ftComputing

# FishFaceTXdev

## Programmierung des fischertechnik ROBO TX Controllers mit C# 2005 / 2008 - Teil 2 : Device-Objekte

Ulrich Müller

# Inhaltsverzeichnis


**Konzept** **3**

Allgemeines 3
Begriffe 4
Die Komponenten des ROBO TX Controllers 4
Der TX Controller als Ganzes 4
Die angebotenen Sensoren und Aktoren 4
Struktur der Assembly FishFaceTX.DLL 6
Allgemeine Klassen 8
class FishControl 8
abstract class DeviceBase 8
Klassen für Sensoren 9
abstract AnalogInput : DeviceBase 9
abstract BinaryInput : DeviceBase 10
Klassen für Aktoren 11
DualOutput : DeviceBase 11
MonoOutput : DeviceBase 11
Komplexe Klassen (Combined Class) 12
Drive2NXT 12
EncoderMotor : Motor 12
LightBarrier 13
Lights 13
LimitedMotor 14
RobMotor : DualOutput 15
RobMotors 15
TrailSensor 16
TwinMotor 16
C# Notizen 17
Thread 17

**Tips & Tricks** **18**

ROBO TX Test Panel 19
Blinker / Schleife 22
Wechselblinker 22
Ampel 22
AmpelListe 22
Motorbetrieb mit Klasse Motor 23
Motorbetrieb mit EncoderMotor 23
Motorbetrieb mit zwei Endtastern 24
Motorbetrieb mit dem RobMotor 24
Industry Robot 25
Der gleichzeitige Betrieb von zwei Motoren. 25
Der gleichzeitige Betrieb von zwei EncoderMotoren. 25
Lichtschranke 26
Beispielprogramme 27
Dreipunktregelung 27
RobRechner 29




Dokumentname : FishFaTXCSdev.doc. Bild : Office|Drache.WMF
Druckdatum : 02.12.2009.

# Konzept

## Allgemeines

Die Assembly FishFaceTX.DLL enthält als Basis-Klasse die Klasse FishFace, die ein einfaches Ansprechen aller Ein- und Ausgänge eines TX Controllers erlaubt (siehe FishFaTXCS.PDF).

Hier werden die erweiterten Möglichkeiten der FishFaceTX.DLL um Operationen über Objekte, die an die Ein-/und Ausgänge des TX Controllers angeschlossen, sind beschrieben.

FishFaceTX bietet ein Ebenen-Konzept :

1. **Kommunikation mit dem Robo TX Controller** über die ***ftMscLib.DLL*** von fischertechnik. Diese Ebene ist von außen nicht zugänglich.
2. Darauf aufsetzend werden die Funktion von ftMscLib.DLL in der ***umFish50.DLL*** noch einmal **zu einfachen Funktionen zusammengefaßt**, die leicht in einer Anwendung genutzt werden können. Diese Ebene ist noch weitgehend sprachunabhängig. Für den Zugriff aus einzelnen Programmiersprachen werden - sprachspezifisch - die notwendigen Deklarationen angeboten. Siehe umFish50.PDF.
3. Sprachspezifische Funktionen zum **kontrolliertenZugriff** auf den **Controller** als Ganzes und die **Einzelelemente** des Controllers. Dazu werden die Funktionen der Ebene 2 genutzt. Bestandteil der Klasse ***FishFace***.
4. **Komplexere Funktionen** für den Modellbetrieb (Wait…), die auf den Funktionen der Ebene 2 aufsetzen. Die Funktionen sind unterbrechbar und abbrechbar. Sie sind ebenfals Bestandteil der Klasse ***FishFace***
5. **Verwaltung und Kontrolle** der am TX Controller zur Verfügung stehenden Resourcen : Klasse ***FishControl***
6. **Basis Objekte** zum Umgang mit den am **Controller** angeschlossenen **Devices**. Das geschieht über Funktionen der Ebene 3. Die Objekte können zusätzlich zu den Eigenschaften und Methoden auch Ereignisse anbieten (Klassen ***PushButton, PhotoResitor, Lamp, Motor ….***
7. **Komplexe Objekte**, die mehrere Objekte der Ebene 6 und auch der eigenen Ebene zu einer Einheit zusammenfassen. z.B. Motor, (End)PushButton, CounterInput zum Komplexen Objekt ***RobMotor***, aber auch mehrere Komplexe Objekte RobMotor zu einem neuen RobMotor**S**. Die Objekte können in der Regel Ereignisse auslösen.

# Begriffe

**Controller** : fischertechnik Elektronikgeräte zum Betrieb von fischertechnik Modellen.
aktuell : ROBO TX Controller. Einschließlich der möglichen Erweiterungen (Extensions).

**Connector** : Steckkontakt am Controller zum Anschluß eines Devices

**Device** : Gerät, das an einen Connector des Controllers angeschlossen werden kann.

**FishFacexx.DLL** : Assembly für den Zugriff auf fischertechnik Controller.
aktuell : FishFaceTX.DLL.

**FishFaceTX** : Oberster Namensraum der FishFacexx.DLL Assembly

**FishFace** : die zentrale Klasse für den kontrollierten Zugriff auf den Controller.

# Die Komponenten des ROBO TX Controllers

## Der TX Controller als Ganzes

Klasse **FishFace** einschl. Messages

Klasse **FishControl** Verwaltung der DeviceObjekte

## Die angebotenen Sensoren und Aktoren

### Universal-Eingänge I1 - I8

**Digital-Werte D5K :** Klasse **D5KInput** : BinaryInput
Hier können auch die Zählereingänge C1.. C4 genutzt werden.

**Taster :** Klasse **PushButton** : D5KInput

**Photo Transistor :** Klasse **PhotoTransistor** : D5KInput

**ReedContact :** Klasse **ReedContact** : D5KInput

**Digital-Werte D10V :** Klasse **HalfTrack** : BinaryInput

**SpurSensor** : Ein Eingang (TrailSensor ist voller SpurSensor mit zwei I-Eingängen)

**Analog-Werte A5K** : Klasse **A5KInput** : AnalogInput

**Temperatursensor** : Klasse **NTC** : A5KInput

**Photowiderstand** : Klasse **PhotoResistor** : A5KInput

**Potentiometer** : Klasse **PotentioMeter** : A5KInput

**Analog-Werte A10V** : Klasse **A10VInput** : DeviceBase

**FarbSensor**: Klasse **ColorSensor** : A10VInput

**AbstandsSensor** : Klasse **DistanceSensor** : AnalogInput

## Counter-Eingänge C1 - C4

**Zähler** : Klasse **CounterInput** : DeviceBase

## Ausgänge M1 - M4

**M-Eingänge** allgemein : Klasse **DualOutput** : DeviceBase

**Motor** : Klasse **Motor** : DualOutput

## Ausgänge O1 - O8

**O-Ausgänge** allgemein : Klasse **MonoOutput** : DeviceBase

Anschluß an O und Masse oder zweipolig an M

**Lampe** : Klasse **Lamp** : MonoOutput

**Magnet** : Klasse **Magnet** : MonoOutput

**Pneumatic-Betätiger** : Klasse **PneuValve** : MonoOutput

**Hupe** : Klasse **Buzzer** : MonoOutput

## Combined Devices

Basis immer DeviceBase

Klasse : **Drive2NXT** (111) : Kopplung von zwei *EncoderMotoren*
(zum Antrieb von Mobile Robots)

Klasse : **EncoderMotor** (MotNr) : "normaler" Motor + Counter (vier Impulse pro Umdrehung)
alternativ : "echter" EncoderMotor (75 Impulse pro Umdrehung)

Klasse **LightBarrier** (103): Lichtschranke aus Lamp / PhotoTransistor

Klasse **Lights** (104): Kombination mehrerer *Lamp*

Klasse **LimitedMotor** (105): Motor und zwei Taster, die den Bewegungsraum des Motors angeben.

Klasse **RobMotor** (MotNr) : Motor, Counter, Endtaster zur Positionierung einer Baugruppe (meist über Schnecke). Motor siehe EncoderMotor. Betrieb asynchron.

Klasse **RobMotors** (102): Kombination mehrerer *RobMotor*

Klasse **TrailSensor** (108): Spursucher auf Basis des gleichnamigen Bauelements für den Anschluß an zwei I-Eingänge (*HalfTrack*)

Klasse **TwinMotor** (109) : Simultanbetrieb von zwei Motoren. Betrieb asynchron bis Stop
Die Drehzahl der beteiligten Motoren kann einzeln gesteuert werden.

# Struktur der Assembly FishFaceTX.DLL

Basis ist ftMscLib.DLL - umFish50.DLL

### FishFace

Klasse für den kontrollierten Zugriff auf den TX Controller, zusätzlich : komplexe Zugriffsmethoden (Kombination von Methoden für den direkten Controllerzugriff).

### FishControl

Verwaltung der zu einem TX-Controller gehörenden Devices, Instanzierung des TX Controllers und allgemeine Methoden für den TX Controller

### FishFaceException

Allgemeine Exception, die bei Fehlern und Fehlfunktionen ausgelöst wird

### Enumerationen

Primär zur Definition der Parameter von Eigenschaften und Methoden.
Ctr, Mot, Out, Unv, Cnt, Dir, AnalogState, TrailState

## Device-Klassen

**DeviceBase**

- **AnalogInput** Auslesen von Eingängen, die als Ergebnis einen numerischen Wert liefern
    - **A10VInput** Anschluß für analoge SpannungsSensoren.
        - **ColorSensor**
    - **A5KInput** Anschluß für WiderstandsSensoren
        - **NTC**
        - **PhotoResistor**
        - **PotentioMeter**
    - **DistanceSensor**
- **BinaryInput** Auslesen von Eingängen, die als Ergebnis true/false liefern
    - **D5KInput** Anschluß von WiderstandsSensoren
        - **PhotoTransistor**
        - **PushButton**
        - **ReedContact**
    - **HalfTrack** Anschluß für einen halben Spursensors
- **CounterInput** Zählereingang
- **DualOutput** Zweipoliger Ausgang
    - **Motor**
- **MonoOutput** Einpoliger (Pol + Masse) Ausgang
    - **Buzzer**
    - **Lamp**
    - **Magnet**
- --- Combined Devices ---
- **Drive2NXT** (2x EncoderMotor)
- **EncoderMotor** (Motor, CounterInput)
- **LightBarrier** (Lamp, PhotoTransitor)
- **Lights** (nx Lamp)
- **LimitedMotor** (Motor, 2x BinaryInput)
- **RobMotor** (Motor, D5KInput, CounterInput)
- **RobMotors** (nx RobMotor)
- **TrailSensor** (2x HalfTrack)
- **TwinMotors** (2x Motor)

# Allgemeine Klassen

## class FishControl

Verwaltung der Resourcen des zugeordneten Interfaces/Controllers.
Bereitstellung von Methoden, die den gesamten Controller betreffen.
Direkter Zugang zu den FishFace-Methoden.

Verwalten der DeviceObjekte in einer Liste - > Indexer, die sich über die Klasse DeviceBase hier anmelden.

**FishControl**()

Eigenschaften : **LibVersion** : der genutzten ftMscLib.DLL
**NotHalt** : Abbruchwunsch anmelden
**ControllerInstance** : Instance des aktuellen TX Controllers

Methoden : internal **AddDevice()**
**Connect()** : Verbindung zum TX Controller herstellen
**DisConnect()** : Verbindung zum TX Controller abbrechen
**StartEvents**() : Starten der aktivierten EventsLoops aller Devices
**Finish()** : Feststellen Abbruchwunsch
**Pause()** : Programmablauf anhalten.

Indexer : Liste der DeviceObjekte **DeviceBase**
Länge der Liste **DeviceCount**

## abstract class DeviceBase

Basis aller DeviceKlassen

**DeviceBase**(FishControl fc, bool WithEvents, int ConnectorNumber)

Ereignisse : abstract **EventLoop**

Eigenschaften : **ControllerNumber**¨: Nummer des TX Controllers (0 = Main … Ext)
**ControllerName** : Name des TX Controllers (Main,..Ext8)
**ConnectorNumber** : Ein-/Ausgangsnummer (1 - )
CombinedDevices : TypNr ab 100 oder MotNr
**ConnectorName** : Name des Connector (I1 … M4)
CombinedDevices : Klassenname
**DeviceType** : Typ des angeschlossenen Devices (A5K … D10V, Mot..)
**DeviceName** : Klassenname
**DeviceText** : Beschreibender Text
**ToString** : DeviceName + ControllerName + ConnectorName
**WithEvents** : Läuft der EventLoop
**ThreadInterval** : Interval (mSek) in dem der Thread aktiviert wird

Methoden : internal **StartThread()** : Starten des EventLoops
abstract **EventLoop()** : Die Routine des EventLoops

# Klassen für Sensoren

## abstract **AnalogInput :** DeviceBase

Für die Universal-Eingänge I1…I8

**AnalogInput**(FishControl fc, bool WithEvents, Ctr ctrId, Unv InputName)

| | |
|---|---|
| Ereignisse : | override **EventLoop**<br>**ChangedToLow**<br>**ChangedToHigh**<br>**ChangedToNormal**<br>**CurrentValue** |
| Eigenschaften: | **LimitHigh**<br>**LimitLow**<br>abstract **ActualValue**<br>**IsLow**<br>**IsHigh**<br>**IsNormal**<br>**ActualRange** |
| Methoden : | **WaitForLow()**<br>**WaitForHigh()** |

### **A5KInput** : AnalogInput

**PhotoResistor**

**NTC**

**PotentioMeter**

### **A10VInput** : AnalogInput

**ColorSensor**

### **DistanceSensor** : AnalogInput

## abstract **BinaryInput : DeviceBase**

Für Eingänge-Eingänge, die bool'sche Werte liefern

**BinaryInput**(FishControl fc, bool WithEvents, Ctr ctrId, Unv InputName)

| | |
|---|---|
| Ereignisse : | override **EventLoop**<br>**ChangedToTrue**<br>**ChangedToFalse** |
| Eigenschaften : | override **ConnectorName**<br>**IsFalse**<br>**IsTrue** |
| Methoden : | **WaitForTrue**()<br>**WaitForFalse**()<br>**WaitForHigh**()<br>**WaitForLow**() |

### **D5KInput** : BinaryInput

**PhotoTransistor** : D5KInput

**PushButton** : D5KInput

**ReedContact** : D5KInput

### **HalfTrack** : BinaryInput

# Klassen für Aktoren

## DualOutput : DeviceBase

Für allgemeine M-Ausgänge

**DualOutput**(FishControl fc, int ConnectorNumber)

Events : keine

Eigenschaften : override **ConnectorName**
**State**

Methoden : **On(**[Speed]**)**
**Left(**[Speed]**)**
**Right(**[Speed]**)**
**Off()**
**Go(**Dir [, Speed]**)**

### Abgeleitete Klassen

**Motor**

Zusätzliche Methoden : Forward, Backward, Up, Down (Basis Left, Right)

**RobMotor** (eigentlich Combined Class)

## MonoOutput : DeviceBase

Für allgemeine O-Ausgänge (Ox und Masse, Ox und Ox+1 anstelle von Masse (M-Ausgang)

**MonoOutput**(FishControl fc, int ConnectorNumber)

Eigenschaften : override **ConnectorName**
**State**

Methoden : **On()**
**Off()**
**Switch()**

### Abgeleitete Klassen

**Buzzer**

Zusätzliche Methode : On(int Duration, Speed Power)

**Lamp**

Zusätzliche Methode : BlinkingOn, BlinkingOff

**Magnet**

**PneuValve**

# **Komplexe Klassen** (Combined Class)

Abgeleitet von DeviceBase, nutzen mehrere DeviceObjekte

## Drive2NXT

Zwei EncoderMotoren im Verbund. Fahrweg über Impulse am schnelleren Motor. Konzept dem graphischen Objekt Bewegung von Lego Mindstorms NXT nachempfunden.

**Drive2NXT(**FishControl fc, Mot LeftMotor, Inp LeftImpulse,
Mot RightMotor, Inp RightImpulse**)**

Ereignisse : keine

Enum : Steering : TurnLeft, BackLeft, StopLeft, SlightLeft, Straight,
TurnRight, BackRight, StopRight, SlightRight

Eigenschaften : override **ConnectorName**
**LeftMotorNr**
**LeftMotorName**
**LeftImpulseSwitchNr**
**LeftImpulseSwitchName**
**RightMotorNr**
**RightMotorName**
**RightImpulseSwitchNr**
**RightImpulseSwitchName**

Methoden . **Forward(**Steering Direction**)**
**Forward(**Steering Direction, Speed speed**)**
**Forward(**Steering Direction, Speed speed, int Impulse**)**
**Backward(**Steering Direction**)**
**Backward(**Steering Direction, Speed speed**)**
**Backward(**Steering Direction, Speed speed, int Impulse**)**
**Stop()**

## EncoderMotor : Motor

Motor mit Impulsezähler

**EncoderMotor(**FishControl fc, [bool WithEvents,] Mot EMotor, Inp ImpulseCounter**)**

Ereignisse : **CountChanged**

Eigenschaften : override **ConnectorName**
override **ThreadInterval**
**ImpulseSwitchNr**
**ImpulseSwitchName**

Methoden : **Go(**Dir Direction, Speed speed, int Impulse**)**
**StartGo(**Dir Direction, Speed speed, int Impulse**)**
**Forward(**Speed speed, int Impulse**)**
**StartForward(**[Speed speed,] int Impulse**)**
**StartLeft(**Speed speed, int Impulse**)**
**Backward(**Speed speed, int Impulse**)**
**StartBackward(**[Speed speed,] int Impulse**)**
**StartRight(**Speed speed, int Impulse**)**
**StartUp(**Speed speed, int Impulse**)**
**Up(**Speed speed, int Impulse**)**
**StartDown(**Speed speed, int Impulse**)**
**Down(**Speed speed, int Impulse**)**
**WaitForDone()**

## LightBarrier

Lichtschranke mit PhotoTransistor und Lampe

ConnectorNumber 103

**LightBarrier**(FishControl fc, [bool WithEvents,] Out LightSource, Inp PhotoSensor)

| | |
|---|---|
| Ereignisse : | **ChangedToFree**<br>**ChangedToBroken** |
| Eigenschaften : | override **ConnectorName**<br>**PhotoNumber** / **PhotoName**<br>**LightNumber** / **LightName**<br>override **ThreadInterval**<br>**IsBroken**<br>**IsFree** |
| Methoden : | **On()**<br>**Off()**<br>**WaitForBroken()**<br>**WaitForFree()**<br>**WaitForPassed()** |

## Lights

Zusammenfassung von mehreren Deviceobjecten Lamp

ConnectorNumber 104

**Lights**(FishControl fc, params Out[] Lamps)

| | |
|---|---|
| Eigenschaften : | override **ConnectorName** = Lights |
| Methoden : | **Switch**([mSek,] params Dir[] SwitchList)<br>**SwitchOff()** |
| Indexer : | LampenListe |

## LimitedMotor

Motor mit Bewegungsraum zwischen zwei Endtastern

ConnectorNumber 105

**LimitedMotor**(FishControl fc, [bool WithEvents,] Mot OpMotor, Inp EndLeft, Inp EndRight)

| | |
|---|---|
| Ereignisse : | **LeftToTrue**<br>**RightToTrue** |
| Eigenschaften : | override **ConnectorName**<br>**MotorNumber**<br>**MotorName**<br>**ConNumberLeft**<br>**ConNameLeft**<br>**ConNumberRight**<br>**ConNameRight**<br>**IsLeft**<br>**IsRight**<br>**ActualState** |
| Methoden : | **StartLeft**([Speed])<br>**MoveLeft**([Speed])<br>**StartRight**([Speed])<br>**MoveRight**([Speed])<br>**WaitForDone**() |

## RobMotor : DualOutput

Verbund von Motor, Endtaster, Impulstaster mit dem Zweck Einzelpositionen eines Modells (z.B. Roboter) ansteuern zu können. Für die Einzelkomponenten werden intern Deviceobjekte angelegt.

ConnectorNumber Mx

**RobMotor**(FishControl fc, [bool WithEvents] Mot MotorName, int MaxPosition)

Ereignisse : **ChangedPosition**

Eigenschaften : **MotorNr**
**MotorName**
**EndSwitchNr**
**EndSwitchName**
**ImpulseSwitchNr**
**ImpulseSwitchName**
**MaxPosition**
**CurrentPosition**
**ActialPosition**
**State**

Methoden : **StartHome()**
**DriveHome()**
**StartDelta(**int Inc**)**
**DriveDelta(**int Inc**)**
**StartTo(**int Pos**)**
**DriveTo(**int Pos**)**
**WaitForDone()**

## RobMotors

Verbund von mehreren RobMotors, die dann sychron operieren können. Für die RobMotoren werden intern Deviceobjekte angelegt.

ConnectorNumber 102

**RobMotors**(FishControl fc, [bool WithEvents,] params RobData[] RMotList)

Ereignisse : **ChangedPosition**

Eigenschaften : **ConnectorName**
**State**
**ThreadInterval**

Methoden : **StartHome()**
**MoveHome()**
**StartTo(**params int[] DestPos**)**
**MoveTo(**params int[] DestPos**)**
**StartDelta(**params int[] DestDelta**)**
**MoveDelta(**params int[] DestDelta**)**
**WaitForDone()**

Indexer : Liste der RobMotoren

## TrailSensor

Spursensor mit Anschlüssen an zwei I-Eingängen

ConnectorNumber 108

**TrailSensor**(FishControl fc, [bool WithEvents] Inp InputLeft, Inp InputRight)

Ereignisse : **ChangedToLeft**
**ChangedToRight**
**ChangedToTrail**
**ChangedToOff**

Eigenschaften : **ConNumberLeft**
**ConNameLeft**
**ConNumberRight**
**ConNameRight**
override **ConnectorName**
**IsLeft**
**IsRight**
**IsOnTrail**
**IsOffTrail**
**TrailState**

Methoden : keine

## TwinMotor

Simultanbetrieb von zwei Motoren

Connector 109

**TwinMotor**(FishControl fc, Mot LeftMotor, Mot RightMotor)

Ereignisse : keine

Eigenschaften : override **ConnectorName**

Methoden : **Run(**Dir DirLeft, Speed SpeedLeft, Dir DirRight, Speed SpeedRight**)**
**Stop()**
**RunBackward(**Speed SpeedLeft, Speed SpeedRight**)**

# C# Notizen

## Thread

### Zugriff auf Anwendungsform aus Thread

```
Label.CheckForIllegalCrossThreadCalls = false;
```

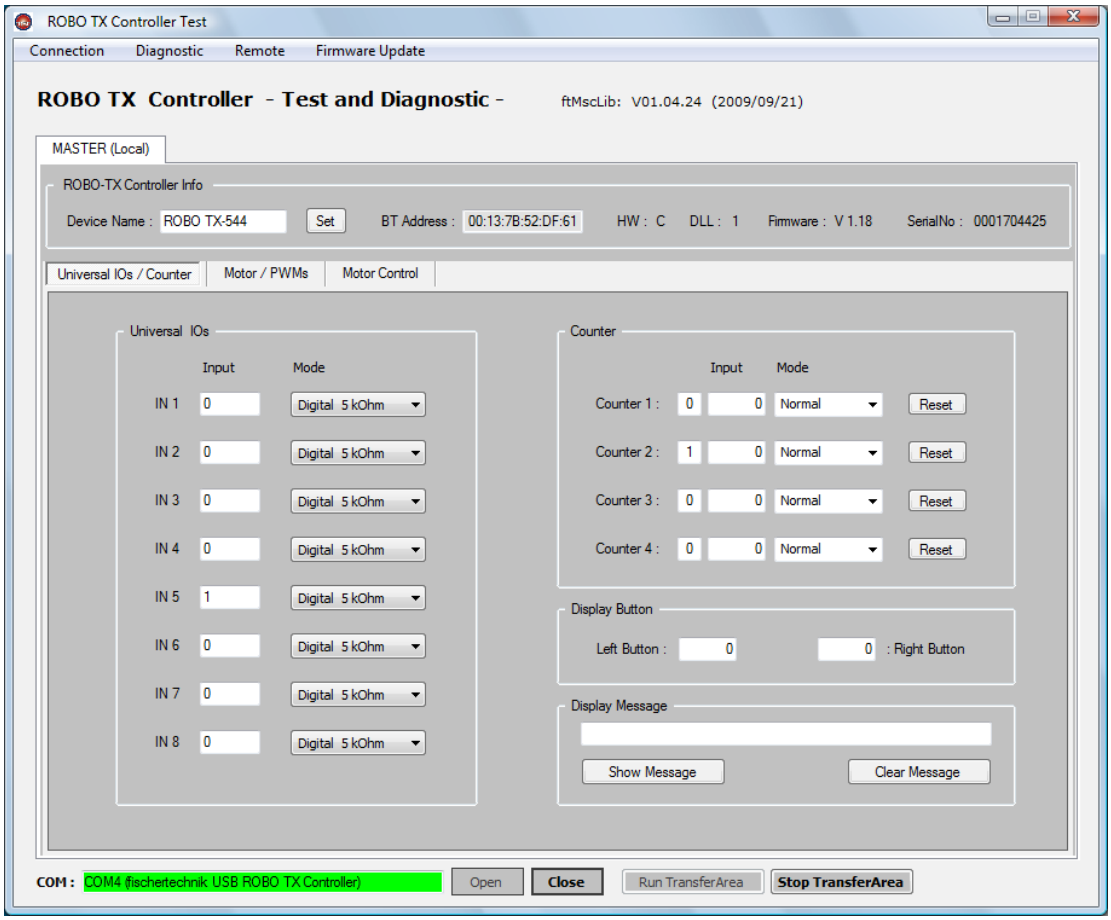
auf Ebene der nutzenden Form (z.B. Form_Load). Alternativ :

`Label.Invoke` (->FishDevices2005xxx TestMain)

```
private delegate void ListBoxText();
private void ToTrue(object sender) {
  lstAusgabe.Invoke(new ListBoxText(delegate() {
    lstAusgabe.Items.Add("I1 war hier zum " +
                                (++LoopCounter).ToString() + ". mal");
  }));
}
```

# Tips & Tricks

# ROBO TX Test Panel


ROBO TX Controller Test
Connection
Diagnostic
Remote
Firmware Update
ROBO TX Controller - Test and Diagnostic -
ftMscLib: V01.04.24 (2009/09/21)
MASTER (Local)
ROBO-TX Controller Info
Device Name : ROBO TX-544
Set
BT Address : 00:13:7B:52:DF:61
HW : C
DLL : 1
Firmware : V 1.18
SerialNo : 0001704425
Universal IOs / Counter
Motor / PWMs
Motor Control
Universal IOs
Input
Mode
IN 1 0 Digital 5 kOhm
IN 2 0 Digital 5 kOhm
IN 3 0 Digital 5 kOhm
IN 4 0 Digital 5 kOhm
IN 5 1 Digital 5 kOhm
IN 6 0 Digital 5 kOhm
IN 7 0 Digital 5 kOhm
IN 8 0 Digital 5 kOhm
Counter
Input
Mode
Counter 1 : 0 0 Normal Reset
Counter 2 : 1 0 Normal Reset
Counter 3 : 0 0 Normal Reset
Counter 4 : 0 0 Normal Reset
Display Button
Left Button : 0
0 : Right Button
Display Message
Show Message
Clear Message
COM : COM4 (fischertechnik USB ROBO TX Controller)
Open
Close
Run TransferArea
Stop TransferArea

RoboTxTest.EXE TestPanel aus dem ftMscLib.DLL-Paket. Stand alone Programm

Belegung für Testzwecke :

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

**M2** Motor mit 8 Schritten
**C2** Taster als Impulszähler
**I2** Endtaster

**O5** Lampe rot
**O6** Lampe gelb
**O7** Lampe grün

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

## Blinker / Schleife

Lampe an O5 wird gestartet für blinken 444 msec an und 333 msec aus. Anschließend laufen in einer while Schleife weitere Befehle. Die Schleife wird durch die ESC-Taste oder NotHalt = true; beendet. Das Blinken selber erfolgt in einem eigenen Thread. Zum Schluß noch Lampe ausschalten.

```
FishControl fc;
LampeRot Lampe;
fc = new FishControl();
LampeRot = new Lamp(fc, Ctr.Main, Out.O5);


void Blinker() {
  lblStatus.Text = "Blinker läuft";
  LampeRot.BlinkingOn(444, 333);
  while (!fc.Finish()) fc.Pause(123);
  LampeRot.BlinkingOff();
}
```

## Wechselblinker

Lampen an O5 und O6 blinken abwechselnd jeweils für 444 mSek. Der Code läuft im gleichen Thread.

```
Blinker = new Lights(fc, Out.O5, Out.O6);

      do {
        Blinker.Switch(444, Dir.On, Dir.Off);
        Blinker.Switch(444, Dir.Off, Dir.On);
      } while (!fc.Finish());
```

## Ampel

Lampen an O5 (rot), O6, O7, die wie eine einfache Verkehrsampel schalten.

```
Ampel = new Lights(fc, Out.O5, Out.O6, Out.O7);

      do {
        Ampel.Switch(4000, Dir.On,  Dir.Off, Dir.Off);
        Ampel.Switch(700,  Dir.On,  Dir.On,  Dir.Off);
        Ampel.Switch(4000, Dir.Off, Dir.Off, Dir.On);
        Ampel.Switch(700,  Dir.Off, Dir.On,  Dir.Off);
      } while (!fc.Finish());
```

## AmpelListe

Bei größeren Ampelanlagen kann eine Ampelsteuerung über eine Liste sinnvoll sein. Hier eine Liste mit den Lampenwerten pro Takt, die in einer for-Schleife abgearbeitet wird.

```
void AmpelListe() {
  Dir[,] Takte =                  { { Dir.On,  Dir.Off, Dir.Off },
     { Dir.On,  Dir.Off, Dir.Off }, { Dir.On,  Dir.Off, Dir.Off },
     { Dir.On,  Dir.On,  Dir.Off }, { Dir.Off, Dir.Off, Dir.On },
     { Dir.Off, Dir.Off, Dir.On },  { Dir.Off, Dir.Off, Dir.On },
     { Dir.Off, Dir.On,  Dir.Off }};

    lblStatus.Text = "Ampel nach Liste";
    while (!fc.Finish()) for (int i = 0; i < 8; i++)
      Ampel.Switch(700, Takte[i, 0], Takte[i, 1], Takte[i, 2]);}
```

## Motorbetrieb mit Klasse Motor

Einfacher Motor an M1:

```
LMotor = new Motor(fc, Ctr.Main, Mot.M1);

void MotorFahren() {
  lblStatus.Text = "Vorwärts mit halber Kraft 2222";
  LMotor.ForwardTime(333, 2222);
  lblStatus.Text = "Rückwärts voll 8888 oder I3";
  LMotor.BackwardTime(512, 8888, TasterDrei);
  lblStatus.Text = "Aufwärts, 5555 oder EndeM1";
  LMotor.StartUp(512, 5555);
  ……;
  LMotor.WaitForDone(EndeM1);    }
```

1. Motor fährt 2222 msec mit halber Geschwindigkeit Vorwärts :

2 .Motor fährt 8888 msec mit voller Geschwindigkeit zurück, wird vorher der Endtaster an I3 erreicht, wird schon dann gestoppt.

3. Motor wird in Richtung Up für min. 5555 msec gestartet. Er wird frühsten nach Ablauf der Zeit mit WaitForDone gestoppt. Alternativ wird er von WaitForDone bei Erreichen von Taster EndeM1 gestoppt.

## Motorbetrieb mit EncoderMotor

EncoderMotor an M2. Hier mit Impulstaster an C2.

```
RMotor = new EncoderMotor(fc, Ctr.Main, Mot.M2);
RMotor.DeviceText = "Mini + Impulstaster";
RMotor.CountChanged +=new EncoderMotor.Counter(EMotor_CountChanged);


  RMotor.Forward(512, 44);
  RMotor.Backward(444, 33);
  RMotor.StartDown(333, 22);
      ------;
  RMotor.WaitForDone();
```

Forward : Vorwärts mit Full-Speed für 44 Impulse.
Backward : Rückwärts mit Half-Speed für 33 Impulse.
Inbeiden Fällen wird nach Erreichen der vorgegebenen Impulszahl abgeschaltet.

StartDown : Abwärts im Schleichgang für 22 Impulse. Der Vorgang wird hier nur gestartet. Der Vorgang wird erst in WaitForDone - nach Erreichen der vorgegebenen Impulszahl - beendet.

Der akuelle Zählerstand (pro Methode ab 0) wird im Feld lblStatus angezeigt.

```
void RMotor_CountChanged(EncoderMotor sender, int count) {
  lblStatus.Text = sender.ConnectorName + " : " + count.ToString();
}
```

## Motorbetrieb mit zwei Endtastern

Einfache Schranke mit Antriebsmotor an M1,

```
Schranke = new LimitedMotor(fc, Ctr.Main, Mot.M1, Unv.I1, Unv.I2);
```

Endtaster Schranke geschlossen I2 (MoveRight), Endtaster Schranke geöffnet I1 (MoveLeft). Zusätzlich Taster zur Anforderung öffnen/schließen an I3. Der Status wird in lblStatus.Text angezeigt. Beispiel Parkhausschranke :

```
do {
  Schranke.MoveRight();
  lblStatus.Text = Taster.ConnectorName + " zum Öffnen";
  Taster.WaitForHigh();
  Schranke.MoveLeft();
  lblStatus.Text = Taster.DeviceText;
  Taster.WaitForHigh();
} while (!fc.Finish());
```

MoveLeft / MoveRight sind synchron, d.h. es wird gewartet, bis der Vorgang abgeschlossen ist. Alternative : StartLeft / StartRight und WaitForDone().

## Motorbetrieb mit dem RobMotor

Einem RobMotor sind im Gegensatz zum EncoderMotor zwei (I)-Eingänge (fest)zugeordnet. Einer dient als Impulszähler, wie beim EncoderMotor, der zweite zur Erkennung der Endstellung, die "linksdrehend" angefahren wird. Die zweite Endstellung wird durch die max. zul. Anzahl von Impulsen festgelegt. Eingesetzt können "echte" EncoderMotoren und eine Kombination von Motor und Impulstaster. Zu beachten ist dabei die deutlich unterschiedliche Impulszahl.

```
Turm = new RobMotor(fc, Ctr.Main, Mot.M1, 199);
```

```
void TurmFahren() {
  Turm.DriveHome();
  Turm.DriveTo(333);
  Turm.DriveTo(55);
}
```

Der RobMotor fährt zunächst auf seine Home-Postion am zugehörenden Endtaster an I1 und dann ozillierend auf Position 333 und 55. Man beachte die bei der Instanzierung angegeben Maximalposition 199. Tatsächlich fährt der Robot nur bis Position 199, der Rest wird gestrichen.

## Industry Robot

Zum Betrieb des Industry Robots wird die Klasse RobMotor**s** verwendet :

```
RobData[] ts = {new RobData(Mot.M1, 444),new RobData(Mot.M2, 666)};
TurmSaule    = new RobMotors(fc, Ctr.Main, ts);
```

Der Industry Robot nutzt hier die Motoren an M1 - M2 dazu gehören die Endtaster I1, I2, sowie die Impulseingänge C1 und C2. Das eigentliche Betriebsprogramm ist dann eher mickrig :

```
void TurmRob() {
  TurmSaule.MoveHome();
  TurmSaule.MoveTo(222, 111);
  TurmSaule.MoveTo(555, 111);
  TurmSaule.MoveTo(111, 333);
}
```

MoveHome : Ansteuern der Home-Position an den zugehörenden Endtastern
MoveTo : Ansteuern der Position Turm : 222, Arm vertikal : 111….

## Der gleichzeitige Betrieb von zwei Motoren.

Die Mobile Robots werden über zwei getrennte Fahrmotoren angetrieben, über die sie dann auch gelenkt werden. Hier wurde der Mobile Robot zusätzlich mit einem Spursensor bestückt.

```
Twins = new TwinMotor(fc, Ctr.Main, Mot.M1, Mot.M2);
```

```
void Duett() {
  Twins.Forward();
  fc.Pause(3233);
  Twins.Stop();
  Twins.TurnLeft(333);
  fc.Pause(1234);
  Twins.Stop();
}
```

Der Robot fährt mit beiden Motoren für 3233 msec Vor und dreht dann 333 msec nach Links

## Der gleichzeitige Betrieb von zwei EncoderMotoren.

Mobile Robots sind häufig mit Impulszählern an ein oder zwei Fahrmotoren ausgerüstet, das erlaubt

```
NXT2 = new Drive2NXT(fc, Ctr.Main, Mot.M1, Mot.M2);
```

```
void NXTFahren() {
  NXT2.Forward(Steering.SlightLeft, 512, 333);
  NXT2.Forward(Steering.Straight, 444, 333);
  NXT2.Backward(Steering.Straight, 333, 333);
  NXT2.StopRun();
}
```

Die Methode Forward / Backward gibt die allgemeine Fahrtrichtung vor, der Parameter Steering gibt die Feinsteuerung an, danach folgen Geschwindigkeit und Fahrweg in Impulsen.

## Lichtschranke

Lichtschranke bestehend aus einem PhotoTransistor und optional einer Lampe.

```
Lichtschranke = new LighBarrier(fc, Out.O0, Inp.I4);
```

Hier wurde keine Lampe angegeben, sie muß dann ggf. z.B. über die +/- Kontakte des Interfaces geschaltet werden.

```
void LSchranke() {
  LichtSchranke.On();
    do {
      LichtSchranke.WaitForFree();
      lblStatus.Text = "Lichtschranke frei";
      LichtSchranke.WaitForBroken();
      lblStatus.Text = "Lichtschranke unterbrochen";
    } while (!fc.Finish());
  }
```

Es wird im Wechsel auf eine freie und eine unterbrochene Lichtschranke gewartet.

# Beispielprogramme

## 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 {
    FishControl   fc = new FishControl();
    EncoderMotor  ReglerMotor;
    PhotoResistor LichtSensor;
    Lamp          Lichtquelle;

    public DreiMain() {
      InitializeComponent();
      Label.CheckForIllegalCrossThreadCalls = false;
      ReglerMotor = new EncoderMotor(fc, Ctr.Main, Mot.M1);
      LichtSensor = new PhotoResistor(fc, true, Ctr.Main, Unv.I4);
      LichtSensor.ChangedToHigh +=
                 new AnalogInput.Limit(LichtSensor_ChangedToHigh);
      LichtSensor.ChangedToLow +=
                 new AnalogInput.Limit(LichtSensor_ChangedToLow);
      LichtSensor.ChangedToNormal +=
                new AnalogInput.Limit(LichtSensor_ChangedToNormal);
      LichtSensor.CurrentValue +=
                  new AnalogInput.Limit(LichtSensor_CurrentValue);
      Lichtquelle = new Lamp(fc, Ctr.Main, Out.O7);
    }
```

Erzeugung der Device-Objekte. true steht hier für "mit Ereignissen" und die Zuordnung der Ereignisroutinen.

```
    void LichtSensor_ChangedToHigh(AnalogInput sender, int Wert) {
      ReglerMotor.RunLeft(444);}
    void LichtSensor_ChangedToLow(AnalogInput sender, int Wert) {
      ReglerMotor.RunRight(444);}
    void LichtSensor_ChangedToNormal(AnalogInput sender, int Wert) {
      ReglerMotor.StopRun();}
    void LichtSensor_CurrentValue(AnalogInput sender, int Wert) {
      lblIstWert.Text = Wert.ToString();}
    private void numSollwert_ValueChanged(object sender,
                                          EventArgs e) {
      int SollWert          = (int)numSollwert.Value;
      LichtSensor.LimitHigh = SollWert + 75;
      LichtSensor.LimitLow  = SollWert - 75;
    }
```

Die Ereignisroutinen sind überraschend kurz : nur ReglerMotor links, rechts und aus sowie Anzeige des aktuellen Meßwertes am Photowiderstand. bei der Routine zur Einstellung des Sollwertes werden zusätzlich noch die Grenzwerte für das Objekt Lichtsensor gesetzt. Auf eine threadsichere Anzeige der Werte wurde verzichtet. Siehe oben Label.CheckForIllegalCrossThreadCalls

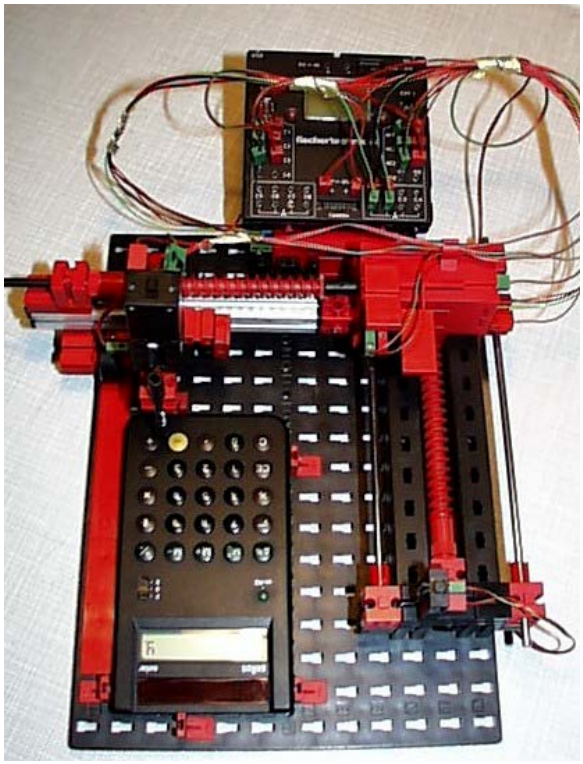
```
    private void cmdAction_Click(object sender, EventArgs e) {
      numSollwert.Value = 700;
      fc.Connect("COM4");
      fc.StartEvents();
      Lichtquelle.On();
      fc.Pause(1234);
      lblStatus.Text = "Ende : ESC-Taste";
      while (!fc.Finish()) fc.Pause(555);
      fc.DisConnect();
      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). Zusätzlich : Start der Eventroutinen.
- die while-Schleife über die Messungen kann durch die ESC-Taste beendet werden. Das ist auch ihre einzige Aufgabe : Das Warte auf das Ende. Der Rest läuft über die Ereignisroutinen.

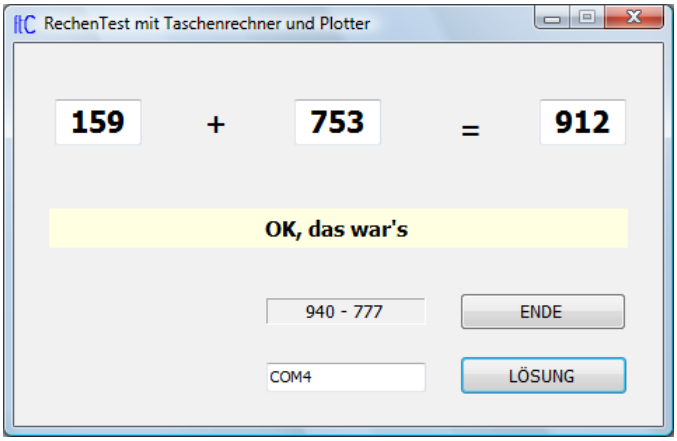
## RobRechner

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.

Siehe auch das gleiche Programm nur auf Basis der Klasse FishFace in Teil 1 (FishFaTXCS.PDF)




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. Neben der ENDE-Taste wird die aktuelle Position angezeigt.
Achtung COM4 ggf. anpassen.

## Deklarationen

```
using FishFaceTX;
namespace RechenTest {
  public partial class frmMain : Form {
    FishControl fc = new FishControl();
    RobMotors   Position;
    Betaetiger  TastenDruecker;
```

## Tabelle der 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
                     };
```

## Instanzierung der Devices

```
public frmMain() {
   InitializeComponent();
   RobData[] PosDaten = new RobData[] {
                  new RobData(Mot.M1,950),new RobData(Mot.M2,1050)};
   Position = new RobMotors(fc, true, Ctr.Main, PosDaten);
   Position.ChangedPosition+=new
                     RobMotors.PositionS(Position_ChangedPosition);
   TastenDruecker = new Betaetiger(fc, Ctr.Main,Mot.M3,Unv.I3, 555);
 }
```

## Die Ereignisroutine zur Anzeige der aktuellen Position

```
private delegate void LabelText();
private void Position_ChangedPosition(RobMotors sender, int[] Pos) {
  lblPosition.Invoke(new LabelText(delegate() {
  lblPosition.Text = Pos[0].ToString() + " - " + Pos[1].ToString();
      }));}
```

Hier mit synchronisierter Anzeige aus einem anderen Thread (dem bei der Instanzierung mit dem Parameter true angeforderten und in der Routine cmdAction_Click mit fc.StartEvents(); gestartetem).

## Neues Device Betaetiger abgeleitet von Motor

```
public class Betaetiger : Motor {
  private PushButton endTaster;
  private FishControl fc;
  private int upTime;
  public Betaetiger(FishControl fc, Ctr ctrId, Mot motName,
         Unv endTaster, int upTime) : base(fc, ctrId, motName) {
    this.fc = fc;
    this.upTime = upTime;
    this.endTaster = new PushButton(fc, ctrId, endTaster);
  }
  public void Druecke() {
    this.Down();
    endTaster.WaitForTrue();
    this.Up();
    fc.Pause(upTime);
    this.Off();
  }
}
```

Zusammenfassung von Motor und EndTaster mit der neuen Methode Druecke, die der Methode TasterGo der FishFace-Lösung entspricht.

## Die Steuerroutine

```
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 ---
      fc.Connect(txtCom.Text);
      fc.StartEvents();
      cmdEnde.Text = "HALT";

      lblStatus.Text = "Fährt auf Home-Position";
      TastenDruecker.Druecke();
      Position.StartHome();
      Position.WaitForDone();
      Position.MoveTo(TasPos[12, 0], TasPos[12, 1]);
      TastenDruecker.Druecke();

      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++) {
         Position.MoveTo(TasPos[Tasten[i], 0], TasPos[Tasten[i], 1]);
         TastenDruecker.Druecke();
       }
    // --- 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;}
}
```

…….

Sieht eigentlich wie die FishFace-Lösung aus. Grund : Anstelle bisherigen mit dem Programm erstellten Methoden TasterGo, MoveHome und MoveTo wurden die entsprechenden Methoden der Device-Klassen genutzt.