C# Ecke

ftComputing : Programme für die fischertechnik-Interfaces und -konstruktionskästen

ftComputing.de

Home
Back
Über Threads

Sitemap
Index
Links
Impressum

Mail

Allgemeines

C# ist eine Sprache der .NET Framework Sprachfamilie und kommt mit einer eigenen umfangreichen Klassenbibliothek und dem .NET Framework Laufzeitsystem. Entwicklungsumgebung ist Visual Studio.NET. C# und alternativ die freie Entwicklungsumgebung SharpDevelop. C# vereint die wesentlichen Element von VC++ mit der Einfachheit von Visual Basic.

Die C# Ecke soll aufzeigen wo innerhalb der Site C#spezifische Themen zu finden sind, aber auch die Programmierung der fischertechnik Interfaces mit C# beschreiben.

Literatur :
Eric Gunnarson : C#, Galileo, zweite Auflage, ISBN 3-89842-183-X (deutsch) als fundierte Einführung,
Nitty Gritty C#, Addison-Wesley (deutsch). Handfeste und preiswerte Einführung/Übersicht für Programmierer mit Erfahrung.
O'Reilly : Programming C# 2. Auflage, ISBN 0-596-00309-9, als Übersicht und
O'Reilly : C# in a Nutshell, 0-596-00181-9, als Referenz neben der recht ansprechenden Hilfe des Visual Studio.NET
Und dann gibt es jetzt auch von den altbekannten VB-Autoren eine C# Version :
Frank Eller, Michael Kofler : Visual C# - Grundlagen, Programmiertechniken, Windows-Programmierung. Addison-Wesley Verlag ISBN 3-8273-2073-9
Doberenz / Kowalski : Visual C#.NET - Grundlagen und Profiwissen. Hanser Verlag. ISBN 3.446.22021-6.
In den beiden letztgenannten Büchern wird auch ausführlich auf die Programmierung mit Windows.Forms eingegangen.

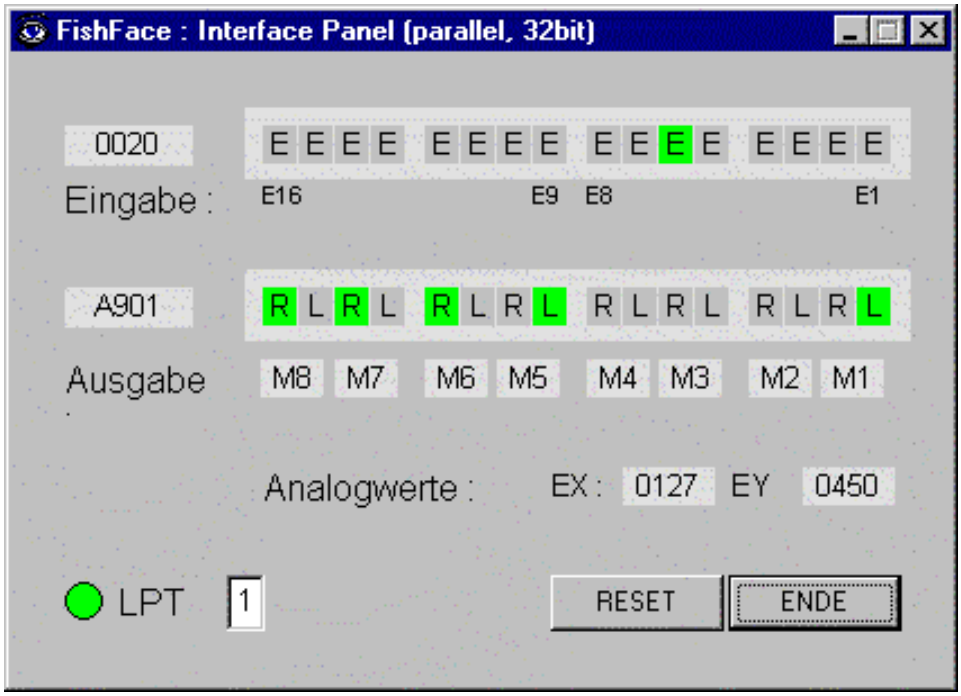
Programmierung

Basis für die Programmierung ist die in VC++ 6.0 erstellte DLL umFish30.DLL für die im Päckchen umFish30.ZIP die

erforderlichen Deklarationen, Klassen und Beispiele bereitgestellt werden. Darauf baut die Assembly FishFa30.DLL auf die zusammen mit Beispielen und dem Handbuch in csmodel.zip enthalten ist :

- Programmierung auf Basis der **native Schnittstelle** von umFish30.DLL (cs-Variante). Die erforderlichen Deklarationen sind in der Source FishFa30.CS (ganz am Anfang) zu finden, sie können 1:1 in eigene Klassen übernommen werden. Die Programmierung auf Grundlage der native Schnittstelle ist sinnvoll, wenn die Absicht besteht, eigene Zugriffssoftware zu entwickeln.
- Die Klasse **FishFace** als Bestandteil der Assembly FishFa30.CS / FishFa30.DLL kapselt die Funktionen von umFish30.DLL in einer C#gerechten Weise und bietet mit den Wait-Funktionen eine Abrundung insbesondere für die (asynchronen) SetMotor(s) Methoden.
  Für die Mehrzahl der Methoden gibt es eine Reihe von Überladungen, die eine Übergabe der parameter sowohl als int-Wert (-> eigene Konstanten, tabellarische Werte) wie auch enums erlauben. Meist werden so auch default Parameter realisiert.
- Klasse FishRobot zu Programmierung von Robots a la "Industry Robots" : Motor mit zugeordnetem Impulsrad/-taster und einem Endtaster.
- Klasse FishStep mit Methoden zur Programmierung einzelner Schrittmotoren und von Schrittmotorpaaren im XY-Verbund (Plotter).

InterfacePanel / Installation



Das **Interface Panel** ist ein separates Tool zur Anzeige und Steuerung eines Interfaces. Es kann besonders zum Test des

Modellaufbaus und zum "GeradeRücken" eines Modells eingesetzt werden. Das Interface Panel sollte als erstes Programm zur Kontrolle der korrekten Installation von Interface und umFish30.DLL Software eingesetzt werden. Es ist Bestandteil des Päckchens umFish30.ZIP (umFishDP30.EXE).

Zur Installation reicht das Kopieren des Inhaltes von umFish30.ZIP in geeignete Verzeichnisse. Für das Universal Interface (LPT) sind (vorbereitete) Registry-Einträge erforderlich.

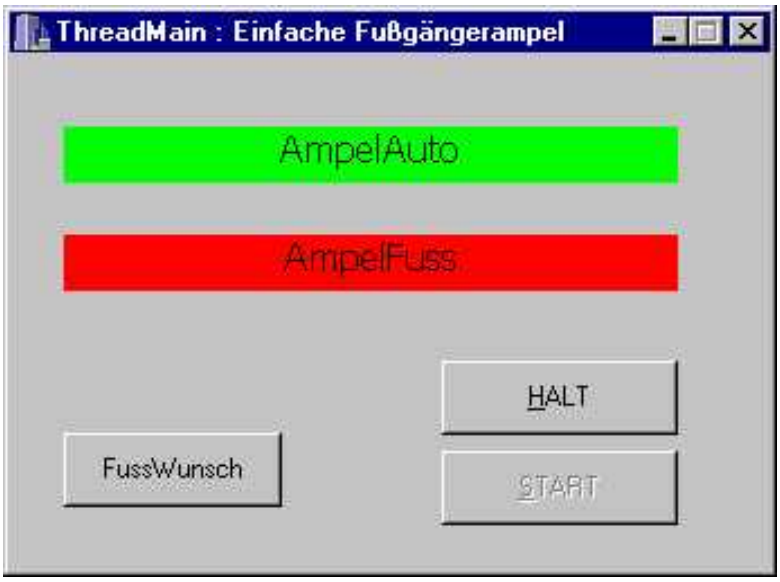
Handbuch

Zur Einführung in die Programmierung der fischertechnik Interfaces unter Nutzung der Assembly FishFa30.DLL und der Programmiersprache C# gibt es ein ausführliches Handbuch mit den Abschnitten Einführung - Beispiel Riesenrad - Referenz - Tips & Tricks. Es ist zusammen mit den im Handbuch angeführten Beispielen in csModel.ZIP untergebracht.

Beispiele

Das Handbuch enthält neben dem obligatorischen HelloFish ein durchgehendes Beispiel zum Betrieb eines Riesenrads (wahlweise mit einem kleinen Simulationsmodell) Hinzu kommen die unten aufgeführten Modellprogramme. Alles zusammen in : csModel.ZIP.

Modelle



AmpelThread : Betrieb einer Fußgängerampel an einem Intelligent Interface mit mit Extensionmodul. Die Auto- sowie die Fußgängerampel laufen in eigenen Threads.

Stanze : Betrieb des Modells [Stanzmaschine mit Transportband 51 663]

RobStanze :Beschicken der Stanze 51 663 durch einen Industry Robot (Knickarm- oder SäulenRobot). Die Programmteile für den Betrieb von Stanze bzw. Robot laufen in eigenen Threads.

Die Sources der Programme sind in csModel.ZIP enthalten.

Der Turm von Hanoi

Programm auf Basis der Klasse HanoiRob. In HanoiRob.ZIP enthalten. Siehe auch die Seite Turm von Hanoi.

Details / Hinweise

- Anmerkungen zu den Thread-Programmen

Stand : 06.06.2004

Über Threads

ftComputing : Programme für die fischertechnik-Interfaces und -konstruktionskästen

Allgemeines

umFish30.DLL ist durch Trennung von Zugriff auf das Interface und den Zugriff der Anwendung auf die Interface Werte threadfähig. Das geschieht durch Auslagerung des Interfacezugriffs in einen eigenen Poll-Thread des Steuerung über den MultiMediaTimer erfolgt. Im Poll-Thread werden die Daten des Interfaces in regelmäßigen Intervallen abgefragt und gleichzeitig die Aufträge der Anwendung übermittelt. Die Kommunikation mit der Anwendung erfolgt über einen Kontrollblock (den ftiDCB), der von der Anwendung asynchron ausgelesen bzw. besetzt wird.

So ist es möglich auf ein Interface (ggf. mit angeschlossenem Slave (Extension Module) unabhängig aus verschiedenen Threads zuzugreifen. Der eigentliche Zugriff auf die nur einmal vorhandene Resource Interface erfolgt immer über den ebenfalls nur einmal vorhandenen Poll-Thread. Der Zugriff auf einzelne Ein- bzw. Ausgänge ist ohne weitere Maßnahmen möglich. Beim gleichzeitigen Zugriff auf alle M-Ausgänge ist eine Maskierung der vom eigenen Thread nicht genutzten M-Ausgänge im MotorStatus-Wort erforderlich, da umFish30.DLL keine Verwaltung von TeilResourcen bietet. Maskierung meint hier Erhalt des Status der "fremden" M-Ausgänge.

Die nachfolgenden Beispiele setzen die Klassen **Thread** und **AutoResetEvent** der .NET Framework Klassenbibliothek des Namespace System.Threading (using System.Threading;) ein. Übersichten und Details dazu siehe Hilfe System.Threading-Namespace und Thread-Member sowie WaitHandle-Member, AutoResetEvent-Member. Die .NET Framwork Klassen sind von Haus aus threadfest. Literatur Eric Gunnarson. Das Thread Kapitel ist instruktiv, es fehlt aber ein AutoResetEvent Beispiel.

Details

Thread Klasse zum Erstellen und Betreiben eines Threads :

```
private Thread robThread = new Thread(new ThreadStart
robExecute));
```

Erstellen eines neuen Threads und der Nutzroutine als Callback Routine (Delegate). Die Nutzroutine ist also Bestandteil (eine Methode) der umgebenden Klasse. Der Thread wird erst durch `robThread.Start()` gestartet und durch Erreichen des Endes der Routine beendet.

AutoResetEvent Klasse zum Erstellen und Betreiben eines Synchronisations Mechanismus (Basis Waithandle), Alternative ManualResetEvent. AutoResetEvent wird nach Abfrage zurückgesetzt ManualResetEvent nicht :

```
private AutoResetEvent robReady = new
AutoResetEvent(false);
```

Event ist nicht signaled (False)

Programm RobStanze

Die Programme **AmpelThread** und **RobStanze** haben die gleiche Struktur. AmpelThread ist im Test wesentlich nervenschonender (es verhakt da nichts) und sollte der Ausgangspunkt eigener Bemühungen sein. RobStanze ist das attraktivere Modell, deswegen wird es hier besprochen. Bei den vorliegenden Programmen wird bei AmpelThread mit einem Intelligent Interface plus Extension Module an COM1 gearbeitet, bei RobStanze sind es Intelligent Interfaces an COM1 und COM2 (weil die Modelle so aufgebaut waren und weil die Verkabelung so einfacher ist (das Verbindungskabel des Extensionmodules ist zu kurz)) :

frmMain	Der Hauptthread mit Form und Gesamtsteuerung
ftR	Instanz der Klasse FishFace zur Rob-Steuerung
ftS	Instanz der Klasse FishFace zur Stanz-Steuerung
umFish30.DLL	im Hintergrund : mit dem Poll-Thread
robReady	Event : Roboter hat Teil plaziert Standard Security, autoReset, not signaled
stanzeReady	Event : Stanze kann neues Teil verarbeiten Standard Security, autoReset, not signaled
robThread	Thread mit der Robotersteuerung
stanzThread	Thread mit dem Stanzprogramm

frmMain global

- **Anlegen der ftR und ftS Instanzen** :
  `private FishFace ftR = new FishFace(false, false, 0);`

```
private FishFace ftS = new FishFace(false,
false, 0);
```

ohne AnalogScan, ohne Slave, mit default PollIntervall

- **Thread Member**
  ```
  private Thread robThread;
  private Thread stanzeThread;
  ```
  Instanziert werden kann hier noch nicht, da die Callback Routine nicht statisch ist.
- **Events**
  private AutoResetEvent robReady;
  private AutoResetEvent stanzeReady;

cmdAction_Click

- **Herstellen der Verbindung zu den Interfaces** :
  ```
  ftR.OpenInterface("COM2", false);
  ftS.OpenInterface("COM1", false);
  ```
  ohne Unterbrechung durch `Application.DoEvents();`
  in einer try - catch Klammer um Openfehler abzufangen. Weitere try/catch wären sinnvoll, aber man hört es auch so laut genug.
- **Anlegen der Threads und Events für Rob / Stanze**
  ```
  robThread = new Thread(new
  ThreadStart(robExecute));
  stanzThread = new Thread(new
  ThreadStart(stanzThread));
  ```
  ohne sie zu starten
  ```
  robReady = new AutoResetEvent(false);
  stanzeReady = new AutoResetEvent(false);
  ```
  nicht signalisiert
- **Starten der Threads**
  ```
  robThread.Start();
  stanzThread.Start();
  ```

robExecute

Die Nutz-(CallBack/Delegate)Routine des Threads.

- **Anfahren der Home Position**
  ```
  ftR.SetMotor(mSaule, ftiLinks, 15, 999);
  ftR.SetMotor(mArmV, ftiLinks, 15, 999);
  ftR.SetMotor(mArmH, ftiLinks, 15, 999);
  ftR.SetMotor(mGreifer, ftiLinks, 15, 999);
  ftR.WaitForMotors(0, mSaule, mArmV, mArmH,
  mGreifer);
  ```
  Alle vier Motoren des Robot werden simultan mit voller Geschwindigkeit gestartet (15), sie sollen 999 Impulse oder bis zum Erreichen des zugehörenden Endtasters laufen (ca. 360 Impulse sind eine volle Drehung, also bis zu den Endtastern). WaitForMotors wartet auf das Erreichen aller vier Endpositionen eine unbegrenzte Zeit (0).

- **BetriebsSchleife**
  `do {} while(!ftR.Finish());`
  wird durchlaufen bis der Thread von außen einen EndeWunsch empfängt (siehe auch cmdEnde_Click).
- **In der Betriebsschleife**
  `frmThreadMain.lblStatus.Text = "....";`
  Anzeige des aktuellen Status
  `ftR.SetMotor(...`
  `ftR.SetMotor(...`
  `ft.WaitForMotors(0, ...);`
  Ausführen der Teilschritte Greifen des Teils, Weg zur Ablage auf dem Transportband,
- **Weiter mit Warten auf stanzeReady**
  `stanzeReady.WaitOne();`
  Ablegen auf dem Transportband nach dem Signal
  Nach dem Ablegen : Setzen Signal RobReady
  `robReady.Set();`
  und ohne Halt zurück zum Magazin.
- Und nach Erhalt des **Ende-Wunsches**
  Fahren des Robots auf eine Ruheposition.

stanzExecute

Nutzroutine des Threads, ähnlich wie robThread.

- **Herstellen der Startposition**
- **Betriebsschleife**
- **In der Betriebsschleife**
  Gleich zu Anfang Warten auf **robReady** und anschließendes Signalisieren **stanzeReady**
- **Weiter in der Betriebsschleife**
- Und nach Erhalt des **Ende-Wunsch**es
  Abstellen der Lampen.

cmdEnde_Click

Kunstvolles Beenden des Stanzbetriebes :

- **Ende-Wunsch an Robot**
  `ftR.NotHalt = true;`
  `stanzeReady-Set();`
  `robThread.Join();`
  NotHalt wird beim Schleifenende durch Finish ausgewertet, die Schleife und damit der Thread werden beendet. Vorher muß es aber erst soweit kommen : normales signalisieren StanzeReady. Anschließend Warten auf das Thread-Ende.
- **Ende-Wunsch an Stanze**
  analog Robot
  daran denken es wird noch ein Teil benötigt, wenn die Testbefehle
  `ftS.WaitForLow(ePhotoV);`

```
ftS.Pause(1234);
```

noch vorhanden sind.

- Abschlußarbeiten.

Stanze 51663

Stanzmaschine mit Transportband No. 51 663

Freier Nachbau der Stanzmaschine aus dem Katalog 2001/2002 aus der Rubrik Trainingsmodelle (dazu mehr) versehen mit Betriebsprogrammen in acht Programmiersprachen : LLWin 3.0, Visual Basic 6, Delphi4, C++Builder4, C#, VB.NET, JavaSwing und VBScript.

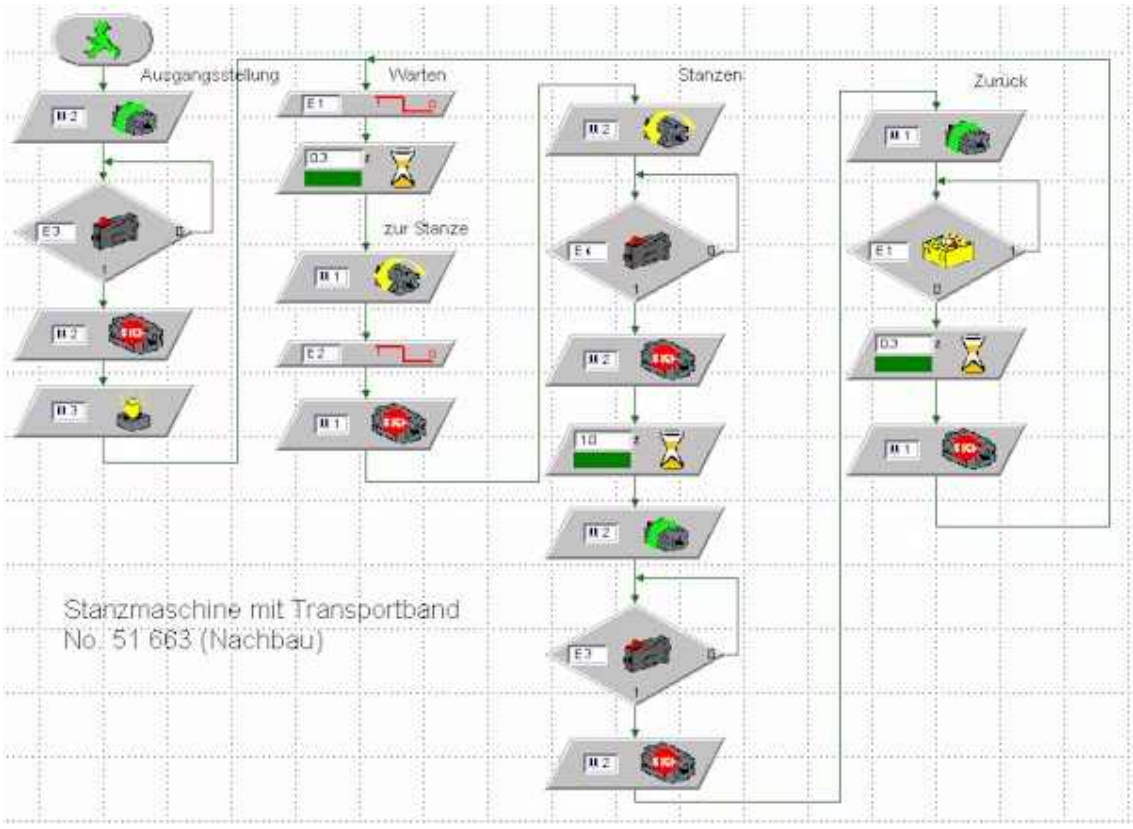
Modelldaten :

```
Private Const mBand = 1              ' Bandmotor an M1
Private Const cbVor = ftiLinks       ' zur Stanze linksdrehend
Private Const cbRuck = ftiRechts     ' und zurück

Private Const mStanze = 2            ' Stanzenmotor an M2
Private Const eOben = 3              ' Taster oben an E3
Private Const eUnten = 4             ' Taster unter an E4
Private Const csAb = ftiRechts       ' abwärts : linksrum
Private Const csAuf = ftiLinks       ' auf : rechtsrum

Private Const ePhotoV = 1            ' Photowiderstand vorn an E1
Private Const ePhotoS = 2            '     bei Stanze       an E2
Private Const mLampen = 3            ' Lampen für Photow.   an M3
```

Neu am Modell ist die Kiste, in die die gestanzten Teile abgelegt werden. Man kann dafür natürlich auch einen Industry Robot einsetzen.... Für das Transportband wurden 20er Zahnräder verwendet (das Band schleift etwas), Original 15er. Als Werkstücke wurden 2 aufeinandergesteckte Radscheiben (23 mm) genommen.

LLWin 3.0

Der **Ablauf** wird anhand des **LLWin-Programms** beschrieben :

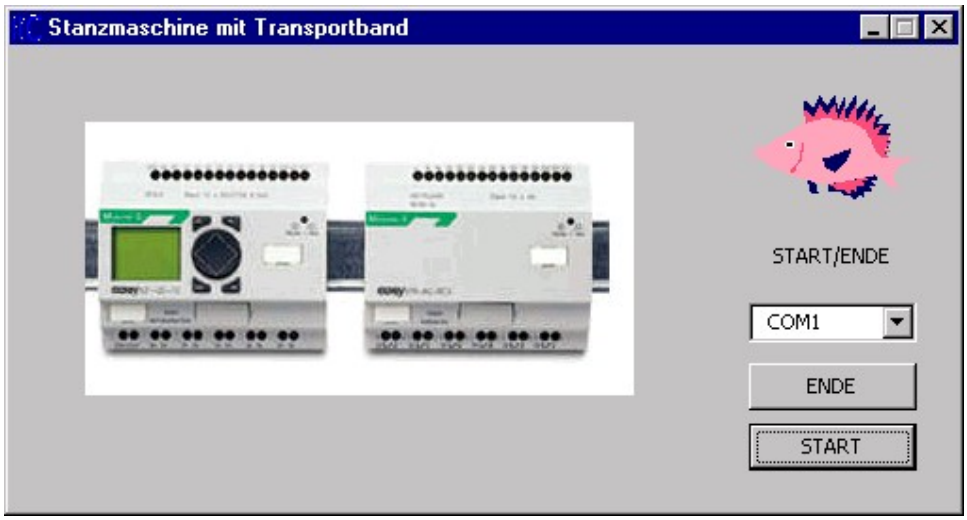


- Bei Start des Programms wird das Modell in Ausgangslage gebracht : Stanze fährt nach oben und die Lampen für die Lichtschranken werden eingeschaltet.
- Warten auf einen true/false-Durchgang an der vorderen Lichtschranke (true heißt kein Teil in der Lichtschranke, wenn da noch ein altes lag muß man es wegnehmen).
- Damit das neue Teil nicht gleich wegreißt 0,3 Sek. warten.
- Dann gehts zur Stanze bis wieder ein true/false-Durchgang (Lichtschranke offen/geschlossen) das Band stoppt.
- Die Stanze fährt nach unten bis Endtaster.
- GedenkSekunde
- Es geht wieder aufwärts.
- und zurück bis die vordere Lichtschranke schließt.
- und darüber hinaus in die Ablage P. Der Bandmotor hat 0,3 Sekunden Nachlauf

Nach Oben

Visual Basic 6

Das **Visual Basic Programm** (erstellt mit dem Template ftComputing50, Einsatz von FishFa50.OCX/umfish20.DLL) sieht dann so aus :



Außer dem Bild (das kommt später) sieht man wenig und das Programm (VB6-Version) dazu funktioniert denn auch wie oben beschrieben :

```
Private Sub Action()

' --- Routine für den Modellbetrieb, kann weitere Subs aufrufen ---
-

ft.ClearMotors
ft.SetMotor mStanze, csAuf
ft.WaitForInput eOben
ft.SetMotor mStanze, ftiAus
ft.SetMotor mLampen, ftiEin

Do

lblStatus = "Wartet"
ft.WaitForLow ePhotoV
ft.WaitForTime 500

lblStatus = "Zur Stanze"
ft.SetMotor mBand, cbVor
ft.WaitForLow ePhotoS
ft.SetMotor mBand, ftiAus

lblStatus = "Stanzen"
ft.SetMotor mStanze, csAb
ft.WaitForInput eUnten
ft.SetMotor mStanze, ftiAus
ft.WaitForTime 1000
ft.SetMotor mStanze, csAuf
ft.WaitForInput eOben
ft.SetMotor mStanze, ftiAus

lblStatus = "Zurück"
ft.SetMotor mBand, cbRuck
ft.WaitForInput ePhotoV, False
ft.WaitForTime 300
ft.SetMotor mBand, ftiAus

Loop Until ft.Finish(0)

End Sub
```

Neu ist eigentlich nur die zusätzliche Anzeige des Betriebszustandes.

Nach Oben

Jetzt noch eine Übersicht weiterer Sprachversionen

Weitere Sprach-Versionen von Stanze

Die eigentlichen Betriebsprogramme für das Stanze-Modell sind alle sehr ähnlich, da sie alle die FishFace Methoden nutzen. Es werden deswegen auch nur beispielhaft zwei Statements gezeigt um den Variantenreichtum heutiger Programmiersprachen zu characterisieren. Zusätzlich ein paar allgemeine Anmerkungen zur Implementierung. Die Variablenschreibweise entspricht Empfehlungen zur jeweiligen Sprache (soweit ich sie kapiert und akzeptiert habe).

Die Programme haben eine Sperre gegen unkontrolliertes Beenden. Sie ist durch Enabled = False der entsprechenden Buttons, Änderung der Beschriftung und Sperre des (x) rechts oben realisiert.

Alle Programme nutzen umFish30.DLL bzw. umFish20.DLL. Beide DLLs setzen den MultiMediaTimer ein, der in einem Extra-Thread eine CallBack-Routine zur Steuerung des Interfaces aufruft. so ist eine solide Zeitbasis zum Pollen der E-Eingänge auf dem nicht Realtime-fähigen Windowssystem erreichbar.

Delphi4

```
lblStatus.Caption := 'Wartet';

ft.SetMotor(mStanze, csAuf);
```

Genutzt wird die FishFa50.DCU auf Basis von umFish20.DLL

Nach Oben

C++Builder4

```
lblStatus->Caption = "Wartet";

ft.SetMotor(mStanze, csAuf);
```

Genutzt wird die FishFa30.H/CPP Unit auf der Basis von umFish30.DLL.
Es wird #include <vcl.h> eingesetzt. Die Oberfläche ist also nahe an Delphi.

Nach Oben

C#

```
lblStatus.Text = "Wartet";

ft.SetMotor(mStanze, csAuf);
```

Genutzt wird die FishFa30.CS Assembly als Source auf Basis von umFish30.DLL (cs-Variante).

Nach Oben

VB.NET

```
lblStatus.Text = "Wartet"

ft.SetMotor(mStanze, csAuf)
```

Genutzt wird die in C# geschriebene FishFa30.DLL in kompilierter Form (Basis umFish30.DLL, cs-Variante).

Nach Oben

JavaSwing

```
lblStatus.setText("Wartet");

ft.setMotor(mStanze, csAuf);
```

Genutzt werden die Klassen JavaFish und FishThread, die den Zugriff auf die Wrapper.DLL javaFish.DLL (VC++ mit speziellen Java-Konstrukten (JNI-Interface) kapselt. javaFish.DLL nutzt dann umFish20.DLL. Da Java ein Konstrukt wie Application.ProcessMessage (Abgabe der Prozesszeit an den Windowsloop) nicht kennt, wurde hier das eigentliche Betriebsprogramm (class Stempel extends FishThread) in einen eigenen Thread gelegt um die Reaktionsfähigkeit der Anwenderoberfläche sicherzustellen.

Nach Oben

VBScript

```
---

ft.SetMotor mStanze, csAuf
```

Genutzt wird FishFa50D.DLL auf Basis von umFish20.DLL

Nach Oben

Download

Im Download-Päckchen Stanze sind die Source zusammengefaßt. Zusätzlich ist noch eine DLL und weitere Zugriffssoftware erforderlich, die Angaben dazu sind im ReadMe des Stanze-Päckchens aufgelistet.

Nach Oben

EasyTrainer von Moeller

Die Firma Moeller (www.moeller.net) hat für ihre speicherprogrammierbaren Relais ein Mini-Trainer Set zusammengestellt, das auch eine Anschlußplatine für fischertechnik enthält (24V), Anschüsse wie beim Interface. Preis liegt liegt das Set im privaten Rahmen (ca. 250 Euro). Die 24V-Teile (Motoren Lampen) kann man problemlos bei www.knobloch-gmbh.de bestellen. Sie passen gut zu den normalen fischertechnik-Teilen.

Der Mini-Trainer bietet einen einfachen Einstieg in die Speicherprogrammierbare Steuerrung (SPS) auf Basis von Stromlaufplänen. Programmiert werden kann direkt am Relais oder mit einem komfortablen Editor am PC.

Ist etwas für Berufsschulen. (Punkt) Sowie Elektriker und E-Ing.s, die es auch zu Hause nicht lassen können.

Man sollte sich mal im (Heizungs)Keller umsehen, vielleicht gibt es da schon so ein Relais zum Thermo-Management oder mal hinter (Schrank)türen sehen da könnte es beim Security-Management (Tag/Nachtlichtschaltung, Bewegungssensor oder Rasensprenger). Man weiß es dann richtig zu schätzen.

Nach Oben

Turm von Hanoi

Über das Scheibenschichten

Ein Stapel von immer kleiner werdenden Scheiben soll über eine Zwischenstation auf einen neuen Platz gestapelt werden. Immer eine aufeinmal und stets die kleinere Scheibe auf die größere.

Die Lösung dieses Problems erfreut sich im Informatik-Unterricht großer Beliebtheit (bei den Lehrern). Es gibt deswegen auch immer wieder Lösungen zu diesem Problem. fischertechnik hat bereits im Kasten Computing von 1984 das Modell eines Hanoi-Roboters angeboten. Auf dieser Site finden sich ebenfalls einige Lösungen für das Problem :

- WinLogo Lösung für den Hanoi Robot von 1984
- VB6 Lösung für den Industry Robot von 1998
- VB6 Lösung für den LynxMotion Robot

Hier Lösungen für **VBA** mit vbaFish, **JScript** mit mscFish, **VB.NET**, **C#.NET**, **Python** und **Delphi**, alle nutzen FishFa30/umFish30.DLL :

Die manuelle Lösung

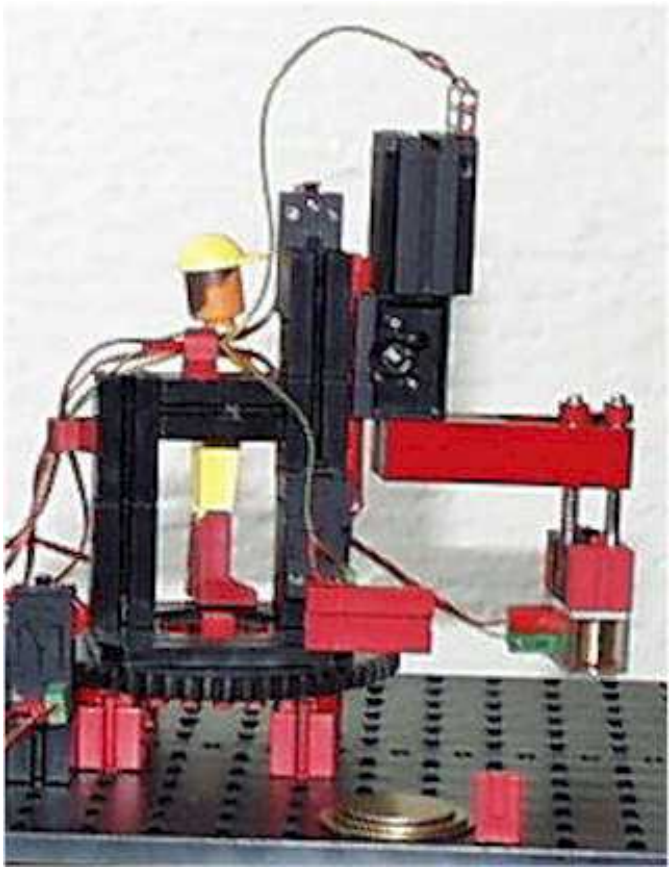
Das nachfolgende kleine Programm Hanoi.ftC (es wurde im PC Magazin Nr. 22 von 1988 als Quick Basic Lösung veröffentlich) gibt im Debug-Fenster eine Lösungsanweisung für das manuelle Umschichten aus :

```
Sub Main
Dim n&
  n = InputBox("Anzahl Scheiben")
  Debug.Clear
  Debug.Print "Spiel mit" & n & " Scheiben"
  Hanoi n, "Links ", "Mitte ", "Rechts"
End Sub

Sub Hanoi(n&, TAnf$, TMit$, TEnd$)
  If n = 1 Then
    Debug.Print "Scheibe von " & TAnf & " -> " & TEnd
  Else
    Hanoi n-1, TAnf, TEnd, TMit
    Debug.Print "Scheibe von " & TAnf & " -> " & TEnd
    Hanoi n-1, TMit, TAnf, TEnd
  End If
End Sub
```

Kern ist das Unterprogramm Hanoi, das aus einer einzigen If-Anweisung besteht. Wenn die Anzahl der Scheiben auf dem Stapel bei 1 angekommen ist, wird direkt vom aktuellen Quellstapel zum Zielstapel geschichtet, sonst wird dreimal hin und her geschichtet. Trick dabei ist der rekursive Aufruf von Hanoi (das Unterprogramm ruft sich selber auf).
Achtung : Zur Anzeige der Lösung ist im Menü Ansicht die Option 'Immer teilen' einzustellen

Die Anweisung Debug.Print "Scheibe von " & TAnf & " -> " & TEnd, die zweimal im Programm auftaucht, ist nun durch eine Anweisung Ziehe zu ersetzen, die einen Robot überredet tätig zu werden, ganz wie im manuellen Verfahren. Dazu ist dann erstmal ein Robot erforderlich :

Der Hanoi Robot



Der Hanoi Robot ist ein Eigenbau der auf dem Schweißroboter-Chassis des Computing Starter Kits basiert. Er entspricht hier weitgehend dem Hanoi Robot von 1984. Allerdings wurde die Technik modernisiert :

M1 : Säulenmotor mit Impulsrad und Endtaster an E1, Impulstaster an E2

M2 : Armmotor mit Endtaster oben E3 und unten E4

M3 : Gefedert gelagerter Magnet (neuere Version bei Knobloch : 32363 - 17,20 Euro)

Gestapelt werden die Eisenscheiben, die im Bild vorn (PosA) neben dem Winkel liegen. Verdrahtet wurde "fliegend".

Die Routinen

-------	Hanoi-Logik
Ziehe(Von, Nach)	Ersatz für die manuelle Arbeitsanweisung : Zerlegt in Hole(Von) und Bringe(Nach)
Bringe(Pos)	Ablegen einer Scheibe auf der gewünschten Zielposition : SauleNach(Pos), ArmSenken, ScheibeGreifen, ArmHeben
Hole(Pos)	Aufgreifen einer Scheibe von der angegebenen Quellposition SauleNach(Pos), ArmSenken, ScheibeLegen, ArmHeben
-----	Robot-Ansteuerung
SauleNach(ZielPos)	Fahren auf die Zielposition. Die Position wird durch eine Impulsrad bestimmt
ArmSenken	Senken des Greiferarms bis zum Endtaster
ArmHeben	Heben des Greiferarms bis zum Endtaster
ScheibeLegen	Einfach : Magnet aus
ScheibeGreifen	Auch einfach : Magnet ein
-----	DrumRum
Grundstellung	Fahren auf Grundstellung (Robot auf PosA, Armhoch, Magnet aus)
Init	Besetzen der globalen Variablen. Dabei auch festlegen der Werte für die drei Stapelpositionen PosA, PosB und PosC

Die Sources

Das komplette VBA Programm gibts gleich zweimal :

- HanoiRobot.ftC : Eine all-in-one Lösung. Alle beschriebenen Routinen sind in einer Source enthalten
- HanoiMain.ftC/HanoiRobot.CLS : Die Robot-Ansteuerung wurde in eine separate Klasse ausgelagert. Hier wohl kaum erforderlich. Mehr um zu zeigen, wie man mit vbaFish auch Klassen einsetzen kann. Die Programme sind funktional gleich

eine Perl-Lösung :

- Hanoi.PL : Ein File mit dem kompletten Programm.

dann gleich noch einmal für Python :

- HanoiRobot.PY : Ein File, aber wie bei HanoiMain/HanoiRobot wurde die Robot-Ansteuerung in eine Klasse ausgelagert. Getestet wurde mit PythonWin. Ablauf auch als rreine

Konsolapplikation.

für Delphi :

- poorHanoiRob.DPR : Eine all-in-one Lösung auf Basis des Templates poorFish30. Alle beschriebenen Routinen sind in einer Source enthalten, Konsolprogramm.

und für JScript :

- HanoiRob.JS : Eine Lösung auf Basis von Unterprogrammen (function)
- HanoiRobC.JS : Eine Lösungsvariante unter Verwendung eines Objects (Klasse), das die Robot-Methoden kapselt.

und dann noch VB.NET und C#.NET:

- HanoiRobot : Eine Windows Lösung unter Nutzung der Klasse HanoiRob.

Alles zusammengefaßt in HanoiRob.ZIP. Zusätzlich ist noch vbaFish30Setup.EXE (VBA) bzw. PythonFish30.ZIP (Python), vbFish30Setup.EXE (Perl), umFish30.ZIP (C#, VB.NET) oder delphiFish30Setup.EXE (Delphi) erforderlich (Auf der jeweiligen Seite wird das genauer erklärt). Außerdem die zugehörenden Entwicklungssysteme - Ausnahme : vbaFish30, da ist schon alles dabei.

Stand : 08.02.2004