 zugeordnet ist, zugewiesen.

GetInput

Feststellen des Wertes des angegebenen I-Einganges

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

Raise 30001, 30002, DoEvents

Siehe auch GetInputs, GetIRKey, Finish

Beispiel

```
If ft.GetInput(ftiI1) Then
   ...
Else
   ...
EndIf
```

Wenn der I-Eingang I1 (Taster, PhotoTransistor, Reedkontakt ...) = True ist, wird der Then-Zweig durchlaufen.

GetInputs

Feststellen der Werte aller E-Eingänge

InputStatus = ft.**GetInputs**()

Raise 30001, 30002, DoEvents

Siehe auch GetInput, IRKey, Finish, FinishIR

Beispiel

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

Wenn die I-Eingänge I1 oder I3 True sind, wird der Then-Zweig ausgeführt.

GetIRKey

Feststellen des Wertes des angegebenen IR-Einganges. Die Code-Tasten der IR-Bedienung werden wahlweise ausgewertet.

Boolean = ft.**GetIRKey**(Code, KeyNr)

Raise 30001, 30002, DoEvents

Siehe auch GetInputs, Finish, FinishIR

Beispiel

```
If ft.GetIRKey(ftiCode1, ftiM2L) Then
   ...
Else
   ...
EndIf
```

Wenn die IR-Taste M2L = True ist und Code1 aktiv ist, wird der Then-Zweig durchlaufen.

GetVoltage

Feststellen des Spannungswertes des angegebenen A-Einganges. Beim Intelligent Interface ist die AnalogZyklen-Angabe beim OpenInterfaceCOM erforderlich.

VoltWert = ft.**GetVoltage**(VoltNr)

Raise 30001, 30002, DoEvents

Siehe auch GetAnalog, OpenInterfaceCOM

Beispiel

```
lblVolt.Caption = ft.GetVoltage(ftiA1)
```

Dem Label lblVolt wird der aktuelle Wert von A1 zugewiesen.

OpenInterfaceUSB

Herstellen der Verbindung zum Interface. OpenInterface muß als erste Methode aufgerufen werden.

ft.**OpenInterfaceUSB**(ifTyp, SerialNr)

- ifTyp : Interface Typ der Aufzählung ftiInterfaces, (_IF_USB, _IF_Over_RF, _IO_Extension).
  ftiROBO_first_USB steht für das erste an USB gefundene Interface. Empfiehlt sich, wenn nur mit einem Interface gearbeitet wird. Die SerialNr kann dann auf 0 gesetzt werden.
- SerialNr (Standardseriennummer) : Kennzeichnung gleichartiger Interfaces durch eine – freivergebbare – laufende Nummer, die in das Interface geschrieben wurde (z.Zt. nur durch ROBO Pro).

Raise 3001

Siehe auch OpenInterfaceCOM, CloseInterface

Beispiel

```
On Error Goto To OpenFehler
   ft.OpenInterfaceUSB ftiROBO_first_USB, 0

   ....
OpenFehler:
  lblStatus.Caption = Err.Number & " : " Err.Source & "." & _
     Err.Description
```

Herstellen einer Verbindung zum ersten ROBO Gerät an USB. Im Fehlerfall wird "`30001 : FishFace.OpenInterface.Interface Problem`" ausgegeben.

OpenInterfaceCOM

Herstellen der Verbindung zum Interface. OpenInterface muß als erste Methode aufgerufen werden.

ft.**OpenInterface**(ifTyp, ComNr, AnalogZyklen)

Der Parameter AnalogZyklen ist optional (default = 0) :

- ifTyp : InterfaceTyp der Aufzählung fitInterfaces (ftiIntelligent.., *_IF_*IIM, *I*F_COM)
- ComNr : Nummer des COM-Portes an dem das Interface angeschlossen ist (z.B. COM1 = 1).
- AnalogZyklen (Long) : Angabe, ob auch die Analogeingänge (AX / AY...) ausgewertet werden sollen. Wert 0 keine Analogeingänge, typische Werte 5 – 10, größere sind möglich. Bedeutet, das die Analogeingänge jeden 5 oder 10. Zyklus abgefragt werden.

Raise 30001

Siehe auch OpenInterfacUSB, CloseInterface

Beispiel

```
On Error Goto To OpenFehler
   ft.OpenInterfaceCOM ftiIntelligent_IF, 1, 5

   ....
OpenFehler:
  lblStatus.Caption = Err.Number & " : " Err.Source & "." & _
     Err.Description
```

Herstellen einer Verbindung zum Intelligent Interface an COM1, die Analog-Eingänge sollen jedes 5te mal gescannt werden. Im Fehlerfall wird "`30001 : FishFace.OpenInterface.Interface Problem`" ausgegeben.

Pause

Anhalten des Programmablaufs.

ft.**Pause** mSek

Raise 30001, 30002, DoEvents, Abbrechbar

Siehe auch WaitForTime

Beispiel

```
ft.SetMotor ftiM1, ftiLinks
ft.Pause 1000
ft.SetMotor ftiM1, ftiAus
```

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

SetCounter

Setzen eines Counters

ft.**SetCounter** InputNr, Value

Raise 30001, 30002

Siehe auch GetCounter, ClearCounter, ClearCounters

SetLamp

Setzen eines O-Ausganges (eines "halben" M-Ausganges). Anschluß einer Lampe oder eines Magneten ... an einen O-Ausgen und Masse.

ft.**SetLamp** LampNr, OnOff, Power

Der Parameter Power ist optional (default = 7)

Raise 30001, 30002

Siehe auch SetMotor, SetMotors, ClearMotors

Beispiel

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

  ft.SetLamp lGruen, ftiEin
  ft.Pause 2000
  ft.SetLamp lGruen, ftiAus
  ft.SetLamp lGelb, ftiEin
```

Die grüne Lampe an O1 und Masse wird für 2 Sekunden eingeschaltet und anschließend die gelbe an O2

SetMotor

Setzen eines M-Ausganges (Motor)

ft.**SetMotor** MotorNr, Direction, Speed, Counter

Die Parameter ab Speed sind optional

MotorNr (ftiOut) : Nummer des zu schaltenden M-Ausganges

Direction (ftiDir) : Schaltzustand (ftiLinks, ftiRechts, ftiEin, ftiAus)

Speed (ftiSpeed) : Geschwindigkeitsstufe, Default : ftiFull

Counter (ftiOut) : Begrenzung der Einschaltzeit. Default = 0, unbegrenzt. Werte > 0 geben die Anzahl Impulse an, die der M-Ausgang eingeschaltet sein soll (Fahren eines Motors um n Impulse). Siehe auch "RobMotoren"

Raise 30001, 30002, DoEvents, Counter (bei Parameter Counter)

Siehe auch SetMotors, ClearMotors, SetLamp, Outputs

Beispiel 1

```
ft.SetMotor ftiM1, ftiRight, ftiFull
ft.Pause 1000
ft.SetMotor ftiM1, ftiLeft, ftiHalf
ft.Pause 1000
ft.SetMotor ftiM1, ftiOff
```

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

Beispiel 2

```
ft.SetMotor ftiM1, ftiLeft, 5, 123
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am I-Eingang I2 oder I1 = True mit Geschwindigkeitstufe 5 eingeschaltet. Das Abschalten erfolgt selbsttätig.

SetMotors

Setzen des Status aller M-Ausgänge

ft.**SetMotors** MotorStatus, SpeedStatus, SpeedStatus16, ModeStatus

Die Parameter ab SpeedStatus sind optional

MotorStatus : Schaltzustand der M-Ausgänge

SpeedStatus, SpeedStatus16 (Long) : Geschwindigkeitsstufen der M-Ausgänge. Default : ftiFull

ModeStatus : Betriebsmodus der M-Ausgänge. Default = 0, normal
Bei ModeStatus RobMode sind vorher mit SetCounter die zugehörenden Counterstände zu setzen.

Raise 30001, 30002, DoEvents, Counter (bei Parameter Counter)

Siehe auch ClearMotors, SetMotor, 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 M-Ausgänge abgeschaltet.

WaitForChange

Warten auf eine vorgegebene Anzahl von Impulsen

ft.**WaitForChange** InputNr, NrOfChanges, TermInputNr

Der Parameter TermInputNr ist optional

InputNr (ftiInp) : I-Eingang an dem die Impulse gezählt werden.

NrOfChanges (Long) : Anzahl Impulse

TermInputNr (ftiInp) : Alternatives Ende. I-Eingang = True

Raise 30001, 30002, DoEvent, Abbrechbar, Counter

Intern wird Counter (InputNr) verwendet, der zu Beginn der Methode zurückgesetzt wird

Siehe auch WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh

Beispiel

```
ft.SetMotor ftiM1, ftiLeft
ft.WaitForChange ftiI2, 123, ftiI1
ft.SetMotor ftiM1, ftiOff
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an I-Eingang I2 oder I1 = True gewartet, der Motor wird abgeschaltet. Vergleiche mit dem Beispiel bei SetMotor. Hier wird das Programm angehalten solange der Motor läuft.

WaitForHigh

Warten auf einen False/True-Durchgang an einem I-Eingäng

ft.**WaitForHigh** InputNr

Raise 30001, 30002, DoEvent, Abbrechbar

Siehe auch WaitForLow, WaitForChange, WaitForInput

Beispiel

```
ft.SetMotor ftiM1, ftiOn
ft.SetMotor ftiM2, ftiLeft
ft.WaitForHigh ftiI1
ft.SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 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 I-Eingang den vorgegebenen Wert annimmt.

ft.**WaitForInput** InputNr, OnOff

OnOff : Endebedingung für I-Eingang InputNr, Default = ftiEin

Raise 30001, 30002, DoEvent, Abbrechbar

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
ft.SetMotor ftiM1, ftiLeft
ft.WaitForInput ftiI1
ft.SetMotor ftiM1, ftiOff
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf I-Eingang = ftiEin gewartet, dann wird der Motor wieder abgeschaltet : Anfahren einer Endposition.

WaitForInputIR

Warten daß der angegebene IR-Eingang den vorgegebenen Wert annimmt.

ft.**WaitForInput** Code, KeyNr, OnOff

OnOff : Endebedingung für IR-Eingang KeyNr, Default = ftiEin

Raise 30001, 30002, DoEvent, Abbrechbar

Siehe auch WaitForChange, WaitForLow, WaitForHigh

Beispiel

```
ft.SetMotor ftiM1, ftiLeft
ft.WaitForInputIR ftiCode1, ftiM2L
ft.SetMotor ftiM1, ftiOff
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf IR-Eingang Code1 / M2L = ftiEin gewartet.

WaitForLow

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

ft.**WaitForLow** InputNr

Raise 30001, 30002, DoEvent, Abbbrechbar

Siehe auch WaitForChange, WaitForInput, WaitForHigh

Beispiel

```
ft.SetMotor ftiM1, ftiOn
ft.SetMotor ftiM2, ftiLeft
ft.WaitForLow ftiI1
```

```
ft.SetMotor ftiM2, ftiOff
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 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 (Long) : Zeit in MilliSekunden. Bei Time = 0 wird endlosgewartet.

MotorNr (ftiOut) : Nummern der M-Ausgänge auf die gewartet werden soll. Es wird auf MotorStatus = ftiAus für die angegebenen M-Ausgänge gewartet. MotorNr ftiM1 – ftiM16 in beliebiger Reihenfolge (max 8). Die nicht betroffenen Motoren müssen nicht angegeben werden.
Bei den Return-Werten ftiWait.ftiNotHalt und .ftiESC werden alle betroffen Motoren angehalten.

Raise 30001, 30002, DoEvents, Abbrechbar

Beispiel

```
ft.SetMotor ftiM4, ftiLeft, ftiHalf, 50
ft.SetMotor ftiM3, ftiRight, ftiFull, 40
Do
  lblPos = ft.GetCounter(ftiI6) & " - " & ft.GetCounter(ftiI8)
Loop While ft.WaitForMotors(100, ftiM4, ftiM3) = ftiTime
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehend 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 .... = ftiTime). Wenn die Position erreicht ist <> ftiTime, ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet. Achtung hier wurde nicht auf NotHalt (ftiNotHalt) oder Esc-Taste (ftiEsc) abgefragt, es könnte also auch vor Erreichen der Zielposition abgebrochen worden sein.

WaitForPositionDown

Warten auf Erreichen einer vorgegebenen Position.

ft.**WaitForPositionDown** InputNr, ByRef Counter, Position, TermInputNr

Ausgegangen wird von der aktuellen Position, die in Counter gespeichert ist, es werden solange Impulse von Counter abgezogen, bis der in Position angegebene 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 I-Eingang TermInputNr = True beendet. Counter und Position müssen immer positive Werte (einschl. 0) enthalten.

Raise 30001, 30002, DoEvents, Abbrechbar, Counter

Siehe auch WaitForPositionUp, WaitForChange

Beispiel

```
Dim Zaehler&
  Zaehler = 12
  ft.SetMotor ftiM1, ftiLinks
  ft.WaitForPositionDown ftiI2, Zaehler, 0
  ft.SetMotor ftiM1, ftiAus
```

Die aktuelle Position ist 12 (Zaehler), der Motor an M-Ausgang M1 wird linksdrehend gestartet. WaitForPositionDown wartet dann auf Erreichen der Position 0, der Motor wird dann ausgeschaltet

WaitForPositionUp

Warten auf Erreichen einer vorgegebenen Position.

ft.**WaitForPositionUp** InputNr, ByRef Counter, Position, TermInputNr

Ausgegangen wird von der aktuellen Position in Counter, es werden solange Impulse auf Counter addiert, bis der in Position angegebene 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 I-Eingang TermInputNr = True beendet. Counter und Position müssen immer positive Werte (einschl. 0) enthalten.

Raise 30001, 30002, DoEvents, Abbrechbar, Counter

Siehe auch WaitForPositionDown, WaitForChange

Beispiel

```
Dim Zaehler&
   Zaehler = 0
   ft.SetMotor ftiM1, ftiRechts
   ft.WaitForPositionUp ftiI2, Zaehler, 24
```

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

WaitForTime

Anhalten des Programmablaufs.

ft.**WaitForTime** mSek

Synonym für Pause

Raise 30001, 30002, DoEvents, Abbrechbar

Siehe auch Pause

Beispiel

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

In der Schleife Do Loop Until 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.