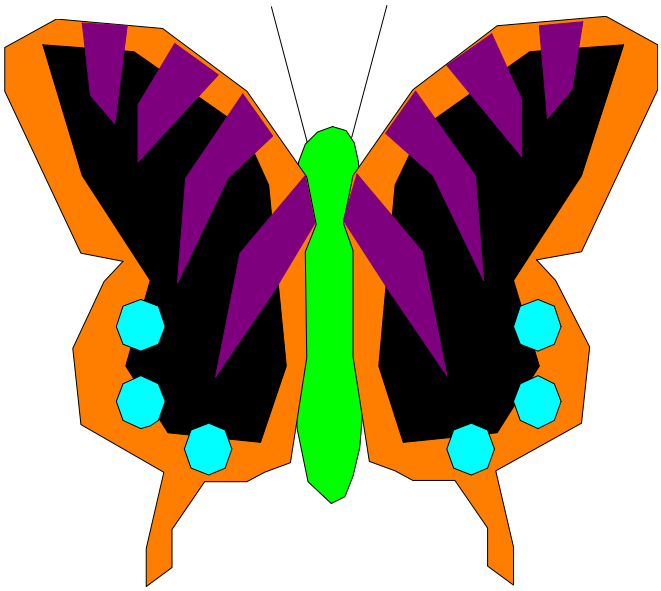
ROBO Pro -> VBA / VB2005 - FishFace2005.DLL

ROBO Starter

Ulrich Müller



Inhaltsverzeichnis

Ulrich Müller, eMail : um@ftcomputing.de Homepage : www.ftcomputing.de

Dokumentname : StarterVB2005.DOC. Druckdatum : 28.07.2006

Übersichten

Allgemeines

Zweck des Dokuments

Hilfe beim Umstieg von der Programmierung in ROBO Pro hin zur Programmierung in VBA oder VB2005. Dazu werden alle ROBO Pro Programme zum ROBO Starter Set (41 863) in einer VBA und einer VB2005 Version vorgestellt. Die Bedeutung der eingesetzten VBA / VB2005 und FishFace Befehle wird jeweils kurz erläutert. Die Erläuterung ersetzt aber kein VBA / VB2005 Lehrbuch, ebenso sollte man für FishFace dessen Referenz zu Rate ziehen.

Interessant dürfte für Programmieranfänger der "Umweg" über VBA zu VB2005 sein, da mit vbaFish40 eine komplette Entwicklungsumgebung (IDE) zur Verfügung steht, die die FishFace Befehle in VBA integriert. Die Zuordnung zu einem Interface und die Fehlerbehandlung geschieht über die IDE. Die IDE selber ist aber deutlich übersichtlicher als die von VB2005, enthält aber allen Komfort – einschl. einer ausführlichen Hilfedatei - , den man für die Entwicklung von Programmen für fischertechnik Modelle benötigt. Da die verwendete Sprachmenge weitgehend gleich ist, ist ein späterer Umstieg auf VB2005 relativ einfach möglich.

Die Beispielprogramme des Dokuments bauen aufeinander auf, sie sollten deshalb nacheinander durchgearbeitet werden.

Installation und Literatur

- Der fischertechnik Computing Kasten ROBO Starter Set (41 863) mit einem installierten ROBO Pro. Der alte Computing Starter Kasten (16 553) mit dem Intelligent Interface kann ebenso genutzt werden. In diesem Fall muß aber ROBO Pro zusätzlich installiert werden.
- Bei Nutzung von VBA : Ein installiertes vbaFish40 ->
  www.ftcomputing.de/zip/vbafish40setup.exe
  zusätzlich www.ftcomputing.de/zip/startervb2005.zip Die Weiteren Schritte können dann – vorerst – entfallen. Zum Päckchen gehören eine ausführliche Dokumentation, die über Menü Hilfe zu erreichen ist, sowie die Möglichkeit über F1 Sofort-Hilfe für einzelne Funktionen von VBA / FishFace zu bekommen. (kostenlos)
- Die VB2005 Express Edition oder höher. Microsoft stellt die Express Edition kostenlos zur Verfügung.
- Ein Einsteigerbuch zu VB2005 :
  Peter Bloch : Einstig in Visual Basic 2005, Galileo Computing
  ISBN : 3-89842-641-6, Preis : 24,90€. Enthält auch eine CD mit VB2005 Express
- FishFace2005.DLL und das Handbuch dazu.
  www.ftcomputing.de/zip/vb2005fish40setup.exe (kostenlos)
  Die "gelben Zettel" im Editor und der Object Browser werden ausführlich unterstützt.
  Achtung : Die Datei FishFace2005.XML muß mit FishFace2005.DLL im gleichen Verzeichnis sein.

- Dieses Dokument und die Beispiele dazu :
  www.ftcomputing.de/zip/startervb2005.zip
  Die Beispiele in ein eigenes Verzeichnis entpacken. Sollte bei Öffnen eines Beispieles ein defekter Verweis auf FishFace2005.DLL (erkenntlich an einer Unzahl an Fehlermeldungen) moniert werden, so ist der über
  Menü | Projekt | Verweis hinzufügen | Durchsuchen
  zu reparieren.
- Bei weitergehendem Interesse für VB2005 :
  Michael Kofler : Visual Basic 2005, Addison-Wesley,
  ISBN : 3-8273-2338-X, Preis : 59,95€, kein VB2005 Express CD

Anschluß des Interfaces

Bei Einsatz von vbaFish40 ist über Menü | Extras | Interface Daten das genutzte Interface einzustellen.

Bei allen VB2005 Beispielprogrammen wird das erste ROBO Gerät an USB (ROBO Interface, ROBO I/O Extension) genutzt :

```
ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
```

Ein Betrieb mit dem Intelligent Interface (oder dem ROBO Interface) an COM ist genauso möglich. Dazu muß das OpenInterface modifiziert werden :

```
ft.OpenInterface(IFTypen.ftIntelligent_IF, Port.COM1, 5)
```

bzw.:

```
ft.OpenInterface(IFTypen.ftROBO_IF_COM, Port.COM1, 5)
```

Port.COM1 ist ggf. anzupassen.

Betrieb der Programme

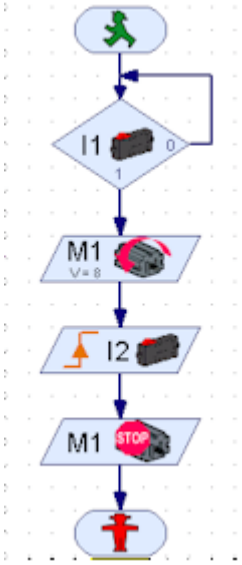
- VBA : Nur innerhalb der IDE über die entsprechenden Buttons.
- Start über F5 aus der VB2005 Entwicklungsumgebung
- Abbrechen über :
  ESC-Taste oder, wenn das nicht hilft mit
  Shift F5 aus der VB2005 Entwicklungsumgebung. Achtung : Die Titelzeile der Entwicklungsumgebung muß aktiv (blau) sein.
- Beenden :
  Konsol Anwendungen über RETURN
  Windows Anwendungen über entsprechenden Button.

Programme

Motorsteuerung

ROBO Pro

Motorsteuerung1



Warten auf Taster an I1

Motor an M1 links, voll

Warten auf aus/ein an Taster an I2

Motor an M1 aus

VBA

Motorsteuerung1

```
Sub Main
  Do
  Loop Until GetInput(ftiI1)
  SetMotor(ftiM1,ftiLinks)
  WaitForHigh(ftiI2)
   SetMotor(ftiM1,ftiAus)
End Sub
```

Das ist alles, der Rest wird durch die Entwicklungsumgebung vbaFish40 erledigt.

Motorsteuerung2

```
Sub Main
  PrintStatus "Motorsteuerung gestartet"
  WaitForInput(ftiI1)
  SetMotor(ftiM1,ftiLinks)
  WaitForHigh(ftiI2)
```

```
   SetMotor(ftiM1,ftiAus)
   Pause(10000)
   SetMotor(ftiM1,ftiRechts)
   WaitForHigh(ftiI3)
   SetMotor(ftiM1,ftiAus)
End Sub
```

Ist auch nicht viel mehr.

VB2005

Motorsteuerung1

```
Imports FishFace40
Module Motorsteuerung1
  Private ft As New FishFace()
  Sub Main()
    ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
    Do : Loop Until ft.GetInput(Inp.I1)
    ft.SetMotor(Out.M1, Dir.Links)
    ft.WaitForHigh(Inp.I2)
    ft.SetMotor(Out.M1, Dir.Aus)
    ft.CloseInterface()
  End Sub
End Module
```

Konsolprojekt anlegen

Anlegen : Menü | Datei | Neues Projekt | Konsolanwendung

Menü | Verweis hinzufügen | Durchsuchen (Aktuell) | C:\Programme\ftComputing\FishFace2005.DLL

Erste Zeile : `Imports FishFace40`

Erste Zeile hinter Module : `Private ft As New FishFace()`

Das Programm wird mit Sub Main() gestartet (Start F5-Taste).

Die Verbindung zu Interface wird über ft.OpenInterface(..) hergestellt und über ft.CloseInterface() beendet. Bei ROBO Pro Programmen wird das durch die ROBO Pro Umgebung erledigt.

Die weiteren Statements entsprechen den ROBO Pro Symbolen.

Da VB2005 im Gegensatz zu ROBO Pro kein "GoTo" kennt ist hier für das Warten auf I1 = True eine Schleifen-Konstruktion fällig, genaueres siehe später.

Das Programm läuft im Konsolfenster und hat keine Ausgaben.

Motorsteuerung2

```
Imports FishFace40
Module Motorsteuerung2
  Private ft As New FishFace()
  Sub Main()
    Try
      ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
      Console.WriteLine("Motorsteuerung gestartet")
      ft.WaitForInput(Inp.I1)
      ft.SetMotor(Out.M1, Dir.Links)
      ft.WaitForHigh(Inp.I2)
      ft.SetMotor(Out.M1, Dir.Aus)
      ft.Pause(10000)
      ft.SetMotor(Out.M1, Dir.Rechts)
```

```
      ft.WaitForHigh(Inp.I3)
      ft.SetMotor(Out.M1, Dir.Aus)
      ft.CloseInterface()
      Console.WriteLine("Motorsteuerung beendet")
    Catch eft As FishFaceException
      Console.WriteLine(eft.Message)
    End Try
    Console.ReadLine()
  End Sub
End Module
```

Das Programm entspricht funktional der ROBO Pro Motorsteuerung2, es wurde durch Funktionen ergänzt, die bei ROBO Pro dessen Umgebung übernimmt. Außerdem wurde das `Do : Loop` Warten durch `ft.WaitForInput(Inp.I1)` ersetzt.

`Try .. Catch .. End Try`

Eine Klammerung von VB2005 Statements mit der vorgeschrieben wird, was im Ausnahmefall zu tun ist. Der Ausnahmefall gilt für alle Statements zwischen Try .. Catch, nach Catch wird dann die Ausnahme behandelt. Hier : Ausgabe einer entsprechenden Anzeige auf der Konsole. Als Ausnahme kommen in erster Linie nicht angeschlossenenes oder falsches Interface infrage.

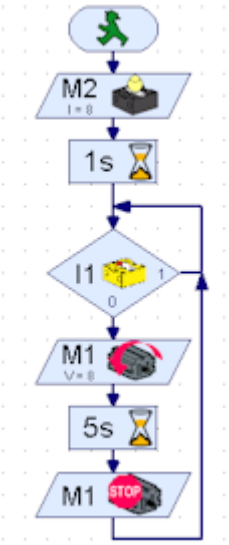
`Console.xxx`

Über Consol.WriteLine("...") können Ausgaben auf der Konsole gemacht werden, hier wird das genutzt um den Status des Programms anzuzeigen.
Mit Console.ReadLine(...) können ganze Zeilen (Text + RETURN) von der Tastatur eingelesen werden. Hier wird das allerdings nur zum Warten auf die RETURN-Taste genutzt – solange bleibt das Konsolfenster sichtbar.

Händetrockner

ROBO Pro

Händetrockner1



Lampe für Lichtschranke ein, warten dass Lichtschranke ansprechen kann.

Warten auf Unterbrechung der Lichtschranke

Motor an M1 Links-Ein

Für 5 Sekunden

Motor Aus

VBA

Haendetrockner1

```
Sub Main
  PrintStatus "Händetrockner gestartet"
  SetMotor(ftiM2,ftiEin)
  Pause(1000)
  Do
    If Not GetInput(ftiI1) Then
      SetMotor(ftiM1,ftiLinks)
      Pause(5000)
      SetMotor(ftiM1,ftiAus)
    End If
  Loop Until Finish()
   PrintStatus "Händetrockner beendet"
End Sub
```

Anstelle der Verzweigungen eine Do .. Schleife. PrintStatus macht eine Druckausgabe in die Statuszeile von vbaFish.

Haendetrockner2

```
Sub Main
  PrintStatus "Händetrockner gestartet"
  SetMotor(ftiM2,ftiEin)
  Pause(1000)
  Do
    If GetInput(ftiI1) Then
      SetMotor(ftiM1,ftiAus)
    Else
      SetMotor(ftiM1,ftiLinks)
    End If
  Loop Until Finish()
   PrintStatus "Händetrockner beendet"
End Sub
```

IF mit Then / Else- Zweigen zu Ein- und Ausschalten des Motors

VB2005

Haendetrockner1

```
Imports FishFace40
Module Haendetrockner1
  Private ft As New FishFace()
  Sub Main()
    Try
      ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
      Console.WriteLine("Händetrockner gestartet")
      ft.SetMotor(Out.M2, Dir.Ein)
      ft.Pause(1000)
      Do
        If Not ft.GetInput(Inp.I1) Then
          ft.SetMotor(Out.M1, Dir.Links)
          ft.Pause(5000)
          ft.SetMotor(Out.M1, Dir.Aus)
        End If
      Loop Until ft.Finish()
      ft.CloseInterface()
      Console.WriteLine("Händetrockner beendet")
```

```
    Catch eft As FishFaceException
      Console.WriteLine(eft.Message)
    End Try
  End Sub
End Module
```

Der Rahmen wie gehabt. Bei `ft.SetMotor(Out.M2 ...` geht's dann los.

Danach eine "Endlos"-Schleife Do ... Loop, die durch ft.Finish() = True beendet wird. Finish wird bei Drücken der ESC-Taste oder aber auch eines Tasters, der dann extra angegeben werden müßte, True. Sonst passiert da eigentlich nicht Aufregendes.

Haendetrockner2

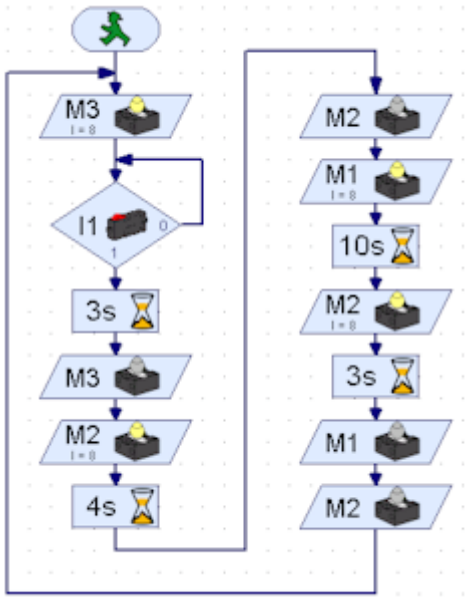
```
Do
  If ft.GetInput(Inp.I1) Then
    ft.SetMotor(Out.M1, Dir.Aus)
  Else
    ft.SetMotor(Out.M1, Dir.Links)
  End If
Loop Until ft.Finish()
```

Nur im Do .. Loop hat sich etwas getan : bei Unterbrechung der Lichtschranke wird nicht mehr stur 5 Sekunden Wind gemacht. Der Motor wird jetzt bei Lichtschranke True abgeschaltet. Das Not wurde weggelassen, deswegen wurde die Reihenfolge der SetMotor geändert.

Ampel

ROBO Pro

Ampel1



Grüne Lampe an M3 Ein
Warten auf Fußgängerwunsch I1 = True
Wunsch erkannt, 3 Sekunden warten lassen.
4 Sek. M3 grün aus, M2 gelb Ein

10 Sek. M2 Aus, M1 rot Ein

3 Sek M2 gelb Ein

M1, M2 Aus und das ganze von vorn.

Ampel2 habe ich mir geschenkt, bei Dauerdrücken von I2 wird gelb geblinkt.
Unterprogramme kommen später.
Ampel3 ist die Level3 Version von Ampel1, gibt also das gleiche VB2005 Programm.

VBA

Ampel1

```
Sub Main
  PrintStatus "Ampel gestartet"
  Do
    SetMotor(ftiM3,ftiEin)
    WaitForInput(ftiI1)
    Pause(3000)
    SetMotor(ftiM3,ftiAus)
    SetMotor(ftiM2,ftiEin)
    Pause(4000)
    SetMotor(ftiM2,ftiAus)
    SetMotor(ftiM1,ftiEin)
    Pause(10000)
    SetMotor(ftiM2,ftiEin)
    Pause(3000)
    SetMotor(ftiM3,ftiAus)
    SetMotor(ftiM2,ftiAus)
  Loop Until Finish()
  PrintStatus "Ampel beendet"
End Sub
```

Wieder der gleiche Dreh

VB2005

Ampel1

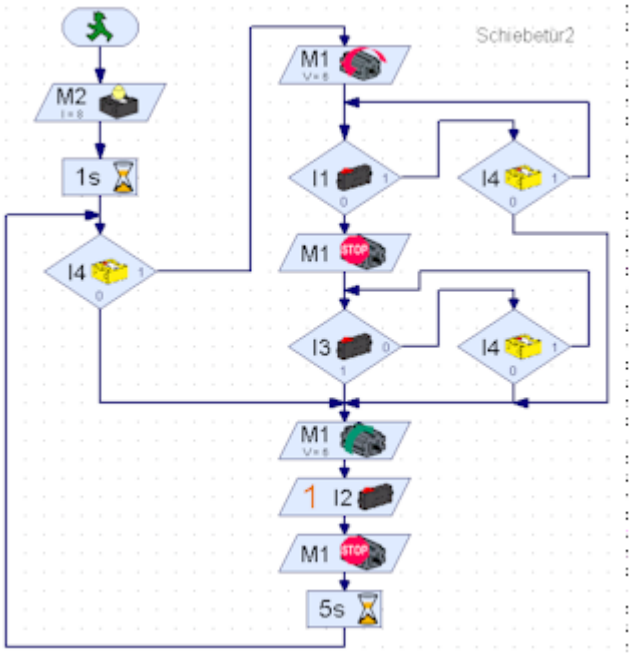
```
Do
  ft.SetMotor(Out.M3, Dir.Ein)
  ft.WaitForInput(Inp.I1)
  ft.Pause(3000)
  ft.SetMotor(Out.M3, Dir.Aus)
  ft.SetMotor(Out.M2, Dir.Ein)
  ft.Pause(4000)
  ft.SetMotor(Out.M2, Dir.Aus)
  ft.SetMotor(Out.M1, Dir.Ein)
  ft.Pause(10000)
  ft.SetMotor(Out.M2, Dir.Ein)
  ft.Pause(3000)
  ft.SetMotor(Out.M1, Dir.Aus)
  ft.SetMotor(Out.M2, Dir.Aus)
Loop Until ft.Finish()
```

Der Do .. Loop sieht eigentlich genauso aus wie das ROBO Pro Programm.

Schiebetür

ROBO Pro

Schiebetür2



Bei Start Tür schließen, wenn etwas dazwischenkommt (Lichtschranke) für 5 Sek. öffnen und dann wieder schließen.

Normalbetrieb : Taster I3 = True

Tür öffnen und nach 5 Sek. wieder schließen. Wenn etwas dazwischen kommt : siehe oben.

Die Angelegenheit ist etwas unübersichtlich, da zusätzlich immernoch die Lichtschranke kontrolliert werden muß.

Schiebetür1 habe ich mir geschenkt, da geht alles ganz einfach und schön der Reihe nach. Bei den Beispielen ist es aber dabei : Für Geniesser.

VBA

Schiebetuer2

```
Const mTuer              = ftiM1
Const mLampe             = ftiM2
Const iTuerNGeschlossen  = ftiI1
Const iTuerOffen         = ftiI2
Const iTuerOeffnen       = ftiI3
Const iLichtSchranke     = ftiI4
Const cTuerOeffnen       = ftiRechts
Const cTuerSchliessen    = ftiLinks
Const cTuerHalt          = ftiAus
Const cLichtSBroken      = False

Sub TuerSchliessen()
   SetMotor(mTuer,cTuerSchliessen)
   Do
     If GetInput(iLichtSchranke)= cLichtSBroken Then Exit Do
   Loop Until GetInput(iTuerNGeschlossen)=False
   SetMotor(mTuer,cTuerHalt)
End Sub

Sub TuerOeffnen()
   SetMotor(mTuer,cTuerOeffnen)
   WaitForInput(iTuerOffen)
   SetMotor(mTuer,cTuerHalt)
```

```
   Pause(5000)
End Sub

Sub Main
   PrintStatus "Schiebetür gestartet"
   SetMotor(mLampe,ftiEin)
   Pause(1000)
   TuerSchliessen
   Do
     If GetInput(iTuerOeffnen) Or _
        GetInput(iLichtSchranke)=cLichtSBroken Then
       TuerOeffnen
       TuerSchliessen
     End If
   Loop Until Finish()
   PrintStatus "Schiebetür beendet"
End Sub
```

Symbolische Konstanten zur besseren Kennzeichnung der beteiligten Komponenten.

Auslagerung von Funktionen aus Sub Main() in neue Sub's (TuerOeffnen / TuerSchliessen)

Gleichzeite Abfrage von mehreren Zuständen (iLichtSchranke / iTuerOeffnen und iLichtSchranke / iTuerNGeschlossen)

VB2005

Schiebetuer2

```
Sub Main()
  Try
    ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
    Console.WriteLine("Schiebetür gestartet")
    ft.SetMotor(mLampe, Dir.Ein)
    ft.Pause(1000)
    TuerSchliessen()
    Do
      If ft.GetInput(iTuerOeffnen) Or _
         ft.GetInput(iLichtSchranke) = cLichtSBroken Then
        TuerOeffnen()
        TuerSchliessen()
      End If
    Loop Until ft.Finish()
    ft.CloseInterface() .....
End Sub
```

Da sich die ROBO Pro Schleifen mit VB2005 nur schwer nachbilden lassen und eh unübersichtlich sind, wird hier sequentieller und mit nur einer Schleife gearbeitet. Außerdem kommen die bei Ampel2 unterschlagenen Sub's hier zum Einsatz.

Start schön der Reihe nach endet mit Aufruf der Sub TuerSchliessen(). Wenn der Sub etwas dazwischen (die Lichtschranke) kommt, wird lediglich die Tür angehalten.

Anschließend kommt die normale Betriebsschleife, die mit einer Abfrage auf iTuerOeffnen-Taster oder iLichschranke startet, hängt die Tür noch vom Start, wird sie hier für 5 Sek. ganz geöffnet. Genauso wird sie geöffnet, wenn der iTuerOeffnen-Taster = True ist. Anschließend : TuerOeffnen – TuerSchliessen, Beenden der Schleife durch ESC-Taste.

```
Private Sub TuerSchliessen()
  ft.SetMotor(mTuer, cTuerSchliessen)
  Do
    If ft.GetInput(iLichtSchranke) = cLichtSBroken Then Exit Do
  Loop Until ft.GetInput(iTuerGeschlossen) = False
  ft.SetMotor(mTuer, cTuerHalt)
```

```
End Sub
```

Die Sub TuerSchliessen startet den Türmotor mTuer in Richtung cTuerSchliessen und wartet dann in einem Do .. Loop auf iTuerNGeschlossen = False (Bei dem Modell ist die Tür geschlossen, wenn Taster I1 geöffnet ist. In der Schleife wird noch die Lichtschranke abgefragt, wenn das der Fall ist, wird die Schleife verlassen. In beiden Fällen wird vor Ende der Sub der mTuer abgestellt.

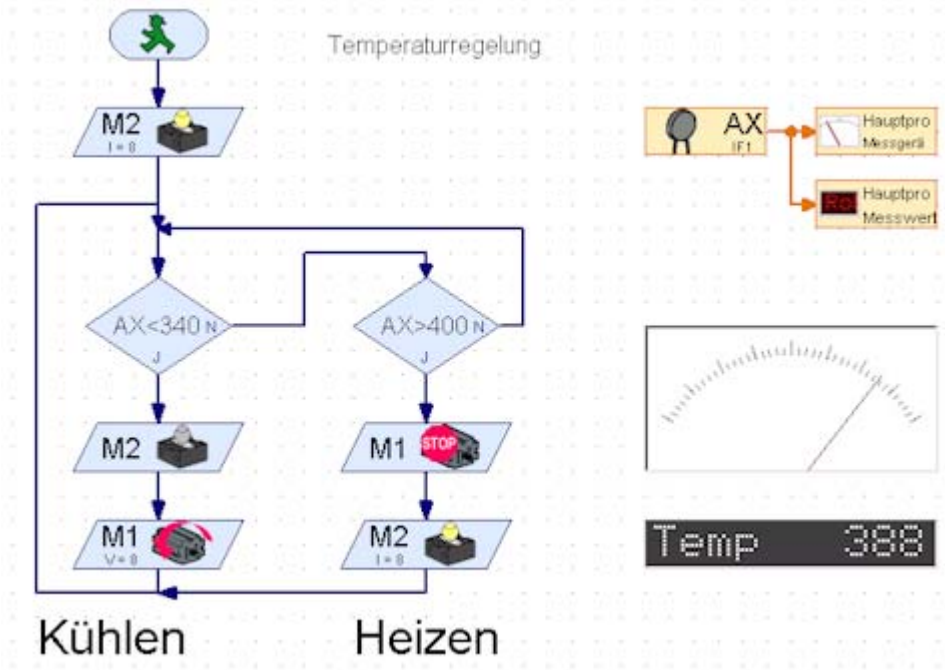
```
Const mTuer As Out = Out.M1
Const mLampe As Out = Out.M2
Const iTuerNGeschlossen As Inp = Inp.I1
Const iTuerOffen As Inp = Inp.I2
Const iTuerOeffnen As Inp = Inp.I3
Const iLichtSchranke As Inp = Inp.I4
Const cTuerOeffnen As Dir = Dir.Rechts
Const cTuerSchliessen As Dir = Dir.Links
Const cTuerHalt As Dir = Dir.Aus
Const cLichtSBroken As Boolean = False
```

Um nicht ständig mit den Nummern der Ein- und Ausgänge arbeiten zu müssen, wurden ihnen symbolische Namen verpaßt, die etwas über ihre Bedeutung im Programm aussagen. Das Programm wird dadurch lesbarer.

Temperaturregelung

ROBO Pro

Temperaturregelung



Eingesetzt wird ein NTC (Negative Temperature Coefficient), der die Temperatur "verkehrtrum" anzeigt (hohe Temperatur – kleiner Wert (zwischen 0 und 1023))

Geregelt wird ständigt : ist es zu heiß (AX<323) wird gekühlt (bis AX>333), ist es zu kalt (AX>333) wird geheizt (bis AX<323). Einen Normalbereich (zwischen 323 und 333) in dem nichts passiert, gibt es hier nicht. Vorteil : immer Aktion, man sollte aber die Grenzwerte nach eigenem Geschmack und den Außentemperaturen anpassen.

Neu ist hier auch die "Temperatur"-Anzeige über Analog-Instument und Text.

VBA

Temperaturregelung

```
Sub Main
  Dim AX As Integer
  Dim Status As String
   PrintStatus "Temperaturregelung gestartet"
   SetMotor(ftiM2,ftiEin)
   Status = "Start : "
   Do
     AX = GetAnalog(ftiAX)
     PrintStatus Status & AX
     If AX < 340 Then
       Status = "Kühlen : "
       SetMotor(ftiM1,ftiLinks)
       SetMotor(ftiM2,ftiAus)
     ElseIf AX > 350 Then
       Status = "Heizen : "
       SetMotor(ftiM1,ftiAus)
       SetMotor(ftiM2,ftiEin)
     End If
     Pause(100)
  Loop Until Finish()
End Sub
```

Anzeige des Status (Heizen / Kühlen) und der aktuellen Temperatur in der Statuszeile.

Einsatz eines Ifs mit Then und ElseIf- Zweig zum Schalten von Heizen / Kühlen. Bei den Beispielen findet sich noch ein Temperaturregelung2 mit einem interessanten

```
Select Case AX
  Case Is < 340
    Status = "Kühlen : "    ...
  Case Is > 350
    Status = "Heizen : "    ...
  Case Else
    Status = "Normal : "    ...
End Select
```

Select Case mit dem sehr übersichtliche Fallunterscheidungen getroffen werden können. Hier gibt es noch einen Fall "Normal" in dem gar nichts passiert. Das kann beim Testen irritieren, wenn sich das Programm (bei entsprechenden Grenzwerten und Temperaturen) so gar nichts rührt. Die erste Variante zeigt da deutlich mehr Aktion.

VB2005

Die Temperaturanzeige bietet willkommenen Anlaß, von den schlichten Konsolprogrammen auf solche mit Windows.Forms umzusteigen. Auch wenn die Temperaturanzeige eher schlicht geraten ist.

```
Imports FishFace40
Public Class TemperaturRegelung
  Private ft As New FishFace()
  Private Sub cmdAction_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles cmdAction.Click
    Dim AX As Integer
    Try
      ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
      lblStatus.Text = "Temperaturregelung gestartet"
      ft.SetMotor(Out.M2, Dir.Ein)
      Do
        AX = ft.GetAnalog(Inp.AX)
        lblTemp.Text = AX
        If AX < 340 Then
          lblStatus.Text = "Kühlen"
          ft.SetMotor(Out.M1, Dir.Links)
          ft.SetMotor(Out.M2, Dir.Aus)
        ElseIf AX > 380 Then
          lblStatus.Text = "Heizen"
          ft.SetMotor(Out.M1, Dir.Aus)
          ft.SetMotor(Out.M2, Dir.Ein)
        End If
        ft.Pause(100)
      Loop Until ft.Finish()
      ft.CloseInterface()
      lblStatus.Text = "Temperaturregelung beendet"
    Catch eft As FishFaceException
      lblStatus.Text = eft.Message
    End Try
  End Sub
End Class
```

Anlegen des Projekts wie bei den Konsolprogrammen, nur eben als Windows Anwendung. Die umgebende Einheit heißt dann hier nicht mehr Module sondern Class, in Praxis ist das erstmal kein Unterschied. Die Anwendung wird jetzt in der Event-Routine zum Button-Click für den Button ACTION (cmdAction) untergebracht. Der Rahmen der Routine wird durch DoppelKlick auf den Button(Entwurf) erzeugt. Hinzu kommen noch die Labels lblStatus und lblTemp. Das ist erstmal alles. Hinweis : hier wurde einfachheitshalber auf die Const-Namen verzichtet.

Die Temperaturabfragen wurden wieder in einem Do .. Loop untergebracht. Dort wird zunächst die aktuelle Temperatur festgestellt und in lblTemp.Text angezeigt und dann, je nach Gegebenheit, nach Kühlen bzw. Heizen verzweigt. Dort wird auch der zugehörende Text für lblStatus.Text ausgegeben.

Temperaturregelung2

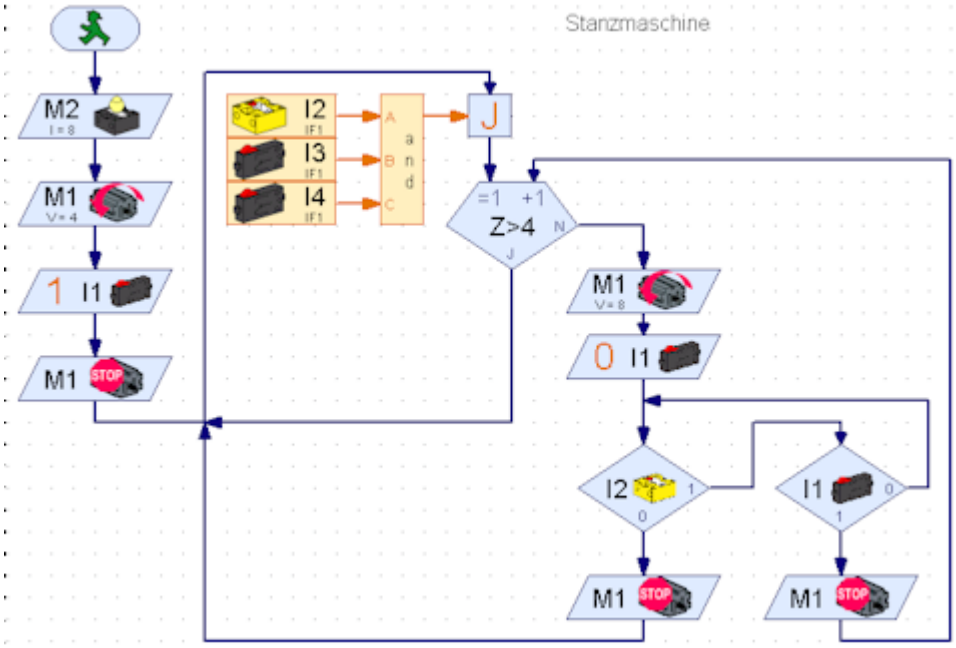
```
ft.SetMotor(Out.M2, Dir.Ein)
  Do
    AX = ft.GetAnalog(Inp.AX)
    lblTemp.Text = AX
    Select Case AX
      Case Is < 340
        lblStatus.Text = "Kühlen"
        ft.SetMotor(Out.M1, Dir.Links)
        ft.SetMotor(Out.M2, Dir.Aus)
      Case Is > 380
        lblStatus.Text = "Heizen"
        ft.SetMotor(Out.M1, Dir.Aus)
        ft.SetMotor(Out.M2, Dir.Ein)
      Case Else
        lblStatus.Text = "In der Toleranz"
        ft.SetMotor(Out.M1, Dir.Aus)
        ft.SetMotor(Out.M2, Dir.Aus)
    End Select
    ft.Pause(100)
  Loop Until ft.Finish()
```

Da die oben gezeigte Temperaturregelung zwar viel Aktion zeigt, aber unnötig viel Aktion zeigt, hier eine Lösung mit Hysterese (nach Duden : Fortdauer einer Wirkung nach Aufhören der Ursache). Es wird ein dritter Zweig hinzugefügt in dem beide M-Ausgänge abgeschaltet werden. Diesmal eingebettet in ein Select Case Konstrukt (übersichtlicher bei mehreren Fällen). Vorteil : Läuft nur noch wenn nötig. Nachteil : Es passiert manchmal (stundenlang) gar nichts.

Stanzmaschine

ROBO Pro

Stanzmaschine1



Bei gleichzeitiger Betätigung von I3 und I4 sowie "heiler" Lichtschranke wird die Stanze für 4 Hübe gestartet. Die Hübe werden durch I1 in der Stellung Stanze oben gezählt. Bei einer Unterbrechung der Lichtschranke wird die Stanze angehalten.

VBA

Stanzmaschine1

```
Sub Main
  Dim i As Integer
  SetMotor(mLichtS,ftiEin)
  SetMotor(mStanze, ftiLinks)
  WaitForInput(iStanzeOben)
  SetMotor(mStanze,ftiAus)
  Do
    PrintStatus "Stanze bereit"
    If GetInput(iLichtOK) And _
       GetInput(iHandLinks) And _
       GetInput(iHandRechts) Then
      For i = 1 To 4
        PrintStatus "Hub : " & i
        SetMotor(mStanze, ftiLinks)
        WaitForInput(iStanzeOben,False)
        Do While Not GetInput(iStanzeOben) And GetInput(iLichtOK)
        Loop
        SetMotor(mStanze,ftiAus)
        If Not GetInput(iLichtOK) Then Exit For
      Next
    End If
  Loop Until Finish()
```

```
End Sub
```

Auffallen tut hier das gewaltige If in dem auf "heile" Lichtschranke und die Taster iHandLinks und iHandRechts (Zweihandeinrückung) geprüft wird, bevor die Stanze gestartet wird. Während des Stanzens wird weiter auf iLichtOK abgefragt. ggf. wird "fluchtartig" (Exit For) der For .. Next Loop verlassen.

Stanzmaschine2

```
Sub Stanzen()
  Dim i As Integer
  For i = 1 To EA()
    PrintStatus "Hub : " & i
    SetMotor(mStanze,ftiLinks)
    WaitForInput(iStanzeOben,False)
    Do While Not GetInput(iStanzeOben) And GetInput(iLichtOK)
    Loop
    SetMotor(mStanze, ftiAus)
    If Not GetInput(iLichtOK) Then Exit For
  Next
  anzTeile = anzTeile + 1
  Printstatus "Anzahl Teile : " & anzTeile
End Sub
```

Hier wurde die eigentliche Stanzfunktion in die Sub Stanzen() verlagert. In dem Do .. Loop der Sub Main() also nur noch ein Aufruf Stanzen.

Hinzugekommen ist eine variable Hubzahl pro Werkstück. Die aktuell gültige Hubzahl wird über die Funktion EA() dem Feld EA entnommen. Außerdem wird laufend die Anzahl der bearbeitenden Teile angezeigt (auch der unvollständig bearbeiteten)

VB2005

Stanzmaschine1

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles cmdAction.Click
  Try
    ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
    cmdEnde.Text = "HALT"
    cmdAction.Enabled = False
    ft.SetMotor(Out.M2, Dir.Ein)
    ft.SetMotor(mStanze, Dir.Links)
    ft.WaitForInput(iStanzeOben)
    ft.SetMotor(mStanze, Dir.Aus)
    Do
      lblStatus.Text = "Stanzmaschine bereit"
      If ft.GetInput(iLichtOK) And _
         ft.GetInput(iHandLinks) And _
         ft.GetInput(iHandRechts) Then
        For i As Integer = 1 To 4
          lblStatus.Text = "Hub : " & i
          ft.SetMotor(mStanze, Dir.Links)
          ft.WaitForInput(iStanzeOben, False)
          Do While Not ft.GetInput(iStanzeOben) And _
                       ft.GetInput(iLichtOK)
          Loop
          ft.SetMotor(mStanze, Dir.Aus)
          If Not ft.GetInput(iLichtOK) Then Exit For
        Next
      End If
    Loop Until ft.Finish()
    lblStatus.Text = "Stanzmaschine beendet"
```

```
    cmdEnde.Text = "ENDE"
    cmdAction.Enabled = True
    ft.CloseInterface()
  Catch eft As FishFaceException
    lblStatus.Text = eft.Message
  End Try
End Sub

Private Sub cmdEnde_Click(ByVal sender As System.Object, _
                  ByVal e As System.EventArgs) Handles cmdEnde.Click
  If cmdEnde.Text = "HALT" Then ft.NotHalt = True Else Me.Close()
End Sub

Private Sub StanzMaschine1_FormClosing(_
        ByVal sender As System.Object, _
        ByVal e As System.Windows.Forms.FormClosingEventArgs) _
        Handles MyBase.FormClosing
  If cmdEnde.Text = "HALT" Then e.Cancel = True
End Sub
```

Vielleicht ist schon bei der Temperaturregelung aufgefallen, daß es gar nicht so einfach ist, ein einmal laufendes Programm zu beenden, Do .. Loop Schleifen laufen auch noch nach einem FormClose weiter.

Deswegen hier ein zusätzlicher Button cmdEnde, der im laufenden Betrieb einen Endewunsch (ft.NotHalt = True) signalisiert, daraufhin wird über ft.Finish() = True eine laufende Schleife beendet, zusätzlich werden Wait.. Befehle in einer Schleife abgebrochen.

Dazu gehören noch einige "Verzierungen" :
- eine laufen cmdAction sollte nicht nocheinmal aufgerufen werden können - cmdAction.Enabled = False.
– cmdEnde hat einen "Mehrfachnutzen", nach Start des Programms ist er mit Abbrechen beschriftet, Abbruch des Programms, bei laufender cmdAction Beschriftung mit HALT und nach erfolgreichem beenden von cmdAction mit ENDE wieder zum Beenden des Programms.
– Das Windows x (rechts oben) sollte während einer laufenden cmdAction außer Betrieb sein – Event FormClosing : Bei cmdEnde.Text = "HALT" kein Close.

Zum produktiven Teil des Programms :

Do .. Loop Schleife um den Betrieb aufrecht zu halten, wie gehabt.

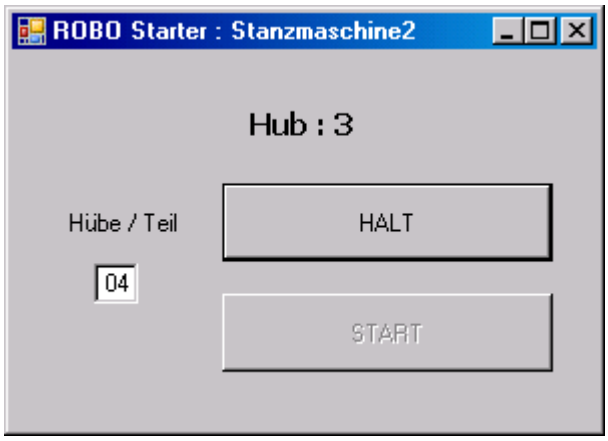
Darin als erstes eine gewaltige Abfrage : Lichtschranke klar, Zweihandeinrückung OK (I3 und I4 = True).

Wenn das zutrifft, wird ein For .. Next Loop gestartet, der vier Stanzzyklen ausführt, wenn er nicht durch Störung an der Lichtschranke vorzeitig abgebrochen wird (Exit For).

Bevor es im Loop so richtig losgeht, wird erstmal der iStanzeOben Taster "freigefahren" damit die nachfolgende Abfrage auf iStanzeOben auch ein Ergebnis liefert.

Das wars denn auch schon, mehr drumrum als drin. Zum DrumRum zählen hier auch wieder die symbolischen Konstanten.

Stanzmaschine2



Hier wird mit variabler Anzahl Hübe / Teil gearbeitet, der aktuelle Wert wird dem Feld mskHuebe entnommen (auf einen Schieberegler wurde verzichtet, faulheitshalber). Der aktuelle Hub wird weiterhin im Feld lblStatus angezeigt.

```
lblStatus.Text = "Stanzmaschine bereit"
  Do
    If ft.GetInput(iLichtOK) And _
       ft.GetInput(iHandLinks) And _
       ft.GetInput(iHandRechts) Then
      Stanzen()
    End If
  Loop Until ft.Finish()
lblStatus.Text = "Stanzmaschine beendet"
```

Das Programm selber hat sich kaum verändert. Lediglich die Stanzfunktionen wurden in eine Sub Stanzen() verlagert :

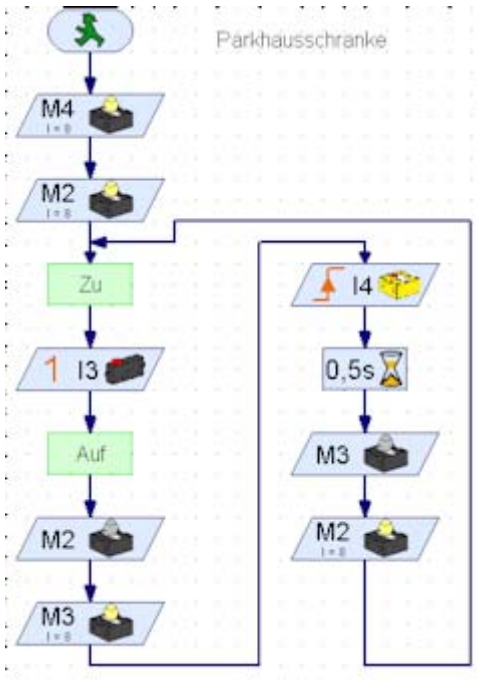
```
Private Sub Stanzen()
  For i As Integer = 1 To mskHuebe.Text
    lblStatus.Text = "Hub : " & i
    ft.SetMotor(mStanze, Dir.Links)
    ft.WaitForInput(iStanzeOben, False)
    Do While Not ft.GetInput(iStanzeOben) And ft.GetInput(iLichtOK)
    Loop
    ft.SetMotor(mStanze, Dir.Aus)
    If Not ft.GetInput(iLichtOK) Then Exit For
  Next
  anzTeile += 1
  lblStatus.Text = "Anzahl Teile : " & anzTeile
End Sub
```

Der For .. Next Loop geht jetz nicht mehr fest bis 4 sondern bis zum Inhalt des Feldes mskHuebe. Am Ende des For .. Next wird jetzt zusätzlich die Anzahl verarbeiteter Teile angezeigt. Im Gegensatz zur ROBO Pro Lösung werden hier auch die "abgebrochenen" Teile mitgezählt (Weils so bequemer ist).

Parkhausschranke

ROBO Pro

Parkhausschranke1



Lichtschranke aktivieren

Ampel auf rot

Schranke schließen

Warten auf Anforderung zum Öffnen

Schranke öffnen

Ampel auf grün

Warten auf Durchfahrt

Ampel auf rot

endlose Schleife ab Schranke schließen

VBA

Parkhausschranke1

```
Sub Oeffnen()
   If GetInput(iSchrankeOffen) Then Exit Sub
   PrintStatus "Schranke öffnet"
   SetMotor(mSchranke,cSchrankeOeffnen)
   WaitForInput(iSchrankeOffen)
   SetMotor(mSchranke, ftiAus)
  PrintStatus "Schranke geöffnet"
End Sub
.....
Sub Main
  SetMotor(mLichtschranke,ftiEin)
  SetMotor(mRot,ftiEin)
  PrintStatus "Schranke bereit"
  Do
    Schliessen
    WaitForInput(iSchrankeOeffnen)
    Oeffnen
    SetMotor(mRot,ftiAus)
    SetMotor(mGruen,ftiEin)
    WaitForHigh(iLichtSchranke)
    Pause(500)
    SetMotor(mGruen,ftiAus)
    SetMotor(mRot,ftiEin)
  Loop Until Finish()
End Sub
```

Ablauf wie ROBO Pro, Schliessen und Oeffnen in Sub's.

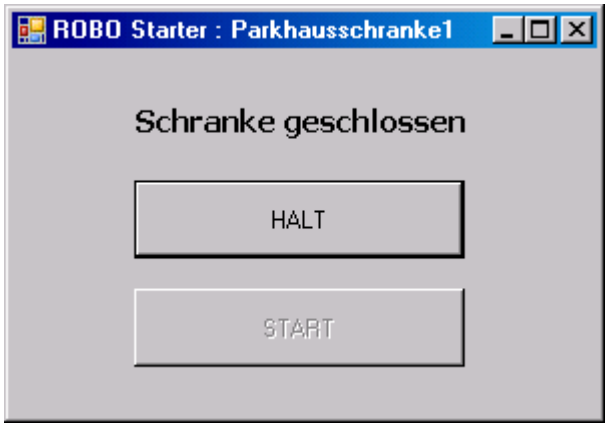
Parkhausschranke2

```
Code = ""
PrintStatus "Schranke bereit"
Do
  Schliessen
  Do
    Code = InputBox("Bitte Code eingeben")
  Loop Until Code = "352"
  Oeffnen
  Code = ""
```

Anstelle des schlichten Drückens von Taster iSchrankeOeffnen jetzt die Eingabe einer Code-Nummer über eine InputBox.

VB2005

Parkhausschranke1



Ablauf wie bei ROBO Pro beschrieben

```
Private Sub cmdAction_Click(ByVal sender As System.Object, _
                ByVal e As System.EventArgs) Handles cmdAction.Click
  Try
    ft.OpenInterface(IFTypen.ftROBO_first_USB, 0)
    cmdEnde.Text = "HALT"
    cmdAction.Enabled = False
    ft.SetMotor(mLichtschranke, Dir.Ein)
    ft.SetMotor(mRot, Dir.Ein)
    lblStatus.Text = "Schranke bereit"
    Do
      Schliessen()
      ft.WaitForInput(iSchrankeOeffnen)
      Oeffnen()
      ft.SetMotor(mRot, Dir.Aus)
      ft.SetMotor(mGruen, Dir.Ein)
      ft.WaitForHigh(iLichtSchranke)
      ft.Pause(500)
      ft.SetMotor(mGruen, Dir.Aus)
      ft.SetMotor(mRot, Dir.Ein)
    Loop Until ft.Finish()
    lblStatus.Text = "Schrankenbetrieb beendet"
    cmdEnde.Text = "ENDE"
    cmdAction.Enabled = True
    ft.CloseInterface()
```

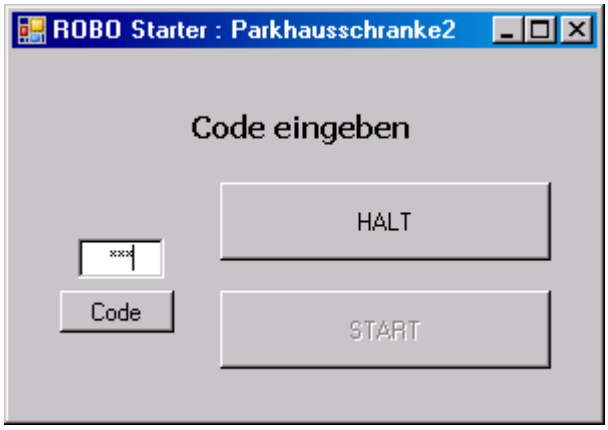
```
  Catch eft As FishFaceException
    lblStatus.Text = eft.Message
  End Try
End Sub
```

Schliessen und Oeffnen wurden in Subs verlagert. Zentraler Punkt hier ist das WaitforInput bei dem auf eine Anforderung zur SchrankenÖffnung gewartet wird und dann das WaitForHigh, bei dem auf eine Durchfahrt gewartet wird.

```
Private Sub Oeffnen()
  If ft.GetInput(iSchrankeOffen) Then Exit Sub
  lblStatus.Text = "Schranke öffnet"
  ft.SetMotor(mSchranke, cSchrankeOeffnen)
  ft.WaitForInput(iSchrankeOffen)
  ft.SetMotor(mSchranke, Dir.Aus)
  lblStatus.Text = "Schranke geöffnet"
End Sub
```

Die Oeffnen Sub. Das einleitende If soll ein "Motorknurren" bei offener Schranke verhindern. Schliessen() analog.

Parkhausschranke2



Wie 1, die Anforderung zum Öffnen wurde durch die Eingabe einer Codenummer + Code-Click ersetzt.

```
CodeOK = False
lblStatus.Text = "Schranke bereit"
Do
  Schliessen()
  Do
    lblStatus.Text = "Code eingeben"
    ft.Pause(100)
  Loop Until CodeOK Or ft.Finish()
  Oeffnen()
  CodeOK = False
  ....
```

Das WaitForInput von Parkhaus1 wurde durch ein Warten auf CodeOK = True ersetzt.

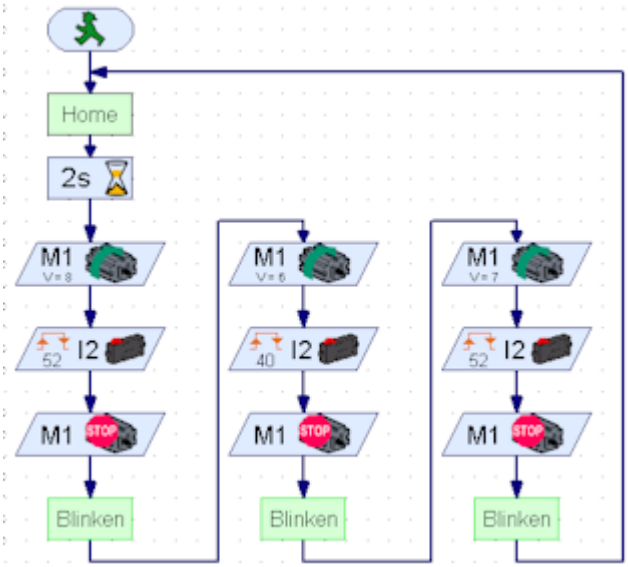
```
Private Sub cmdCode_Click(ByVal sender As System.Object, ...
  CodeOK = False
  If txtCode.Text = "352" Then CodeOK = True
  txtCode.Text = ""
End Sub
```

Das geschieht in der Click-Routine zu cmdCode nachdem auf die Codeziffern verglichen wurden. Gegenüber der ROBO Pro Lösung eine radikal einfachere Methode. Achtung die Eigenschaft PasswordChar von txtCode wurde auf "*" gesetzt.

Schweißroboter

ROBO Pro

Schweissroboter



Fährt in einer Endlosschleife zuerst die Home-Position (Endtaster I1) und dann um 52 – 40 – 52 Increments die Schweißpositionen an, dort wird heftig geblinkt.

VBA

SchweissRoboter

```
Sub Home()
  If GetInput(ftiI1) Then Exit Sub
  SetMotor(ftiM1,ftiLinks)
  WaitForInput(ftiI1)
  SetMotor(ftiM1,ftiAus)
End Sub

Sub Main
  Do
    PrintStatus "Home"
    Home
    Pause(2000)
    PrintStatus "Position 1"
    SetMotor(ftiM1,ftiRechts,ftiFull,52)
    WaitForMotors(0,ftiM1)
    Blinken
    PrintStatus "Position 2"
    SetMotor(ftiM1,ftiRechts,ftiFull,40)
    WaitForMotors(0,ftiM1)
    Blinken
    PrintStatus "Position 3"
    SetMotor(ftiM1,ftiRechts,ftiFull,52)
    WaitForMotors(0,ftiM1)
    Blinken
  Loop Until Finish()
End Sub
```

Der eigentlich so vertraute SetMotor hat noch weitere Parameter : Angabe der Geschwindigkeit und der zurückzulegenden Strecke in Impulsen. Dazu wird eine feste Belegung der zugehörenden Taster vorausgesetzt : Bei Motor an M1 – Endtaster an I1 und Impulstaster an I2. Der über SetMotor gestartete Motor schaltet sich nach Erreichen der vorgegebenen Anzahl Impuls selbstätig wieder ab. Hier wird das mit WaitForMotors abgewartet. Die Parameterliste könnte noch weitere Motoren enthalten.

Über Sub Home() wird die Home-Position (am Endtaster) angefahren. Das If GetInput soll ein "knurren" des Motors, wenn der Robot schon am Endtaster steht.

Ein MoveHome / MoveTo, wie bei VB2005, steht bei vbaFish40 nicht zur Verfügung.

VB2005

Schweissroboter

dto. Zusätzlich werden die Aktionen im Statusfeld angezeigt.

```
Private WithEvents ft As New FishRobot(New Integer(,) {{1, 222}})
```

Diesesmal wird mit der Klasse FishRobot gearbeitet, die es erlaubt mehrere Robot-Komponenten simultan (gleichzeitig) zu betreiben, das wird hier allerdings nicht genutzt. Wohl aber die Beschreibung der einzelnen Robot-Komponenten bei der Instanzierung : Erste Komponente an M1, max. Fahrweg 222 Impulse, indirekt zugeordnet : I1 als Endtaster und I2 als Impulszähler.

```
Do
  lblStatus.Text = "Home"
  ft.MoveHome()
  ft.Pause(2000)
  ft.MoveDelta(52)
  Blinken()
  ft.MoveDelta(40)
  Blinken()
  ft.MoveDelta(52)
  Blinken()
Loop Until ft.Finish()
```

Die fällige Do .. Loop Schleife gerät deswegen hier etwas knapper. MoveHome und MoveDelta sind Methoden der Klasse FishRobot. Bei Einsatz der Methode MoveTo kann man eine bestimmte Position ohne Kenntnis der aktuellen Anfahren : ft.MovoTo(144) heißt z.B. direktes Anfahren der Position 3 des Beispiels. Die Anzeige der aktuellen Position erfolgt in einer Eventroutine von FishFace :

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