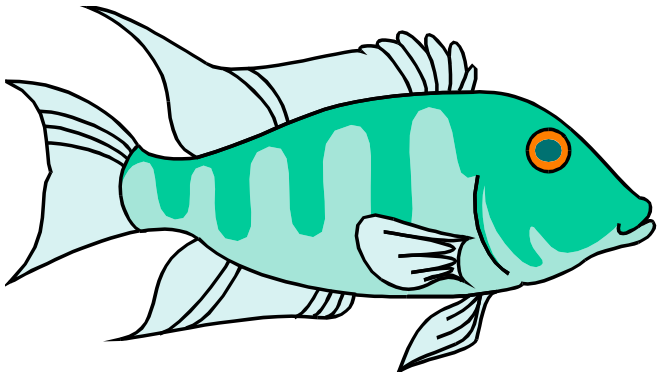
ftComputing

FishFaceTX für C#

Programmierung des ROBO TX Controllers mit C# 2005 und 2008

Ulrich Müller



Inhaltsverzeichnis

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873
eMail : UM@ftComputing.de
HomePage : www.ftcomputing.de C# Ecke : www.ftcomputing.de/csecke.htm

Dokumentname : FishFaTXCS.doc. Druckdatum : 02.12.2009
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish11.WMF

Einführung

Allgemeines

MIt der in C# geschriebenen Assembly FishFaceTX.DLL (Namensraum FishFaceTX) wird die Möglichkeit geboten, den fischertechnik ROBO TX Controller unter einer .NET 2.0 (und höher) Sprache zu programmieren (beschrieben wird hier der Einsatz von C# 2005). FishFaceTX.DLL setzt auf umFish50.DLL und die auf ftMscLib.DLL (einer systemkonforme.DLL, die von fischertechnik gestellt wird) auf. Die zentrale Klasse FishFace von FishFaceTX erlaubt die Ansteuerung der TX Controller über USB und Bluetooth.

Angeboten werden Befehle zur Schaltung der M-Ausgänge und zur Abfrage der Eingänge eines Interfaces. Der TX Controller wird durch einen MultiMediaTimer in umFish50.DLL überwacht. Dabei werden die Counter der C-Eingänge mit ihren Sollwerten verglichen. Bei Erreichen eines Sollwertes wird der Counter auf Null zurückgesetzt und der zugeordnete Motor abgeschaltet. Zusätzlich werden die Motoren beim Erkennen der ESC-Taster abgeschaltet.

Universal-Eingänge (I-Eingänge), die im Modus Analog laufen, liefern Raw-Werte im Bereich von 0 – 5000. Die M-bzw.O-Ausgänge können mit einer Power (PWM) im Bereich von 0-512 betrieben werden.

Die mit FishFaceTX erstellten Programme laufen im sog. "Online"-Betrieb, d.h. das auf dem Windows PC laufende Programm muß über USB bzw. Bluetooth mit dem Controller verbunden sein.

Getestet wurde mit : C# 2005 und VB2005 Express unter Vista mit .NET 3.5. Ein Ablauf unter .NET 2.0 ist auch möglich ebenso ist ein Upgrade auf C#2008 leicht durchzuführen.

Installation

Vorausgesetzt wird ein Windows System ab Windows 2000 mit einem installierten C# 2005 (ab Express Edition, C# 2008 ist auch möglich) mit dem .NET-Framework 2.0. Bezug der Express Edition siehe Literaturübersicht

Zusätzlich erforderlich ist die Installation des Programmpaketes "PC-Programming-11.zip" (oder höher) von www.fischertechnik.de Downloads. Es enthält, neben ftMscLib.DLL, die erforderlichen USB/Bluetooth Treiber und das Testtool "ROBO TX Test" für den Controllertest. Dort werden auch die erforderlichen COM-Namen für den Verbindungsaufbau mit dem TX Controller angezeigt (ROBO Pro tut das leider nicht).

Zusätzlich sollte das fischertechnik ROBO Pro installiert sein. Es ermöglicht die schnelle Erstellung einfacher Testprogramme.

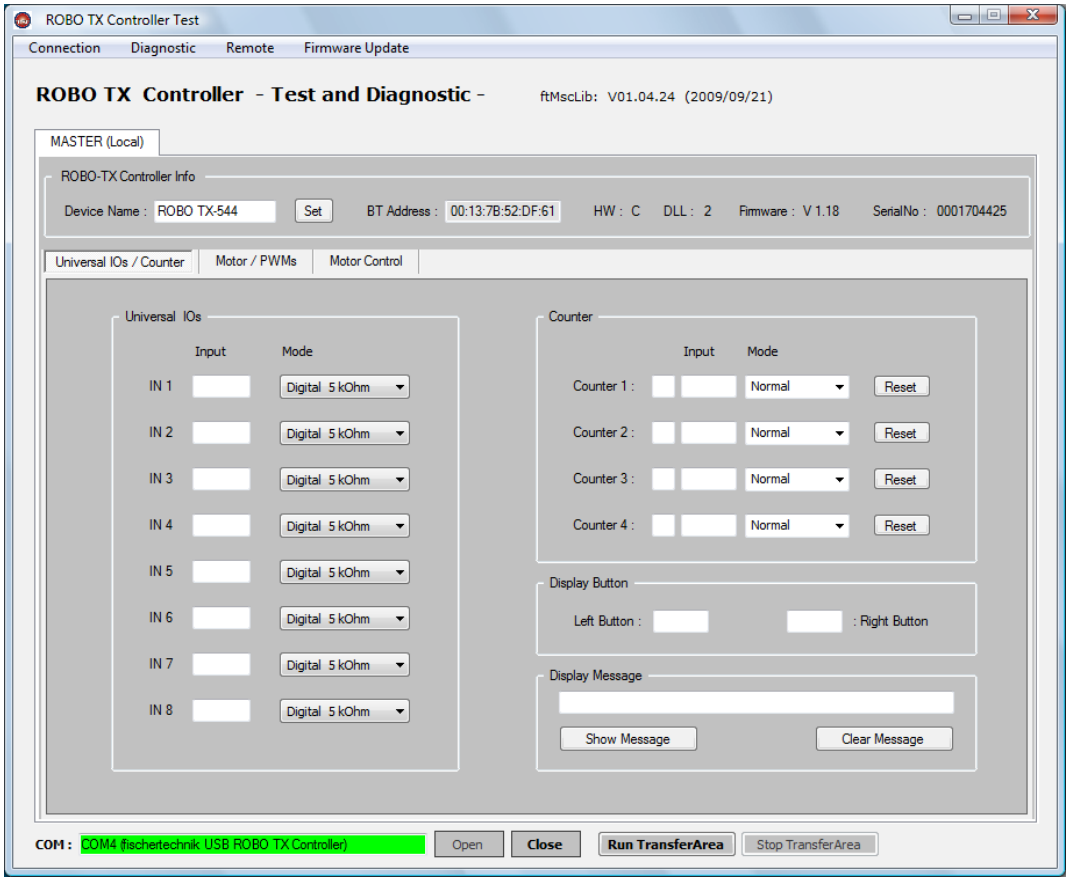
Das Paket FishFaceTX.zip enthält die Assembly FishFaceTX.DLL und die umFish50.DLL, nicht aber die ebenfalls erforderliche ftMscLib.DLL, sie ist in "PC-Programming-11.ZIP und in ROBO Pro enthalten. Zusätzlich im Paket enthalten sind Programmbeispiele und Programmvorlagen(Templates). FishFaceTX.DLL sollte an einem zentralen Ort untergebracht werden um die erforderlichen Verweise leicht einrichten zu können. Hinweis : Es kann vorkommen, daß in den BeispielProjekten die Verweise auf FishFaceTX.DLL nicht stimmen, nach dem Laden des Projektes einfach im Projektmappen Explorer (Verweise) richtigstelllen. umFish50.DLL bringt man am besten im Verzeichnis \Windows\System32 unter.

Robo TX Controller

Allgemein

Ein Master und 0 - 8 an den Master angeschlosse Extensions. Master und Extension sind baugleich, sie müssen aber über ihr Display entsprechend konfiguriert werden. Pro Rechner kann nur ein Master betrieben werden. Parallel dazu können aber ROBO Interfaces betrieben werden. Die Programme laufen stets im "online" Modus auf dem PC, sie können über einen USB-Anschluß oder Bluetooth betrieben werden. Der Bezeichnung für einen Ein- oder Ausgang muß bei den entsprechenden FishFaceTX-Methoden der Name der jeweiligen Extension vorangestellt werden, beim Main-Controller kann der Name entfallen.

ROBO TX Test Panel



Das Test Tool aus dem fischertechnik Download-Päckchen (PC-Programming-TXC11.zip)

Belegung bei den Code-Beispielen im weiteren Text meist :

M1 Motor mit 4er Impulsrad
C1 Taster als Impulszähler
I1 Endtaster

M2 Motor mit 4er Impulsrad
C2 Taster als Impulszähler
I2 Endtaster

I3 Taster
I4 Farbsensor
I5 Phototransistor
I6 Photowiderstand
I7 Spursensor links
I8 Spursensor rechts

Anstelle eines einfachen Motors mit Impulstaster kann auch ein Encodermotor angeschlossen werden. Dabei ist auf die deutlich größere Impulszahl zu achten.

Sensoranschlüsse

I1 - I8 Universalanschlüsse

Beim TestPanel müssen sie entsprechend konfiguriert werden, bei FishFace gibt es pro Anschlußtyp eine entsprechende Methode. Bei den einzelnen Sensortypen werden die von fischertechnik angebotenen Sensoren aufgeführt, weitere dürften möglich sein.

D5K Digital 5 kOhm : **GetInput**
Taster, Reedkontakt, PhotoTransitor
Anschluß : 2polig an I-Eingang

D10V Digital 10 V : **GetTrack**
SpurSensor
Anschluß : rot an +9V, grün an Masse, gelb / blau an zwei verschiedene I-Eingänge

A5K Analog 5kOhm : **GetAnalog**
NTC, Photowiderstand, Potentiometer
Anschluß : 2polig an I-Eingang

A10V Analog 10 V : **GetVoltage**
FarbSensor, Spannung allgem.
Anschluß : rot an +9V, grün an Masse, schwarz an I-Eingang

Dist cm : **GetDistance**
DistanceSensor (Meßbereich 3 - 400 cm)
Anschluß : rot an +9V, grün an Masse, schwarz an I-Eingang

C1 - C4 Zählereingänge

Zählereingänge : Schnelle Zähler für den EncoderMotor, können aber auch genauso gut mit normalen Motoren und der Kombination Impulsrad/Taster genutzt werden. Außerdem können sie wie D5K-Eingänge genutzt werden.

Aktorenanschlüsse

M1 - M4 : Motor, Lampe, Magnetventil, Elektromagnet, Summer, EncoderMotor. Power einstellbar im Bereich 0 - 512

Alternativ

O1 - O8 : Einpolig + Masse, einzelne M-Eingänge können als zwei O-Eingänge genutzt werden. Anschließbare Devices wie M-Eingänge, bei Motoren ist dann aber kein Drehrichtungswechsel möglich.

Literatur zu C#

- Bernard Volz : Einstieg in Visual C# 2008 (mit DVD Visual Studio Express)
  Galileo ISBN 978-3-8362-1191-8 (24,90€) Guter Einstieg auch für Anfänger.
- Frischalowski : Visual C# 2008 Einstieg für Anspruchvolle
  (mit DVD Visual C# 2008 Express)
  Addison-Wesley ISBN 978-3-8273-2577-8 (29,95€)
- Eric Gunnarson : C#, Galileo, zweite Auflage, ISBN 3-89842-183-X (deutsch) als fundierte Einführung
- O'Reillys Taschenbibliothek : "C# 3.0 kurz & gut", als Referenz neben der recht ansprechenden Hilfe des Visual Studio.NET ISBN 978-3-89721-544-3

Und dann gibt es jetzt auch von den altbekannten VB-Autoren eine C# Version :

- Frank Eller : Visual C# 2005 - Grundlagen, Programmiertechniken, Datenbanken - Addison-Wesley ISBN 978-3-8273-2288-3.
- Doberenz / Gewinnus : Visual C# 2005 - für Profis,
  Hanser ISBN-13 978-3-446-406553-7.
  Außerdem das Kochbuch dazu. C# 2008 Ausgaben sind ebenfalls verfügbar.

Die Doberenz Bücher sind wirklich für Profis gedacht.

Die StartAmpel

So geht's los :

- TX Controller anschließen, dessen Funktion mit dem ROBO TX Test kontrollieren (COM-Namen merken)
- An den TX Controller anschließen M3 : grüne, M2 gelbe, M1 rote Lampe
- Console-Projekt StartAmpel aus Verzeichnis ???? öffnen
- In der Projektübersicht Verweis (Referenz / Verweise) FishFaceTX.DLL (Assemblies) ggf. korrigieren.
- Strg+F5 : Starten.

Code des Console-Programms :

```
class Program {
  static FishFace tx = new FishFace();
  static void Main(string[] args) {
    try {
// --- Achtung hier eigenen COMxx eintragen ---
      tx.OpenController("COM4");
      Console.WriteLine("--- Gestartet, Ende : ESC-Taste ---");
      do {
          tx.SetMotor(Mot.M1, Dir.On);
          tx.Pause(1000);
          tx.SetMotor(Mot.M2, Dir.On);
          tx.Pause(500);
          tx.SetMotor(Mot.M1, Dir.Off);
          tx.SetMotor(Mot.M2, Dir.Off);
          tx.SetMotor(Mot.M3, Dir.On);
          tx.Pause(2000);
          tx.SetMotor(Mot.M3, Dir.Off);
      } while (!tx.Finish());
    }
    catch (FishFaceException eft) {
      Console.WriteLine(etx.Message);
    }
    finally {
      Console.WriteLine("--- Beendet ---");
      tx.CloseController();
    }
  }
}
```

Das Programm steht im Verzeichnis StartAmpel.

Zu den Elementen :

- Das (Console) Programm befindet sich in der Klasse Program
- `static FishFace tx = new FishFace();` : anlegen einer neuen Instanz der Klasse FishFace (Teil von FishFaceTX.DLL) mit dem Name tx. Unter diesem Namen werden dann die Methoden (Funktionen) der Klasse FishFace angesprochen.
- Es gibt in der class Program nur eine einzige (statistische) Methode : Main, mit der die Anwendung auch gleich gestartet wird. Sie enthält ein try – catch – finally Block um eventuelle Fehler, die durch das Interface ausgelöst wurden, abzufangen. Der Hauptteil des Programms befindet sich im try-Teil.
- `tx.OpenController("COMxx");` : Herstellen einer Verbindung zum TXcontroller (siehe ROBO TX Test : Linke untere Ecke).
- `tx.SetMotor(Mot.M1, Dir.On);` : Einschalten der roten Lampe
  Die Methoden von FishFace bieten meist einen Auswahlliste (enum) möglicher Parameterwerte. Hier aus der Aufzählung Mot und Dir. Es können aber auch eigene Konstanten angegeben werden (z.B. const Mot LampeRot = Mot.M1).
- `tx.Pause(1000);` : Das Programm wird für 1000 MilliSekunden (1 Sekunde) angehalten.
- `tx.SetMotor(Mot.M2, Dir.On);` die gelbe Lampe wird für 500 MilliSekunden zugeschaltet. und dann werden beide aus und die grüne Lampe an M3 wird für 2000 MilliSekunden angeschaltet.
- Das läuft dann in einer Endlos-Schleife, die durch die Esc-Taste abgebrochen werden kann.
- Danach und der Ordnung halber : `tx.CloseController();`, die Verbindung zum Interface gekappt.

Siehe auch www.ftcomputing.de/csecke.htm .

Beispiele

Dreipunktregelung

Eine Lampe an Out.O7 sitzt auf auf einem Schneckenantrieb mit Motor an Mot.M1. Sie soll den Photowiderstand an Unv.I4 mit einem vorgegeben SollWert so beleuchten, daß der Meßwert an Unv.I4 stets innerhalb vorgegebener Grenzen (LimitHigh / LimitLow) bleibt.


```
using FishFaceTX;
namespace Dreipunkt {
  public partial class DreiMain : Form {
    FishFace tx = new FishFace();
    Mot ReglerMotor = Mot.M1;
    Unv LichtSensor = Unv.I4;
    Out Lichtquelle = Out.O7;

    int SollWert; int LimitHigh; int LimitLow; int ActualValue;

    public DreiMain() {InitializeComponent();}
    private void numSollwert_ValueChanged(object sender,
                 EventArgs e) {
      SollWert    = (int)numSollwert.Value;
      LimitHigh   = SollWert + 100;
      LimitLow    = SollWert - 100;
```

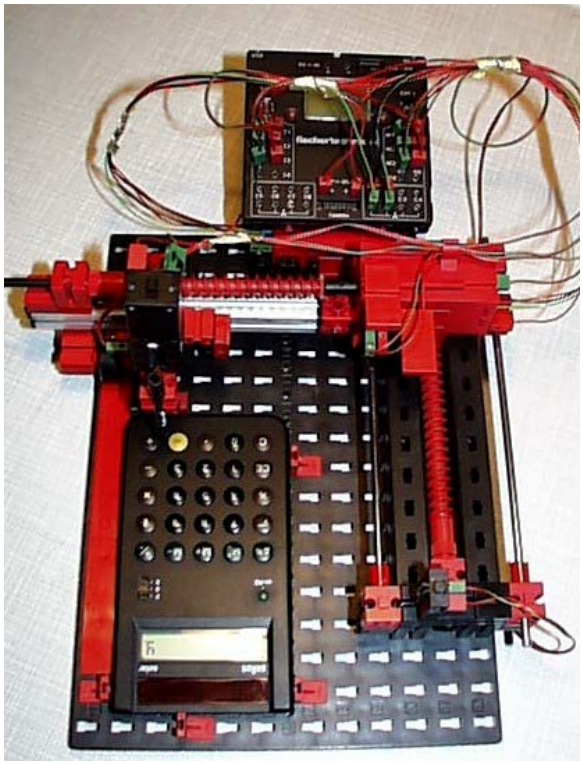
```
    }
    private void cmdAction_Click(object sender, EventArgs e) {
      numSollwert.Value = 700;
      tx.OpenController("COM4");
      tx.SetLamp(Lichtquelle, Dir.On);
      tx.Pause(1234);
      lblStatus.Text = "Ende : ESC-Taste";
      while (!tx.Finish()) {
        ActualValue = tx.GetAnalog(LichtSensor);
        lblIstWert.Text = ActualValue.ToString();
        if (ActualValue < LimitLow)
            tx.SetMotor(ReglerMotor, Dir.Right, 400);
        else if (ActualValue > LimitHigh)
                tx.SetMotor(ReglerMotor, Dir.Left,  400);
        else tx.SetMotor(ReglerMotor, Dir.Off);
        tx.Pause(111);
      }
      tx.CloseController();
      lblStatus.Text = "Das war's";
}}}
```

Der Sollwert wird durch das Control numSollwert (NumericUpDown) in der Ereignisroutine numSollwert_ValueChanged samt Grenzwerten eingestellt.

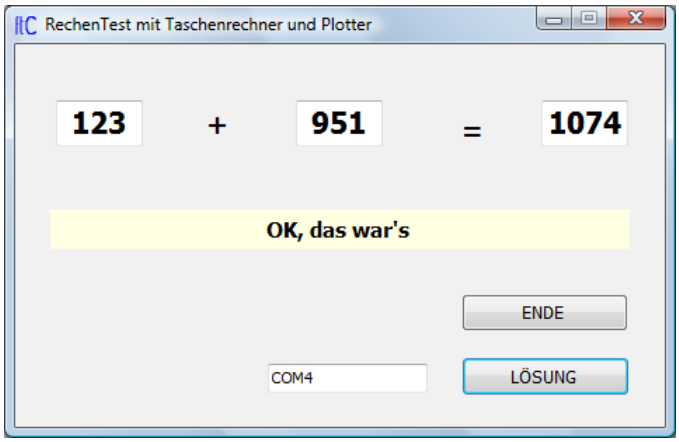
In der Click-Routine des Buttons cmdAction läuft in einer Endlosschleife das eigentliche Programm :
- tx.OpenController / tx.CloseController : Verbindung zum TX Controller (Name COM4 anpassen).
- die while-Schleife über die Messungen kann durch die ESC-Taste beendet werden.
- in der Schleife wird zuerst der aktuelle Wert am Photowiderstand gemessen und im Label-Control lblIstWert angezeigt.
- Danach erfolgt die eigentliche Regelung : akt. Wert kleiner LimitLow : Motor nach Rechts
  akt. Wert größer LimitHigh : Motor nach links, sonst Motor Stopp
- gemessen wird alle 111 mSek.

RobRechner

Ein plotterähnlicher Robot löst einfache Rechenaufgaben auf dem Taschenrechner



RechenPlotter : Die im Bild senkrechte Schnecke wird als X-Achse bezeichnet und durch einen Encodermotor an M1, C1 und I1 betrieben. Waagerecht Y-Achse mit Encodermotor (M2, C2, I2) An der Y-Schnecke hängt an einer Zahnstange mit Hubgetriebe ein Minimotor (M3 und I3) mit einem gefederten Taster. Der benutzte Taschenrechner muß mit Geschick auf der Bauplatte arretiert werden (Display nach oben ist eigentlich schöner). In jedem Fall müssen die Koordinaten für die Tasten 0 - 9 und "+", "=", "Clear" für den speziellen Aufbau mit Geschick ermittelt und in eine Tabelle eingetragen werden.



Auf START-Button drücken um den Rechenplotter auf Grundstellung zu fahren. Die Tastenbeschriftung wechselt dann auf Lösung. Eine Aufgabe eingeben (ohne Lösung) und auf Lösungsbutton Klicken. Der Rechenplotter marschiert los. Taschenrechnerergebnis mit hier angezeigten Ergebnis vergleichen.
Achtung COM4 ggf. anpassen.

Deklarationen

```
using FishFaceTX;
namespace RechenTest {
  public partial class frmMain : Form {
    FishFace tx = new FishFace();
    Mot xMotor  = Mot.M1;
    Mot yMotor  = Mot.M2;
    Mot taster  = Mot.M3;
    Unv tEnd    = Unv.I3;

    const int txFull = 512;
    const int xMax   = 1040; int xStart = 0; int xDest  = 0;
    const int yMax   = 1040; int yStart = 0; int yDest  = 0;
```

Tabelle mit Tastenpositionen

```
    private int[,] TasPos ={{940, 380},     // --- Taste 0
                            {820, 380},     // --- Taste 1
……….
                            {530, 780},     // --- Taste 9
                            {940, 1020},    // --- Plus
                            {940, 780},     // --- Gleich
                            {940, 200}      // --- Clear
                       };
public frmMain() {InitializeComponent();}
```

Taste betätigen

```
private void TasterGo() {
      // --- Taste drücken ---
      tx.SetMotor(taster, Dir.Right);
      tx.WaitForInput(tEnd);
      tx.SetMotor(taster, Dir.Left);
      tx.Pause(555);
      tx.SetMotor(taster, Dir.Off);}
```

Tastermotor fährt nach unten bis Endtaster (auf die Rechnertaste) und dann 555 mSek wieder nach oben um freizukommen.

Anfahren der Home-Position

```
    private void MoveHome() {
      xDest = 0;
      yDest = 0;
      tx.StartRobMotor(xMotor, Dir.Left, txFull, 9999);
      tx.StartRobMotor(yMotor, Dir.Left, txFull, 9999);
      tx.WaitForMotors(xMotor, yMotor);
      xStart = 0;
      yStart = 0;}
```

Hier mit Trick 17b : Die Motoren für X und Y werden nach links mit dem Ziel 9999 Schritte gestartet, das kann nie erreicht werden, der zugehörende Endtaster schlägt eher zu.

Anfahren einer Tastenposition

```
    private void MoveTo(int x, int y) {
      int diff, vorz = 1;
      xDest = x > xMax ? xMax : x;
      if (xDest > xStart) {
        tx.StartMotor(xMotor, Dir.Right, txFull, xDest - xStart);
        vorz = 1;}
      else if (xStart - xDest > 0) {
        tx.StartRobMotor(xMotor, Dir.Left, txFull, xStart - xDest);
        vorz = -1;}
      yDest = y > yMax ? yMax : y;
      if (yDest > yStart) {
        tx.StartMotor(yMotor, Dir.Right, txFull, yDest - yStart);
        vorz = 1;}
      else if (yStart - yDest > 0) {
        tx.StartRobMotor(yMotor, Dir.Left, txFull, yStart - yDest);
        vorz = -1;}
      diff   = tx.WaitForMotor(xMotor);
      xStart = xDest + diff * vorz;
      diff   = tx.WaitForMotor(yMotor);
      yStart = yDest - diff * vorz;
    }
```

Zunächst wird die maximal mögliche Position festgestellt und der Destwert ggf. gekürzt. Dann wird die zu fahrende Richtung bestimmt und aus aktueller (Start) Position und Dest-Position die Anzahl der zu fahrenden Schritte bestimmt und der Fahrauftrag gegeben. Die Motoren werden bei Erreichen des Ziels von einer zentralen Routine (MultiMediaTimer) gestoppt. Das WaitForMotor ist also nur eine Feststellung : Ziel erreicht. Eventl. Abweichungen vom Ziel werden in die aktuelle Position eingebracht.

Die Steuer-Routine

```
private void cmdAction_Click(object sender, EventArgs e) {
  int OP1, OP2; int i, ptrTas; int[] Tasten = new int[24];

  try {
    if (cmdAction.Text == "START") {
    // --- Auf StartPosition fahren ---
      tx.OpenController(txtCom.Text);
      cmdEnde.Text   = "HALT";
      lblStatus.Text = "Fährt auf Home-Position";
      TasterGo();
      MoveHome();
      MoveTo(TasPos[12, 0], TasPos[12, 1]);
      TasterGo();

      lblStatus.Text  = "Gestartet";
      cmdEnde.Text    = "ENDE";
      cmdAction.Text  = "LÖSUNG";
      cmdEnde.Focus();
    }
    else if (cmdAction.Text == "LÖSUNG") {
    // --- Lösung suchen ----
      OP1 = Convert.ToInt32(txtOP1.Text);
      OP2 = Convert.ToInt32(txtOP2.Text);
      ptrTas = 0; Tasten[ptrTas++] = 12;                // --- Clear
    // --- Ziffernpositionen für ersten Operanden
       for (i = 0; i < txtOP1.Text.Length; i++)
         Tasten[ptrTas++] =
                 Convert.ToInt32(txtOP1.Text.Substring(i, 1));
```

```
          Tasten[ptrTas++] = 10;                    // --- Addition
      // --- Ziffernpositionen für zweiten Operanden
         for (i = 0; i < txtOP2.Text.Length; i++)
           Tasten[ptrTas++] =
                   Convert.ToInt32(txtOP2.Text.Substring(i, 1));

           Tasten[ptrTas++] = 11;                    // --- Taste =

// --- Anfahren der erforderlichen Tasten ---
           cmdEnde.Text = "HALT"; lblStatus.Text =
                              "Kontrolle mit dem Taschenrechner";
           for (i = 0; i < ptrTas; i++) {
             MoveTo(TasPos[Tasten[i], 0], TasPos[Tasten[i], 1]);
             TasterGo();}
           // --- Anzeige korrektes Ergebnis ---
           txtErgebnis.Text = Convert.ToString(OP1 + OP2);
           lblStatus.Text = "OK, das war's";
           cmdEnde.Text = "ENDE";
         }
       }
       catch (FishFaceException txe) {
         MessageBox.Show(txe.Message, this.Text,
         MessageBoxButtons.OK, MessageBoxIcon.Stop);}
       catch (Exception ee) {lblStatus.Text = ee.Message;}
     } ……….
```

OpenController / CloseController wie gewohnt, hier in einer try - catch Konstruktion um evtl. Fehler von TX-Controller / Modell abzufangen.
Die größte Teil des Programms beschäftigt sich mit dem Ermitteln der anzufahrenden Tasten für die aktuelle Aufgabe.
Das tatsächliche Anfahren der Tasten geschieht in einer unscheinbaren, kleinen Schleife über die Methode MoveTo, die über die Liste der anzufahrenden Tasten (Tasten[]) auf die Tabelle der Tastenpositionen (TasPos[,]) zugreift.

Referenz

Programmrahmen

Anwendungen, die die FishFaceTX.DLL verwenden, enthalten eine Reihe von immer wiederkehrenden Elementen :

```
// ---(0)----
using FishFaceTX;

// ---(1)---

FishFace tx = new FishFace();

// ---(2)---

try  {
// ---(3)---
  tx.OpenController(COM-Name);
       .... Anwendungs-Code hier, bei größeren Anwendungen
            Methoden-Aufrufe .......
}
catch(FishFaceException txe) {
// ---(4)---
       ... Ausgabe von Fehlermeldungen ...
       .... txe.Message;
}
finally {
// ---(5)---
  tx.CloseController();
}
```

(0) Die FishFace-Funktionen befinden sich im Namespace FishFaceTX der FishFaceTX.DLL. Für die FishFaceTX.DLL ist ein entsprechender Projektverweis erforderlich.

(1) Die Klasse FishFace muß entsprechend installiert werden.

(2) Der try – Block fängt mögliche Fehler aus der Klasse FishFace ab (aber nur diese), man kann bei Bedarf weitere catch-Blöcke hinzufügen, wenn man das tx.CloseController direkt hinter den catch-Block stellt, geht es auch ohne finally.

(3) Mit OpenController wird die Verbindung zum Interface hergestellt (am einfachsten gibt man hier den eigenen Anschluß angeben fest ein). (5) tx.CloseController() hebt die Verbindung wieder auf.

(4) Im catch-Block können Fehlernachrichten ausgegeben werden, wenn FishFace-Methoden eine Exception ausgelöst haben.

Auf Basis dieses Programmrahmens kann man nette kleine Testprogramme erstellen, z.B. zum probieren mit den Beispielen der Referenz.

Verwendete Variablenbezeichnungen

Die Variablen sind durchweg enums. Einige Angabe erfolgen als int, das wird besonders gekennzeichnet

Die Parameter-Angaben erfolgen – soweit nicht extra notiert – By Value

ctrId	Bezeichnung des zutreffenden Controllers (Ctr)
Direction	Drehrichtung eines Motors (Dir)
CounterNr	Name eines C-Einganges (Cnt)
InputNr	Name eines (Universal)I-Einganges (Unv)
LampNr	Name eines O-Ausganges ("halben"-M-Ausganges) (Out)
MotorNr	Name eines M-Ausganges (Mot)
mSek	Zeitangabe in MilliSekunden
OnOff	Ein/Ausschalten eines M-Ausganges (Dir)
Value	allgemeiner Integer-Wert
bool	Wahrheitswert true/false
Power	Leuchtstärke (0 - 512)
Speed	Motorgeschwindigkeit (0 - 512)

enum's

Verwendung zur Eingaben von Parametern bei den FishFace-Methoden.

Dir	Angabe der Drehrichtung ...
Ctr	Angabe des zugehörenden Controllers (Main oder Ext1…, Main kann entfallen)
Unv	Angabe der Nummer eines I-Einganges
Cnt	Angabe der Nummer eines C-Einganges
Mot	Angabe der Nummer eines M-Ausganges
Out	Angabe der Nummer eines O-Ausganges

Exceptions

FisFaceException

Allgemeine Exception in Zusammenhang mit Zugriffen auf den TX Controller.
MessageAufbau : Verursachende Methode.Fehler

Klasse FishFace

Enthalten in der Assembly FishFaceTX.DLL (Source-File : FishFaceTX.CS).
FishFace ist die Basisklasse der Assembly FishFaceTX.DLL. Verwendet wird der Namespace FishFaceTX. Hinweis : Die Univeral-Eingänge erfordern eine Einstellung auf den Nutzungszweck (Analog, Digital, Widerstand, Spannung) das geschieht "fliegend" beim ersten Zugriff auf den einzelnen Unv-Eingang und wird von der Art der Zugriffsmethode abgeleitet werden.

Konstruktor

FishFace()
Ohne Parameter

Eigenschaften

bool	**NotHalt** Anmelden eines Abbruchwunsches (Default = false).
string	**Version** (get, static) Version der FishFaceTX.DLL

Methoden

ClearCounter

Löschen (0) des angegebenen Counters an einem C-Eingang

tx.**ClearCounter**([ctrId,] CounterNr)

Siehe auch : GetCounter

CloseController

Schließen der Verbindung zum Controller

tx.**CloseController**()

Siehe auch : OpenController

Finish

Feststellen eines Endewunsches (NotHalt, Escape [, Digitaler-Eingang])

bool = tx.**Finish**([[ctrId,]InputNr])

Exception : ControllerProblem, KeinOpen. DoEvents

Siehe auch : GetInput

Beispiel :

```
do {
  Console.WriteLine("läuft");
  tx.Pause(2345);
} while (!tx.Finish(Unv.I1));
```

Die do .. while-Schleife wird solange duchlaufen, bis entweder tx.NotHalt = true, die ESC-Taste gedrückt oder I1 = true wurde. Die Schleife wird mindestens einmal durchlaufen.

Alternativ :

```
while (tx.Finish(Unv.I1) == false) {
  Console.WriteLine("läuft");
  tx.Pause(2345);
}
Console.WriteLine("--- FINIS ---");
```

Die Schleife wird ggf. übersprungen.

GetAnalog

Feststellen eines Analogwertes an einem I-Eingang (NTC, Photowiderstand, Potentiometer).
Es wird der intern vorliegende (Raw)Wert ausgegeben.

Value = tx.**GetAnalog**([ctrId,] InputNr)

Exception : ControllerProblem, KeinOpen, DoEvent

Siehe auch : GetVoltage, GetDistance

Beispiel

```
Console.WriteLine(" NTC : " + tx.GetAnalog(Unv.I1).ToString());
```

WriteLine gibt den aktuellen Wert des angeschlossenen NTX am Universal-Eingang I1 aus.

GetCounter

Auslesen des Wertes des angegebenen Counters an einem C-Eingang

Value = tx.**GetCounter**([ctrId,] CounterNr)

Siehe auch : ClearCounter

Beispiel

```
Console.WriteLine("Counter für C2 : " +
tx.GetCounter(Cnt.C2).ToString());
```

Der aktuelle Zählerstand, der dem C-Eingang C2 zugeordnet ist, wird ausgegeben.

GetDistance

Auslesen eines zu einem I-Eingang (UltraschallSensor) gehörenden Distanzwertes [cm].

Value = tx.**GetDistance**([ctrId,] InputNr);

Exception : ControllerProblem, KeinOpen, DoEvent.

Siehe auch

Beispiel

```
int Abstand = tx.GetDistance(Dev.Ext1, Unv.I1);
```

Der aktuelle Abstand eines DistanceSensors am ersten Extension Controller in cm zu einem Hindernis wird bestimmt.

GetInput

Auslesen des Zustandes des angegebenen I-Einganges(Taster, Reedkontakt,
PhotoTransistor(auch Eingänge C1 - C4, als I9 - I12)

bool = tx.**GetUnvut**([ctrId,] InputNr)

Exception : ControllerProblem, KeinOpen. DoEvents

Siehe auch : Finish, WaitForInput

Beispiel

```
if (tx.GetInput(Unv.I1)) {
  ...
```

```
}
else {
  ...
}
```

Wenn der I-Eingang I1 (Taster, PhotoTransistor, Reedkontakt ...) = true ist, wird der erste Block durchlaufen. Bei `!tx.GetInput(Unv.I1)` wird der else-Zweig durchlaufen. Möglich ist auch `if(tx.GetInput(Unv.I1) == false)` { ... } oder `if(!tx.GetInput(Unv.I1)) {...}`

GetTrack

Auslesen des Zustandes des angegebenen I-Einganges(SpurSensor)

bool = tx.**GetTrack**([ctrId,] InputNr)
true bedeutet auf Spur (schwarzer Grund), false von der Spur (weißer Grund)

Exception : ControllerProblem, KeinOpen. DoEvents

Beispiel:

```
if (tx.GetTrack(Unv.I1)) {
  ...
}
else {
  ...
}
```

Wenn der I-Eingang I1 (einer der beiden Ausgänge eines SpurSensors) = true (auf Spur) ist, wird der erste Block durchlaufen. Bei `!tx.GetTrack(Unv.I1)` wird der else-Zweig (von Spur) durchlaufen.

GetVoltage

Feststellen des Spannungswertes des angegebenen I-Einganges.

Value = tx.**GetVoltage**([ctrId,] InputNr);

Exception : ControllerProblem, KeinOpen; DoEvents

Siehe auch : GetAnalog

Beispiel :

```
lblVolt.Text = tx.GetVoltage(Unv.I1).ToString();
```

Dem Label lblVolt wird der aktuelle Wert von I1 zugewiesen.

OpenController

Herstellen der Verbindung zum TX Controller. OpenController muß als erste Methode aufgerufen werden.

tx.**OpenInterface**(string COMname);

Exception : InterfaceProblem

Siehe auch : CloseController

Beispiel

```
try {
  tx.OpenController("COM4");
......
}
catch(FishFaceException txe) {
  Console.WriteLine(txe.Message);
}
finally  {
  tx.CloseController();
}
```

Herstellen der Verbindung zum TX-Controller am COM-Port COM4 (Ermittlung des richtigen COM-Ports über ROBO TX Test) ImFehlerfall wird (nach einigen Sekunden) der Text `'ControllerProblem.Open'` ausgegeben

Pause

Anhalten des Programmablauf für mSek MilliSekunden

tx.**Pause**(mSek)

Exception : KeinOpen; DoEvents; Abbrechbar

Beispiel

```
tx.SetMotor(Mot.M1, Dir.Left);
tx.Pause(1000);
tx.SetMotor(Mot.M1, Dir.Off);
```

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

SetLamp

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

tx.**SetLamp**([Dev,] Out, OnOff, Power)

- Power : Intensität des "Leuchtens", optional, default = 512.

Exception : ControllerProblem, KeinOpen

Siehe auch :

Beispiel

```
const Out Gruen = Out.O1, Gelb = Out.O2, Rot = Out.O3;

  tx.SetLamp(Gruen, Dir.On);
  tx.Pause(2000);
  tx.SetLamp(Gruen, Dir.Off);
  tx.SetLamp(Gelb, Dir.On);
```

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

SetMotor

Setzen eines M-Ausganges (Motor). Die Motordrehzahl kann gewählt werden (Default = 512 (Full)), ebenso die Fahrstrecke in Anzahl Impulsen.

tx.**SetMotor**([ctrId,] MotorNr, Direction [, Speed])

Speed default 512

Exception : ControllerProblem, KeinOpen;

Siehe auch : SetLamp

Beipiel 1

```
tx.SetMotor(Mot.M1, Dir.Right, 512);
tx.Pause(1000);
tx.SetMotor(Mot.M1, Dir.Left, 400);
tx.Pause(1000);
tx.SetMotor(Mot.M1, Dir.Off);
```

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

StartMotor

Starten eines "Encoder"Motors (echter oder normaler + Impulstaster) am M-Ausgang und zugehörenden C-Eingang. Der Motor läuft asynchron für die angegebene Anzahl von Zählimpulsen und wird dann selbsttätig abgeschaltet.

tx.**StartMotor**([ctrId,] MotorNr, Direction, Speed, ICount)

Beispiel

```
tx.StartMotor(Mot.M1, Dir.Left, 512, 123);
……
int diff = tx.WaitForMotor(Mot.M1);
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am C-Eingang C1 mit Geschwindigkeitsstufe Full(512) eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Schließlich wird auf das Erreichen von 123 Impulsen gewartet (der Motor kann dann schon abgeschaltet sein). Die Variable diff enthält eine evtl. Abweichung in Impulsen von der Zielvorgabe

StartRobMotor

Starten eines "Encoder"Motors (echter oder normaler + Impulstaster) am M-Ausgang und zugehörenden C-Eingang (gleiche Nummer). Der Motor läuft asynchron für die angegebene Anzahl von Zählimpulsen und wird dann selbsttätig abgeschaltet. Gegenüber StartMotor ist dem Motor ein Endtaster festzugeordnet (gleiche Nummer), der bei Dir.Left ausgewertet wird, Taster = true führt zum vorzeitigen Abbruch des Motors.

tx.**StartRobMotor**([ctrId,] MotorNr, Direction, Speed, ICount)

Beispiel

```
tx.StartMotor(Mot.M1, Dir.Left, 512, 9999);
tx.StartMotor(Mot.M2, Dir.Left, 512, 9999);
……
tx.WaitForMotors(Mot.M1, Mot.M2);
```

Die Motoren an M-Ausgang M1 / M2 werden für 9999 Impulse an C-Eingang C1 / C2 (also praktisch "endlos") mit Geschwindigkeitsstufe Full(512) eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Schließlich wird auf das Erreichen der Position Taster I1 / I2 gewartet (die Motoren können dann schon abgeschaltet sein). Genutzt wird das Zum Anfahren der "Home"Position eines Modells.

WaitForHigh

Warten auf einen false/true-Durchgang an einem digitalen I-Eingang

tx.**WaitForHigh**([ctrId,] InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForLow, WaitForInput.

Beispiel

```
tx.SetMotor(Mot.M1, Dir.On);
tx.SetMotor(Mot.M2, Dir.Left);
tx.WaitForHigh(Unv.I1);
tx.SetMotor(Mot.M2, Dir.Off);
```

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 digitale I-Eingang den vorgegebenen Wert annimmt.
(Default = true)

tx.**WaitForInput**([ctrId,] InputNr, OnOff)

OnOff ist optional, default = true

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar

Siehe auch : WaitForLow, WaitForHigh.

Beispiel

```
tx.SetMotor(Mot.M1, Dir.Left);
tx.WaitForInput(Unv.I1);
tx.SetMotor(Mot.M1, Dir.Off);
```

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

WaitForLow

Warten auf einen true/false-Durchgang an einem ditigalen I-Eingang

tx.**WaitForLow**([ctrId,] InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForInput, WaitForHigh.

Beispiel

```
tx.SetMotor(Mot.M1, Dir.On);
tx.SetMotor(Mot.M2, Dir.Left);
tx.WaitForLow(Unv.I1);
tx.SetMotor(Mot.M2, Dir.Off);
```

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).

WaitForMotor

Warten auf ein MotorReadyEreignis

tx.**WaitForMotor**([ctrId,] MotorNr)

MotorNr : Motor auf den gewartet werden soll. Zusätzlich wird eine evtl. Differenz zur Zielvorgabe zurückgegeben. Die kann dann in die Bestimmung der aktuellen Position einbezogen werden.

Exception : IControllerProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : StartMotor, StartRobMotor

Beispiel

```
int actPos = 112;
tx.SetMotor(Mot.M4, Dir.Right, 400, 50);
int diff = tx.WaitForMotor(Mot.M4);
actPos = actPos + 50 + diff;
```

Der Motor am M-Ausgang M4 wird rechtsdrehend mit halber Geschwindigkeit für 50 Impulse gestartet. Der Motor beendet sich selber. Die aktuelle Position ergibt sich aus der alten, der vorgegebenen Fahrstrecke und einer evtl. Überschreitung der Fahrstrecke.

WaitForMotors

Warten auf ein MotorsReadyEreignis

tx.**WaitForMotors**([ctrId,] MotorNr, ...)

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

Exception : IControllerProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : StartMotor, StartRobMotor

Beispiel

```
tx.SetMotor(Mot.M4, Dir.Left, 400, 50);
fx.SetMotor(Mot.M3, Dir.Right, 512, 40);
tx.WaitForMotors(Mot.M4, Mot.M3);
```

Der Motor am M-Ausgang M4 wird linksdrehend mit halber Geschwindigkeit für 50 Impulse gestartet, der an M3 rechtsdrehen mit voller Geschwindigkeit für 40 Impulse. Wenn die Position erreicht ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet.

Allgemeine Anmerkungen

Die Methoden erwarten ein vorhergehendes OpenController. Ggf. wird eine entsprechende Exception ausgelöst. Sie enthalten meist ein **DoEvents** um das Programm unterbrechbar zu machen. Wird im Ablauf ein InterfaceProblem festgestellt, wird eine entsprechende **Exception** ausgelöst.

Das Schalten von M- bzw. O-Ausgängen über SetMotor / SetLamp gilt stets bis zum nächsten SetMotor/SetLamp. Also SetLamp(Out.O1, Dir.On) … SetLamp(Out.O1, Dir.Off), entsprechend für SetMotor.

Die StartMotor/StartRobMotor -Methoden sind **asynchron** d.h. der oder die angesprochenen Motoren werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter und beenden sich nach Erreichen der vorgegebenen Position selber.

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

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

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

Anmerkungen zu C#

Programmrahmen

Aufbau eines ftComputing-Programms

1. try – catch(FishFaceException e) (- finally) Block
   sollte die gesamte Anwendung umfassen. Wenn mehr Detailierung erforderlich ist, natürlich mehr, ggf. auch weitere catch – Klauseln.
2. OpenController – CloseController
   umschließt die Anwendung. Häufig reicht der feste "COMxx" Eintrag für den einzigen ROBO TX Controller an USB. Bei Betrieb über Bluetooth ist ein anderer COM-Name fällig. Hilfestellung bei der Namensfindung bieten RoboTxText oder das mitgelieferte RoboTXdevs.
3. Programmabbruch bei Fehlfunktionen, das Modell kann sonst "gegen die Wand fahren" Die Wait.. Methoden können durch die ESC-Taste am Keyboard abgebrochen werden. Das gleiche tut die Eigenschaft NotHalt = true, der man auf der Form eine Button zuordnen sollte.
4. Sperren von Buttons und des (x) rechts oben um ein Programmende erst nach Beenden aller Interna zu erlauben.
5. `do { ... } while(!tx.Finish());`
   "Endlos"-Schleife, die durch Esc-Taste und NotHalt = true abgebrochen wird, ist nützlich bei Anwendungen, die auf Taste am Interface warten oder einen Funktions-Block wiederholen. Ein in Finish eingebautes Application.DoEvents sorgt für die Unterbrechbarkeit der Bedieneroberfläche.

Anlegen eines Console-Programmes

Beschrieben wird der Ablauf für C# 2005 :

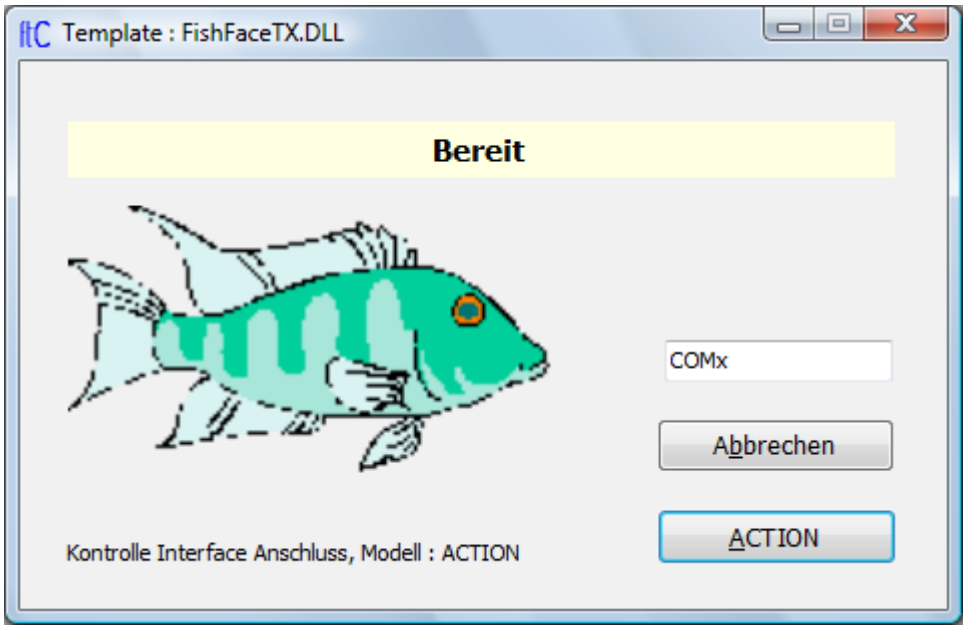
1. Menü Datei | Neues Projekt... | Konsolenanwendung
2. Projektnamen wählen : FishTXConsole
3. Ggf. Program.cs umbenennen
   Projektmappen Explorer : FishTXConsole
4. Projektmappen Explorer : Verweise
   Hinzufügen : FishFaceTX.DLL (über Tab Durchsuchen oder Aktuell)
5. In FishTXConsole.cs
   `using FishFaceTX;` ergänzen
6. Speichern
   Projektmappenverzeichnis anlegen, nicht erforderlich, wenn nur Einzelprojekt (man spart dann eine Verzeichnis-Ebene).

Anlegen eines Windows-Programmes

Beschrieben wird der Ablauf für C# 2005 Express :

1. Menü Datei | Neues Projekt .. | Windows Anwendung auswählen
2. Projektnamen wählen : BeispielFishFace
3. Projektmappen Explorer : Form1.cs umbenennen im Feld
   Eigenschaften Name : BeispielFishFace.cs
   Achtung : cs muß klein geschrieben werden.
4. Projektmappen Explorer : Verweise
   Hinzufügen : FishFaceTX.DLL (über Tab Durchsuchen oder Aktuell)
5. In der Source :
   `using FishFaceTX;` ergänzen.
7. Speichern
   Projektmappenverzeichnis anlegen, nicht erforderlich, wenn nur Einzelprojekt (man spart dann eine Verzeichnis-Ebene).

Vorlagen



Vorlage : FishTXWindows

Das Paket FishFaceTX.zip enthält ein Verzeichnis Templates in dem sich eine Reihe von Projekten befinden, die sich gut als Programmrahmen für das Ausprobieren von Codeausschnitten dieses Handbuchs, aber auch für das Ausprobieren eigener Ideen eignen. Um sie als Vorlagen auf der eigenen C# 2005 Installation nutzen zu können, sind sie vorher noch als Vorlagen zu speichern. Sie erscheinen dann anschließend in dem Auswahlfenster für neue Projekte.

FishTXConsole

```
namespace FishTXConsole1 {
  class Program {
    static FishFace tx = new FishFace();
    static void Main(string[] args) {
      try {
        Console.WriteLine("--- Main gestartet ---");
        tx.OpenController("COM4");
        Console.WriteLine("--- Beenden : Escape-Taste ---");
        do {
          // ----------- Hier die Anwendungsbefehle einfügen ----
        } while (!tx.Finish());
      }
      catch (FishFaceException txe) {
        Console.WriteLine(txe.Message);
      }
      finally {
        tx.CloseController();
        Console.WriteLine("--- RETURN Taste drücken ---");
        Console.ReadLine();
      }
    }
  }
}
```

Ist eine ganz einfache Konsolen-Anwendung. Mit einem try – catch – finally Block zum Abfangen von FishFace-Fehlern und einem Console.WriteLine für Programmausgaben. Das OpenControlle auf eigenen Bedarf anpassen.

FishTXWindows

Für Anwendungen mit der Klasse FishFace. Über ein TextControl kann das gewünschte Interface ausgewählt werden. Für Status-Anzeigen ist lblStatus.Text vorgesehen.

Programm-Struktur

```
using FishFaceTX;

namespace FishTXWindows1 {
  public partial class frmMain : Form {
    FishFace tx = new FishFace();

    private void Action() {
      // --- Routine für den Modellbetrieb ---
      do {
        // --- Endlosschleife,
        //     Abbruch durch Halt-Button oder ESC-Taste,
      } while (!tx.Finish());
    }
```

Hier ist die enthaltene #region zusammengeklappt, das Programm sieht dann verblüffend übersichtlich aus.

Methode **Action** : nimmt die eigentliche Anwendung auf. Die do-Schleife kann auch gelöscht werden.

#region --- Programm-Kontrolle --- (jetzt aufgeklappt) : mit dem Steuerungs-Code für Start und Beenden der Anwendung.

Das Programm verfügt über folgende Controls :

- lblStatus : Label zur Anzeige des Programm-Status
- txtCOM : Zur Eingabe des aktuellen COM-Namens für tx.OpenController.
- cmdEnde : Button zu Abbrechen / Beenden des Programms. Die Beschriftung wechselt entsprechend.
- cmdAction : Button zum Start der Anwendung, während des Ablaufs der Anwendung disabled. Mit Click-Routine cmdAction_Click.
- lblHinweis : Label mit Bedienungshinweisen.

```
#region --- Programm-Kontrolle ---
public frmMain() {
  InitializeComponent();
}
private void cmdAction_Click(object sender, EventArgs e) {
// --- Open ROBO TX Controller an USB/über Bluetooth siehe COM-Name
//     während des Ablaufs von Action() gesperrt ---------------
  try {
    tx.OpenController(txtCom.Text);
    cmdAction.Enabled = false;
    cmdEnde.Text = "&HALT";
    lblStatus.Text = "--- Bei der Arbeit ---";
    lblHinweis.Text = "--- Ende : HALT-Button oder ESC-Taste ---";
    Action();        // --- Haupt-Routine der Anwendung
  }
  catch (FishFaceException txe) {
    MessageBox.Show(txe.Message, this.Text,
      MessageBoxButtons.OK, MessageBoxIcon.Stop);
  }
  finally {
    tx.CloseController();
    cmdAction.Enabled = true;
    cmdEnde.Text = "&ENDE";
    lblStatus.Text = "--- Auf ein Neues ---";
    lblHinweis.Text = "Nochmal : ACTION, sonst ENDE oder ESC-Taste";
    cmdAction.Focus();
  }
}

private void cmdEnde_Click(object sender, EventArgs e) {
// --- Programmende nur, wenn Action nicht läuft ---
  if (cmdEnde.Text == "&HALT") tx.NotHalt = true; else this.Close();
}
private void formClosing(object sender, FormClosingEventArgs e) {
// --- Wenn Action läuft, wird ein Programmabbruch über (x)
//     unterbunden ------
  if (cmdEnde.Text == "&HALT") e.Cancel = true;
}
#endregion
```

cmdAction enthält einen try – catch – finally Block, der die Fehler aus FishFaceTX abfängt. Das OpenController öffnet, nach Vorgabe von txtCom, den ROBO TX Controller. Nach erfolgreichem Open wird die eigentliche Anwendung in Action(); gestartet.

Bei Open-Fehlern und Fehlern in Action() wird der catch-Teil des Blocks aufgerufen , die Fehlernachricht wird angezeigt, cmdAction wird beendet.

finally schließt die Interface-Verbindung wieder und rückt die Buttons gerade.

```
private void cmdEnde_Click(object sender, EventArgs e) {
 if (cmdEnde.Text == "&HALT") tx.NotHalt = true; else this.Close();
}
```

Die Click-Routine zu cmdEnde löst bei Beschriftung mit &HALT ein NotHalt = true aus, d.h. die in Action() oft vorhandene do-Schleife "rauscht durch" d.h. alle Wait.. Methoden und das Finish der do-Schleife werden beendet, die Schleife und damit Action() ist zu Ende.

Bei einer anderen Beschriftung von cmdEnde (Abbrechen, ENDE) wird das gesamte Programm beendet.

Ein Betätigen der ESC-Taste hat die gleiche Wirkung wie die Betätigung von cmdEnde.

```
private void formClosing(object sender, FormClosingEventArgs e) {
  if (cmdEnde.Text == "&HALT") e.Cancel = true;
}
```

formClosing wird z.B. bei einem Click auf (x) rechts oben aufgerufen. Es läßt ein Programmende nur zu, wenn der cmdEnde-Button nicht mit &HALT beschriftet ist.

Erstellen Vorlagen

Am einfachsten ist es, die obenaufgezählten Beispielprogramme (Programmrahmen) in Vorlagen zu konvertieren :

1. Das Projekt wie normal laden
2. An den eigenen Geschmack anpassen, testen
3. Menü Datei | Volage Exportieren
   Name z.B. FishFaceProjekt
4. Details :
   Symbol vergeben
   Beschreibung eingeben
   Fertigstellen