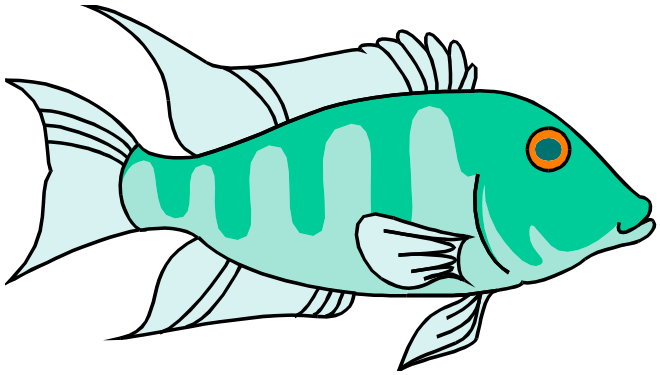
ftComputing

FishFace40 für C# 2005

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 : FishFa40CS2005.doc. Druckdatum : 18.05.2008
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish11.WMF

Einführung

Einleitung

Dies ist nach FishFa30 für C# und FishFace40 für C# nun die vierte Ausgabe der Dokumentation zur Assembly FishFaxx.DLL für C# und .NET. Der Funktionsumfang von FishFace ist weitgehend gleichgeblieben, in dieser Ausgabe werden die Möglichkeiten des des Ultraschallsensors unterstützt.

Geändert haben sich unterstützte Interfaces : FishFa30 mit Universal- und Intelligent Interface, FishFace40 mit Intelligent- und ROBO Interface und die unterstützte Sprache. Bei FishFa30 und FishFace40 ist es C# in der Version für .NET 1.0. FishFace2005 unterstützt jetzt C# in der Version für .NET 2.0. Es ist auch unter Vista und .NET 3.5 lauffähig.

Da die ROBO Serie insgesamt deutlich komplexer ist als das Intelligent Interface mit Extension, wird in besonderen Kapiteln auch auf die ROBO Hardware-Belange und Konfigurationsmöglichkeiten eingegangen.

Allgemeines

MIt der in C# geschriebenen Assembly FishFace2005.DLL (Namensraum FishFace40) wird die Möglichkeit geboten, die fischertechnik Interfaces unter einer .NET 2.0 Sprache zu programmieren (beschrieben wird hier der Einsatz von C# 2005). FishFace2005.DLL setzt auf umFish40.DLL (eine systemkonforme.DLL, die wiederum die von fischertechnik gestellte FtLib nutzt) auf. Die zentrale Klasse FishFace von FishFace40 erlaubt die Ansteuerung der Interfaces der ROBO Serie und des Intelligent Interfaces, jeweils ggf. mit Extensions. Es können mehrere Interfaces innerhalb eines Programmes simultan betrieben werden.

Von der Basisklasse FishFace abgeleitet sind die Klassen FishRobot für den Betrieb von Robot-Motoren und FishStep für den Betrieb von Schrittmotoren.

Angeboten werden Befehle zur Schaltung der M-Ausgänge und zur Abfrage der Eingänge eines Interfaces. Dazu wird das Interface in einem besonderen Thread(FtLib) von umFish40.DLL in regelmäßigen Abständen abgefragt. Zusätzlich werden die Veränderungen (Ein/Aus) an den I-Eingängen gezählt, sie werden außerdem zur Bestimmung der Schaltdauer der M-Ausgänge herangezogen. Dazu ist eine feste Zuordnung des an einen M-Ausgang angeschlossenen Motors und der an die I-Eingänge angeschlossenen Ende- und Impulstaster notwendig(RobMotoren). Die M-Ausgänge können außerdem mit verschiedener "Geschwindigkeit" betrieben werden, dazu werden sie in Intervallen ein- und ausgeschaltet (PWM).

Die A-Eingänge (Analog-Eingänge) liefern Raw-Werte im Bereich von 0 – 1023. Der Betrieb eines Intelligent Interfaces mit Auslesen der A-Eingänge benötigt dafür zusätzliche Zeit, deswegen werden die Analog-Eingänge beim Intelligent Interface nicht in jedem Zyklus abgefragt, es kann beim OpenInterface angegeben werden, nach wieviel Zyklen wieder Analogwerte ausgelesen werden sollen (Parameter AnalogZyklen).

Wenn das ROBO Interface mit einer RF-Platine ausgerüstet ist kann es auch am Funk-Betrieb teilnehmen.

Die mit FishFace40 erstellten Programme laufen im sog. "Online"-Betrieb, d.h. das auf dem Windows PC laufende Programm ist über USB bzw. COM mit dem Interface verbunden. Siehe auch Anmerkungen zu den Interfaces.

Getestet wurde mit : C# 2005 Express auf Windows 2000 mit .NET Framework v2.0, auf Windows XP mit .NET Framework v2.0 und unter Vista mit .NET 3.5

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 sollte das fischertechnik ROBO Pro installiert sein. Es ermöglicht die einfache Installation der Firmware-Updates von fischertechnik in den ROBO Interfaces, die Vergabe von Seriennummern, bringt die bei USB-Anschluß erforderlichen Treiber mit und ermöglicht natürlich auch die schnelle Erstellung einfacher Testprogramme.

Das Setup-Programm cs2005Fish40Setup.EXE
(www.ftcomputing.de/zip/cs2005fish40setup.exe)
enthält alles was man zum Arbeiten mit der Assembly FishFace2005.DLL (Namensraum FishFace40) benötigt.
{app} : gewählter Installationspfad (default : C:\Programme\ftComputing),
{sys} : Windows\System-Verzeichnis :

- {app} : dieses Dokument (FishFa40CS2005.PDF, auch über das Start-Menü erreichbar) und ein cs2005Fish40.TXT (ReadMe).
- {app} : Die Assembly FishFace2005.DLL und FishFace2005.XML (für ObjectBrowser)
- {app} : Das Interface Panel umFishDP40.EXE
- {app}\Templates40\CS2005:
  Die Programmrahmen für die Beispielsource des Handbuches
- {app}\Modelle40\CS2005 : Die Modellprogramme, begonnen mit StartAmpel
- {sys} : umFish40.DLL

Das Setup-Programm läuft, wie üblich, weitgehend automatisch. Installationspfad, anzulegende Desktop-Icons, Einträge ins Start-Menü können gewählt werden.

Hinweis : Es kann vorkommen, daß in den Projekten die Verweise auf FishFace2005.DLL nicht stimmen, nach dem Laden des Projektes im Projektmappen Explorer reparieren (Bei Standard Installation liegt FishFace2005.DLL im Verzeichnis ..\ftComputing. Wenn man FishFace2005.DLL in ein anderes Verzeichnis schieben will, ist FishFace2005.XML mit zu verschieben.

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

"Alte" Programme

Programm-Source früherer C# Versionen können nach einer Konvertierung durch CS2005 wieder verwendet werden.

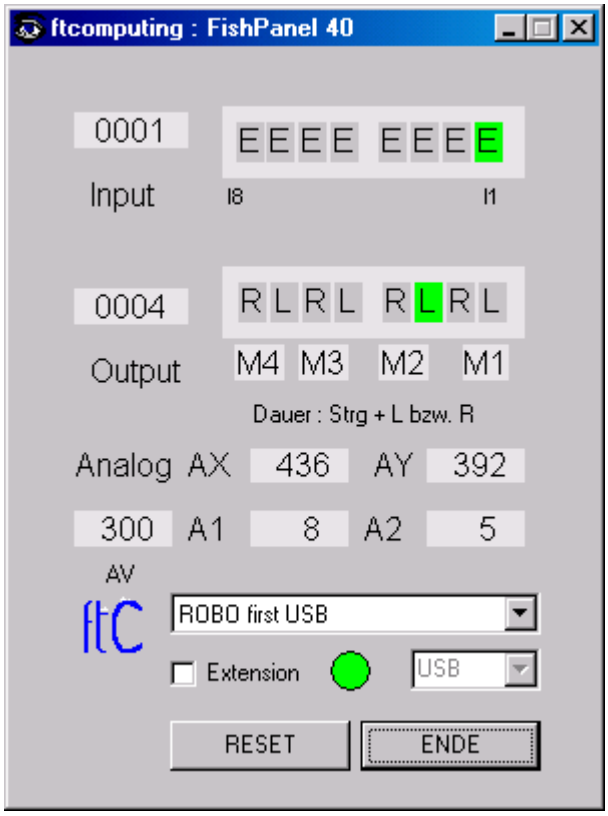
C# .NET 1.x Projekte werden beim ersten Laden in die CS2005 IDE nach .NET 2.0 konvertiert. Das geschieht in der Regel problemlos. Man sollte das entsprechende Protokoll aber beachten. Insbesondere sollte man den Verweis auf die (.NET 1.0) FishFace40.DLL in FishFace2005.DLL ändern. Ein Ablauf mit der alten Version ist zwar möglich, könnte dann aber nach der alten .NET-Runtime Version fragen.

Zu beachten ist, dass konvertiert 1.x Projekte nicht 1:1 den neuerstellten 2.0 Projekten entsprechen, die verwendeten Verzeichnisse und Dateien unterscheiden sich. Insbesondere werden Dateien mit Windows Forms nicht in solche mit partial class (zweite Datei mit dem von der IDE generierten Code) konvertiert. Bei kleinen Projekten ist manchmal ein Neuaufbau von Hand sinnvoll.

Die FishFace2005.DLL dagegen kann nicht zusammen mit alten Programmen eingesetzt werden, da sie nicht abwärtskompatibel ist (Source und Object).

Mit C# 2005 erstellte Programme laufen anstandslos auch unter Vista und .NET 3.5. Die Programmsources können problemlos von C# 2008 übernommen werden.

Interface Panel



Das Interface Panel dient zur Anzeige der Werte eines fischertechnik Interfaces und zum Schalten der M-Ausgänge (Output).

Nach Start des Panels kann eingestellt werden, ob mit Extension gearbeitet werden soll.

Über die obere ComboBox kann der Interface-Typ gewählt werden Bei Wahl eines Interfaces an COM kann zusätzlich noch der COMPort gewählt werden.

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

Die Input-Zeile zeigt den Status aller I-Eingänge an, links als Hexa-Wert.

Bei dem hier abgebildeten Interface Panel werden die D-Eingänge noch nicht unterstützt. Das FishPanel-Control von FishFace2005.DLL tut dies aber. Bei Bedarf also selber stricken.

Die Output-Zeile zeigt den Status der M-Ausgänge an, links wieder als Hexa-Wert. Ein Klick auf L bzw. R schaltet den entsprechenden Ausgang für die Dauer des Klicks ein, wird gleichzeitig die Strg-Taste gedrückt, auch dauerhaft. Das Ausschalten erfolgt dann durch Klick auf M1 ... Alle M-Ausgänge können durch Klick auf den RESET-Button gleichzeitig ausgeschaltet werden.

Legt gleichzeitig die Richtung Links (Dir.Links, Dir.Left) fest, also nach dem Modellaufbau testen in welche Richtung es beim L-Klick geht und bei der Programmierung dann berücksichtigen (bei Nichtgefallen : die Motoren umpolen). Analoges gilt für einen R-Klick.

Die Analog-Zeile zeigt die (dezimalen) Werte an die an den Eingängen AX und AY gemessen werden.

Literatur zu C#

- Microsoft : C# 2005 Express Edition. ISBN 3-86063-???-?
  Kurzer Überblick der Möglichkeiten von C# 2005. **CD mit der C# 2005 Express Edition** (19,90 €)
- Andreas Kühnel : Visual C#, ISBN 3-89842-662-9. Ein solide Einführung mit einem großen Themenspektrum. Auch für (beinahe) Anfänger geeignet.
- 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. ISBN 3-8273- 1856-4
- [dpunkt] Mössenböck : Software Entwicklung mit C# - Ein kompakter Lehrgang. dpunkt.verlag, ISBN 3-89864-126-0. Eher Kurzreferenz für erfahrene Programmierer mit Kenntnissen in anderen Sprachen. Neuauflage für C# 2005 unterwegs.
- O'Reilly : Programming C# 2. Auflage, ISBN 0-596-00309-9, als Übersicht.
- 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 ISBN 3-8273-2073-9. Neuausgabe für C# 2005 bereits verfügbar ISBN 3-8273-2288-X
- Doberenz / Kowalski : Visual C#.NET - Grundlagen und Profiwissen , Hanser ISBN 3-446-22021-6. Neuausgabe für C# 2005 ebenfalls verfügbar, aber nur sattelfesten C# Programmierern zu empfehlen, die die Besonderheiten von C# 2005 nutzen wollen.

In den beiden letztgenannten Büchern wird auch ausführlich auf die Programmierung mit Windows.Forms eingegangen.

Die StartAmpel

So geht's los :

- Interface anschließen, Funktion mit dem Interface Panel testen
- An das Interface anschließen M1 : grüne, M2 gelbe, M3 rote Lampe
- Console-Projekt StartAmpel aus Verzeichnis Modelle40 öffnen
- In der Projektübersicht Verweis (Referenz / Verweise) FishFace2005.DLL (Assemblies) ggf. korrigieren.
- F5 : Starten.

Code des Console-Programms :

```
class StartAmpel  {
   static FishFace ft = new FishFace();

   static void Main(string[] args) {
     try {
       ft.OpenInterface(IFTypen.ftROBO_first_USB, 0);
       do {
         ft.SetMotor(Out.M3, Dir.Ein);
         ft.Pause(1000);
         ft.SetMotor(Out.M2, Dir.Ein);
         ft.Pause(500);
         ft.SetMotor(Out.M3, Dir.Aus);
         ft.SetMotor(Out.M2, Dir.Aus);
         ft.SetMotor(Out.M1, Dir.Ein);
         ft.Pause(2000);
         ft.SetMotor(Out.M1, Dir.Aus);
       } while (!ft.Finish());
       ft.ClearMotors();
     }
     catch (FishFaceException eft) {
       cn.WriteLine(eft.Message);
     }
     finally {
       ft.CloseInterface();
     }
   }
}
```

Das Programm steht in StartAmpel.cs.

Hinzu kommen noch die Projekt-Datei StartAmpel.csproj, und die Dateien der Projektmappe : StartAmpel.sln und StartAmpel.suo. Sie werden alle automatisch angelegt.

Zu den Elementen :

- Das (Console) Programm befindet sich in der Klasse StartAmpel
- `static FishFace ft = new FishFace();` : anlegen einer neuen Instanz der Klasse FishFace (Teil von FishFace2005.DLL) mit dem Name ft. Unter diesem Namen werden dann die Methoden (Funktionen) der Klasse angesprochen.
- Es gibt hier 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.
- `ft.OpenInterface(IFTypen.ftROBO_first_USB, 0);` : Herstellen einer Verbindung zum Interface, hier dem ersten (und meist auch einzigen) ROBO Interface, das an USB gefunden wird.
- `ft.SetMotor(Out.M3, Dir.Ein);` : Einschalten der roten Lampe
  Die Methoden von FishFace bieten meist einen Auswahlliste (enum) möglicher Parameterwerte. Hier aus der Aufzählung Out und Dir. Es können aber auch einfache Zahlen oder eigene Konstanten angegeben werden.
- `ft.Pause(1000);` : Das Programm wird für 1000 MilliSekunden (1 Sekunde) angehalten.
- `ft.SetMotor(Out.M2, Dir.Ein);` die gelbe Lampe wird für 500 MilliSekunden zugeschaltet. und dann werden beide aus und die grüne Lampe an M1 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 : `ft.CloseInterface();`, die Verbindung zum Interface gekappt.

Das wars denn auch schon für den Anfang. Weiter kann es mit dem Durcharbeiten des Abschnitts Riesenrad gehen. Zu empfehlen, wenn die eigenen Programmierkünste noch nicht besonders ausgeprägt sind. Der Abschnitt Referenz beschreibt Eigenschaften und Methoden von FishFace im Detail. Im Abschnitt Tips & Tricks werden kleine Problemlösungen vorgestellt. Will man noch mehr Beispiele, sollte man sich auf www.ftcomputing.de/csecke.htm umsehen.

Beispiel Riesenrad

Allgemeines

Es wird die schrittweise Entwicklung eines Betriebsprogrammes für das Modell Riesenrad aus dem fischertechnik Kasten "Fun Park" (57 484, Anleitung allein 62 959) mit der Programmiersprache C# unter der Enwicklungsumgebung C# 2005 Express Edition, sowie der Assembly FishFace2005.DLL beschrieben.

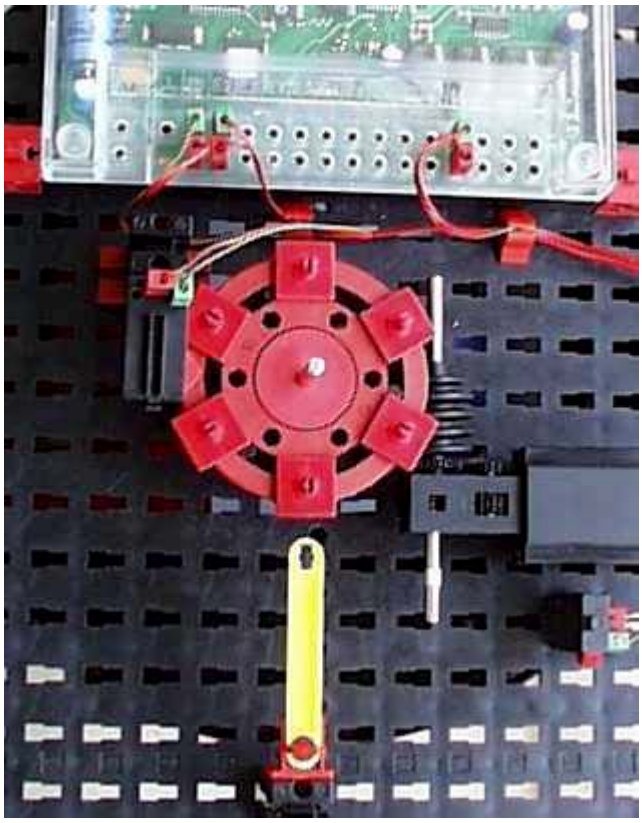
Der angebene Code für das Betriebsprogramm ist unabhängig von der Entwicklungsumgebung.

Das Modell

Das mit Motorantrieb ausgerüstete Modell entspricht weitgehend dem Original des Kastens Fun Park. Es wurde auf den Betrieb mit dem Intelligent Interface umgerüstet :

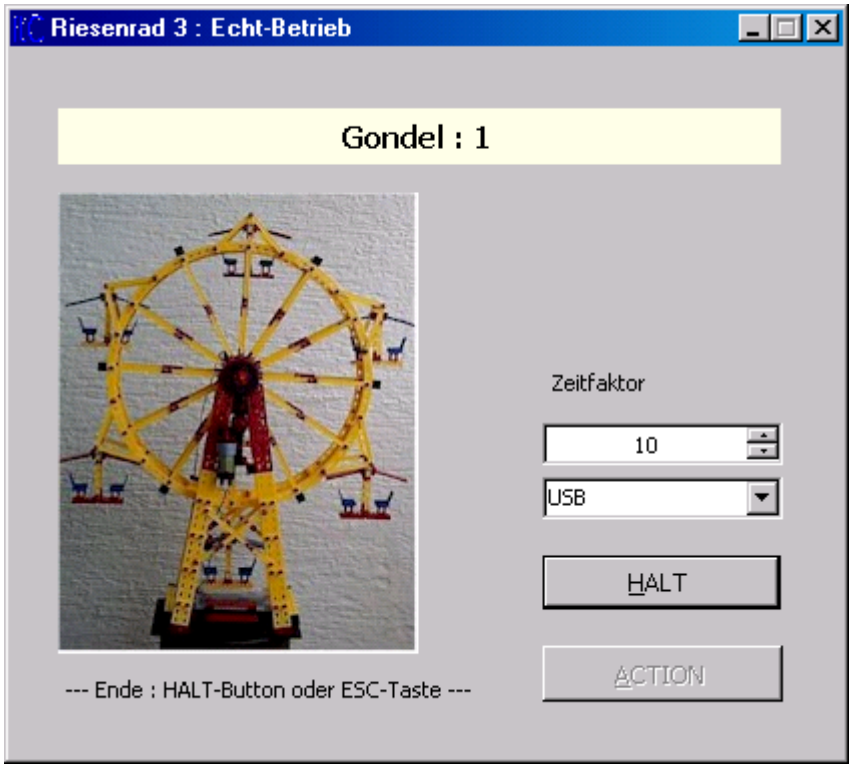
- Antriebsmotor an M1
- Im Fußbereich wurde ein Interface (wahlweise ROBO Serie oder Intelligent Interface) und der Taster I1 untergebracht.
- Am hinteren Scheibenrad wurden Schaltnocken (siehe Bild rechts) und der Taster I2 montiert.

Wenn der Wunsch der Einarbeitung in C# im Vordergrund steht, kann alternativ zum realen Modell kann aber auch ein Simulationsmodell eingesetzt werden. Es ist bei weitem nicht so sperrig :



Es kann die gleichen Steuerfunktionen ausführen wie das Original. Das Zeitverhalten entspricht nicht ganz dem großen Modell. Besonders beim Anfahren und Anhalten macht sich das bemerkbar. Die verwendete Zeitkonstante bewegt sich im Bereich 1500 (Riesenrad mit 12 Männeken / Simulations Modell) bis zu 3200 (leeres Riesenrad). Der gelbe Zeiger markiert die Position der unteren Gondel.

Programmrahmen



Das Windows Form Programm ist sehr einfach gehalten, hier der Aufbau von Schritt 3, die vorhergehenden Schritte enthalten teilweise noch weniger Elemente :

- lblStatus : Label Control zur Anzeige des aktuellen Status
- nudRunden bzw. nudFaktor : Spin Control zur Eingabe der Rundenzahl bzw. des Rundenfaktors.
- cmdAction : Button Control zum Start des Programms. Die zugehörende Ereignis-Routine cmdAction_Click enthält in einem `try – catch – finally` Block die erforderlichen `OpenInterface – CloseInterface`-Aufrufe. Es ruft nach `OpenInterface` das eigentliche Betriebsprogramm `Action();` auf.

Die fertigen Programme sind in der Projektmappe Riesenrad des Verzeichnisses Modelle40 zu finden.

Die Programme nutzen die Grundstruktur von \Templates40\CS2005\FishFaceWindows. Erläuterungen dazu siehe Abschnitt "Anmerkungen zu C#".

Schritt 1 : Aus- und Einsteigen

```
public partial class Riese1 : Form  {
  FishFace ft = new FishFace();

  private void Action() {
    lblStatus.Text = "--- Riese1 : gestartet ---";
    for (int i = 0; i < 6; i++) {
      ft.SetMotor(Out.M1, Dir.Links);
      ft.WaitForHigh(Inp.I2);
      ft.Pause(1500);
      ft.SetMotor(Out.M1, Dir.Aus);
      lblStatus.Text = "Gondel : " + i;
      ft.Pause(4000);
    }
    lblStatus.Text = "--- Das war Riese1 ---";
  }
```

Alle Gondeln werden nacheinander zur Einsteigeposition gefahren. Zum Aus- und Einsteigen wird eine feste Zeit gehalten.

Kernpunkt der Positionserkennung ist der Befehl WaitForHigh (Warten auf einen false/true-Durchgang). ein schlichtes Warten auf I2 = true reicht nicht, da der Taster noch auf true stehen kann. Da mit I2 = true die exakte Einsteigeposition nicht gewährleistet ist (Lage der Schaltnocken, Bremsverhalten des Modells), wird nach I2 = true noch ein paar (1,5 Sekunden) weitergefahren und dann erst abgeschaltet. Danach folgt die Pause für das Aus- und Einsteigen. Das ganze immer schön in einer 6er-Schleife.

ACHTUNG : Die Länge der Pause (hier Pause 1500) hängt beim Original Riesenrad von der Beladung und der Befestigung der Hauptachse ab. Sie lag bei mir zwischen 1500 (12 Männeken) und 3200(leer).

Schritt 2 : Die Fun-Runden

```
private void Action() {
 lblStatus.Text = "--- Riese2 : gestartet ---";
  while (!ft.Finish(Inp.I1)) {
    lblStatus.Text = "--- Ein- und Aussteigen ---";
    for (int i = 0; i < 6; i++) {
      ft.SetMotor(Out.M1, Dir.Links);
      ft.WaitForHigh(Inp.I2);
      ft.Pause(1500);
      ft.SetMotor(Out.M1, Dir.Aus);
      lblStatus.Text = "Gondel : " + i.ToString();
      ft.Pause(4000);
    }
    lblStatus.Text = "---Tour linksrum ---";
    ft.SetMotor(Out.M1, Dir.Links);
    ft.Pause(1000 * (int)nudFaktor.Value);
    ft.SetMotor(Out.M1, Dir.Aus);
    ft.Pause(1000);
    lblStatus.Text = "---Tour rechtsrum ---";
    ft.SetMotor(Out.M1, Dir.Rechts);
    ft.Pause(1000 * (int)nudFaktor.Value);
    ft.SetMotor(Out.M1, Dir.Aus);
    ft.Pause(1000);
  }
  lblStatus.Text = "--- Das war Riese2 ---";
}
```

Das ganze Programm wird in eine while Schleife gepackt, so erhält man ein schönes Demo-Programm, das durch I1 = true oder die ESC-Taste abgebrochen werden kann. Zusätzlich wird in lblStatus die aktuelle Funktion angezeigt.

Nach dem Aus-/Einsteigen (wie bisher) wird 10 Sekunden links und dann 10 rechts gedreht. Da es bei sofortiger Richtungsumschaltung richtig knirschen kann, wird dazwischen 1 Sekunde Pause eingelegt.

Wenn man die Runden gerne länger hätte, kann man im Spin Control einen Rundenfaktor eingeben (Vorgabe 10, Rundenzeit also 10 Sekunden).

Schritt 3a : Echt-Betrieb

```
    private void Action()
    {
      lblStatus.Text = "--- Riese3 : gestartet ---";
      while (!ft.Finish(Inp.I1))
      {
        lblStatus.Text = "--- Aus- und Einsteigen ---";
        for (int i = 0; i < 6; i++) {
          ft.SetMotor(mRadMotor, cLinks);
          ft.WaitForHigh(eRadPos);
          ft.Pause(1500);
          ft.SetMotor(mRadMotor, cAus);
          lblStatus.Text = "Gondel : " + i.ToString();
          ft.WaitForLow(eQuittung);
        }
        lblStatus.Text = "--- Fahrbetrieb : Quittungstaster ---";
        ft.Pause(3000);
        if (!ft.GetInput(eQuittung)) ft.NotHalt = true;
..... weiter wie gehabt -----
```

Bei einem Echt-Betrieb sind die genauen Zeiten für Aus- und Einsteigen nicht vorhersebar, die Pause(4000) durch eine WaitForLow(Inp.I1) ersetzt. Das Programm wartet bis I1 gedrückt und wieder freigegeben wird.

Nach dem Aus- und Einsteigen muß der Betrieb durch erneutes Drücken der I1-Taste innerhalb von 3 Sekunden freigegeben werden, sonst wird das Programm beendet : Feierabend. Zum Abbruch wurde hier `ft.NotHalt` auf `true` gesetzt. Das läßt alle FishFace-Methoden "durchrauschen". Bei ft.Finish führt das dann zum Beenden der while-Schleife. Eine bequeme Methode für den Abbruch in Notfälle (das Modell läuft "gegen die Wand") oder zum Beenden einer Endlosschleife. Hier hätte es auch ein schlichtes `break` getan.

Man kann sich zum Betrieb natürlich locker noch mehr einfallen lassen.

Schritt 3b : Symbolische Namen

```
// --- Globale Daten ----------------------------------------
private FishFace ft = new FishFace();
const int mRadMotor = (int)Out.M1, eRadPos = (int)Inp.I2,
          eQuittung = (int)Inp.I1,
          cLinks = (int)Dir.Links, cRechts = (int)Dir.Rechts,
          cAus = (int)Dir.Aus;
```

Anstelle der allgemeinen Bezeichnungen (enums) für die Ein- und Ausgänge des Interfaces sollte man, wenn's was größeres wird, besser spezielle symbolische Namen setzen. Hier mRadMotor, eRadPos, eQuittung. Da beim Aufruf der Methoden ein EntwederOder gilt, sind auch für die eigentlich passenden enums eigene symbolische Namen erforderlich : cLinks ...

(zusammen mit dem TypeCasting kann man sie allerdings weiterverwenden – s.o.). Sie können dann überall verwendet werden, wo jetzt die Enums stehen.

Aussehen tut das dann so (Ausschnitt) :

```
for(int i = 1; i <= 6; i++) {
   ft.SetMotor(mRadMotor, cLinks);
   ft.WaitForHigh(eRadPos);
   ft.Pause(1500);
   ft.SetMotor(mRadMotor, cAus);
   lblStatus.Text = "Gondel : " + i.ToString();
   ft.WaitForLow(eQuittung);
}
```

Schritt 4 : Zählen statt warten

```
lblStatus.Text = "Fahrbetrieb : Quittungs-Taster";
ft.Pause(3000);
if (!ft.GetInput(eQuittung)) ft.NotHalt = true;
   ft.SetMotor(mRadMotor, cLinks);
   for(int i = 1; i <= (int)nudRunden.Value; i++) {
      lblStatus.Text = "--- Runde : " + i.ToString() + " linksrum";
      ft.WaitForChange(eRadPos, 12);
   }
   ft.SetMotor(mRadMotor, cAus);
   ft.Pause(1000);
   ft.SetMotor(mRadMotor, cRechts);
   for(int i = 1; i <= (int)nudRunden.Value; i++) {
      lblStatus.Text = "--- Runde : " + i.ToString() + " rechtsrum;
      ft.WaitForChange(eRadPos, 12);
   }
   ft.SetMotor(mRadMotor, cAus);
   ft.Pause(1000);
}
lblStatus.Text = "--- Betrieb beendet ---";
```

Die Pause(1000 * nudFaktor.Value) wurde durch eine Schleifenkonstruktion ersetzt. Die Schleife selber enthält ein WaitForChange(eRadPos, 12), das ist genau eine Runde, da WaitForChange die Flanke d.h. den Übergang von true/false und false/true zählt. Die Schleife wird sooft duchlaufen, wie es der Spin Control nudRunden.Value (ex nudFaktor) angibt. Das bringt zum Einen eine angebbare Rundenzahl, zum Anderen die Möglichkeit, die aktuelle Rundennummer auch auszugeben.

Schritt 5 : Überkreuz Beladen

In der Praxis werden bei einem Riesenrad die Gondeln selten in ihrer direkten Reihenfolge "beladen". Das Beladen übernimmt hier ein gleichnamige Unterprogramm. Das Unterprogramm selber entspricht weitgehend dem bisherigen Belade-Code, lediglich die Ansteuerung der Postion findet jetzt in einer Schleife statt.

```
private void Beladen(int Position, int Runde) {
   ft.SetMotor(mRadMotor, cLinks);
   for(int n = 1; n <= Position; n++) ft.WaitForHigh(eRadPos);
   ft.Pause(1500);
   ft.SetMotor(mRadMotor, cAus);
   lblStatus.Text = "Gondel : " + Runde.ToString();
   ft.WaitForLow(eQuittung);
}
```

Anstelle des Belade-Codes in Action() findet man dort eine for-Schleife, die Beladen aufruft. Der erste Parameter gibt die relative Nummer der nächsten Beladeposition an (Anzahl Gondeln, die zu überspringen sind, + 1). Der zweite Parameter gibt an die wievielte Gondel zu beladen ist, sie wird aus der äußeren for-Schleife abgeleitet.

```
while (!ft.Finish(eQuittung)) {
  lblStatus.Text = "--- Aus- und Einsteigen ---";
  for(int i = 1; i <= 3; i++) {
    Beladen(1, i * 2 -1);
    Beladen(3, i * 2);
  }
lblStatus.Text = "Fahrbetrieb : Quittungs-Taster";
```

Beladen wird in der Reihenfolge 1 – 4 – 5 – 2 – 3 – 6. Also die gegenüberliegende Gondel und deren Nachbar. Denkbar sind natürlich auch noch andere Beladepläne.

Man könnte auch noch durch Anbau eines zusätzlichen Tasters samt Nocken die Gondel Nr. 1 identifizieren. Dann kann man auch noch die einzelnen Gondeln mit der zugehörenden Nummer beschriften, so kann die man über die Position jede Gondel genau Buchführen.

Referenz

Programmrahmen

Anwendungen, die die FishFace2005.DLL verwenden, enthalten eine Reihe von immer wiederkehrenden Elementen :

```
// ---(0)----
using FishFace40;

// ---(1)---

FishFace ft = new FishFace();

// ---(2)---

try  {
// ---(3)---
  ft.OpenInterface(IFTypen.ft..., ...    );
       .... Anwendungs-Code hier, bei größeren Anwendungen
            Methoden-Aufrufe .......
}
catch(FishFaceException eft) {
// ---(4)---
       ... Ausgabe von Fehlermeldungen ...
       .... eft.Message;
}
finally {
// ---(5)---
  ft.CloseInterface();
}
```

(0) Die FishFace-Funktionen befinden sich im Namespace FishFace40 der FishFace2005.DLL. Für die FishFace2005.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 ft.CloseInterface direkt hinter den catch-Block stellt, geht es auch ohne finally.

(3) Mit OpenInterface wird die Verbindung zum Interface hergestellt (am einfachsten gibt man hier den eigenen Anschluß angeben fest ein). (5) ft.CloseInterface() 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. Mit der FishFace2005.DLL werden im Verzeichnis \Templates40 auch verschiedene fertige Programmrahmen mitgeliefert, die man dann selber als Vorlagen speichern kann. Dazu siehe "Anmerkungen zu C#" im Anhang.

Verwendete Variablenbezeichnungen

Die Variablen sind durchweg vom Typ Integer (int). Parallel dazu gibt es Aufrufvarianten (overload), die enums verwendet. Hier werden zur Beschreibung des Wertebereichs einer Variablen die beschreibende Namen angegeben und in Klammern das entsprechende enum bzw. der Datentyp.

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

AnalogNr	Nummer eines Analog-Einganges (Inp)
AnalogWert	Rückgabewert beim Auslesen von AX/AY (AXS1 – AXS3) : 0 – 1023
AnalogZyklen	Anzahl der Zyklen nach dem die Analogwerte ausgelesen werden. (nur Intelligent Interface, typisch : 5)
Code	Angabe mit welcher Code-Taste die Eingaben des IR_Senders ausgewertet werden sollen (IRCode)
ComNr	Nummer des COM-Ports für eine Interface-Verbindung (Ports).
Counter	Wert eines ImpulsCounters (int)
DeviceData	Struktur mit Angaben zum Interface
Direction	Drehrichtung eines Motors (Dir)
DistanceMode	Modus in dem die D-Eingänge betrieben werden (Distance.Off, .UltraSonic, .Voltage)
DisNr	Bezeichnung des Distanzsensoreinganges (D1/D2)
ifTyp	Typ des angeschlossenen Interface (IFTypen)
InputNr	Nummer eines I-Einganges (Inp)
InputStatus	Rückgabewert beim Auslesen aller I-Eingänge (0 – 0xFFFFFFFF, 1bit pro Eingang)
KeyNr	Nummer der vom IR-Sender erwarteten Taste (IRKeys)
LampNr	Nummer eines O-Ausganges ("halben"-M-Ausganges) (Out)
MessageData	Struktur mit den Daten einer Nachricht
ModeStatus	Status der Betriebsmodi aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1, Werte 00 normal, 01 RobMode
MotorNr	Nummer eines M-Ausganges (Out)
MotorStatus	SollStatus aller M-Ausgänge. Jeweils 2 bit pro Ausgang. Begonnen bei 0-1 für M1 (00 = Aus, 01 = Links, 10 = Rechts). Bei O-Ausgängen kann jedes bit einzeln gesetzt werden.
mSek	Zeitangabe in MilliSekunden
NrOfChanges	Anzahl Impulse (int)
OnOff	Ein/Ausschalten eines M-Ausganges (Dir)
Position	Positionsangabe in Impulsen (int)
SerialNr	Standardseriennummer eines ROBO Interfaces
Speed	Geschwindigkeit mit der ein M-Ausgang betrieben werden soll (Speed)
SpeedStatus	Status der Geschwindigkeiten aller M-ausgänge. Jeweils 4 bit pro Ausgang. Begonnen bei 0-3 für M1. Werte 0000 – 1111 (Full). Ab M17 : SpeedStatus16.
TermInputNr	Nummer eines I-Einganges mit der die Methode beendet werden soll (Nr)

Value	allgemeiner Integer-Wert
VoltNr	Nummer eines Spannungseinganges (Inp)
WaitWert	Rückgabewert von WaitForMotors (Wait)

enum's

Verwendung zur Eingaben von Parametern bei den FishFace-Methoden und den darauf aufbauenden Klassen FishRobot und FishStep.

IFTypen	Bezeichnung der anschließbaren Interfaces
Port	Angabe des zu nutzenden Ports
Dir	Angabe der Drehrichtung ...
Distance	Betriebsmodus der D-Eingänge
Inp	Angabe der Nummer eines Einganges
Out	Angabe der Nummer eines Ausganges
IRCode	Auswertungsart beim IR_Sender
IRKeys	Getätigte Taste am IR_Sender
Speed	Geschwindigkeitsangabe
Wait	Return-Werte von WaitForMotors

Parallel dazu können auch entsprechende numerische (int) Angaben gemacht werden. Dazu siehe "Verwendete Variablenbezeichungen". Zu beachten ist, daß es hier ein Entweder/Oder gibt : in einer Methode können nur entweder enums oder eigene Konstanten verwendet werden. Wenn mans mag kann man allerdings auch den enums im Bedarfsfall ein (int) voranstellen.

Strukturen

DeviceData

Informationen zum aktiven Interface

Name	Bezeichnung des Interface (string)
Type	Interfacetyp (int)
SerialNr	aktuelle Seriennummer (int)
Firmware	Firmware Version (string)

MessageData

Daten einer Nachricht

HwId	Empfängerkreis (byte)
SubId	Klassifizierung (byte)
MsgId	Nachrichtennummer (ushort)
Msg	Nachricht (ushort)

Klasse FishFace

Enthalten in der Assembly FishFace2005.DLL (Source-File : FishFace2005.CS).
FishFace ist die Basisklasse der Assembly FishFace2005.DLL. Verwendet wird der Namespace FishFace40.

Konstruktor

FishFace()
Ohne Parameter

Eigenschaften

DeviceData	**ActDevice** Informationen über das aktive Interface in der Struktur DeviceData. Es muß ein erfolgreiches OpenInterface vorangegangen sein.
bool	**IsDistanceMode** Betriebsmodus der D-Eingänge. true : mit Distanzsensor
bool	**NotHalt** Anmelden eines Abbruchwunsches (Default = false).
int	**Outputs** Lesen/Schreiben der Werte aller M-Ausgänge (MotorStatus)
string	**Version** (get, static) Version der DLL

Methoden

ClearCounter

Löschen (0) des angegebenen Counters

ft.**ClearCounter**(InputNr)

Siehe auch : ClearCounters, GetCounter, SetCounter

ClearCounters

Löschen (0) aller Counter

ft.**ClearCounters**()

Siehe auch : ClearCounter, GetCounter, SetCounter

ClearMessagesIn

Löschen des Speichers der einkommenden Nachrichten

ft.**ClearMessagesIn**()

Exception : KeinOpen, NachrichenFehler

Siehe auch : ClearMessagesOut, GetMessage, IsMessage, SendMessage, WaitForMessage

ClearMessagesOut

Löschen des Speichers der ausgehenden Nachrichten

ft.**ClearMessagesOut**()

Exception : KeinOpen, NachrichenFehler

Siehe auch : ClearMessagesIn, GetMessage, IsMessage, SendMessage, WaitForMessage

ClearMotors

Abschalten aller M-Ausgänge

ft.**ClearMotors**()

Exception : InterfaceProblem, KeinOpen

Siehe auch : SetMotor, SetMotors, SetLamp Outputs

CloseInterface

Schließen der Verbindung zum Interface

ft.**CloseInterface**()

Siehe auch : OpenInterface

Finish

Feststellen eines Endewunsches (NotHalt, Escape, I-Eingang(optional))

bool = ft.**Finish**(Optional InputNr)

Exception : InterfaceProblem, KeinOpen. DoEvents

Siehe auch : GetInput, GetInputs

Beispiel :

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

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

Alternativ :

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

Die Schleife wird ggf. übersprungen (!ft.Finish(Nr.I1) ginge hier natürlich auch.

Überladung IRKeys :

Feststellen eines Endewunsches (NotHalt, Escape, IRKey)

bool = ft.**Finish**(IRCode, IRKey);

Exception : InterfaceProblem, KeinOpen. DoeEvents.

Beispiel :

```
while (!ft.Finish(IRCode.Code1, IRKeys.M3L) {
....
}
```

Der while Loop wird solange durchlaufen, bis entweder ft.NotHalt = true, die ESC-Taste gedrückt oder M3L am IR-Sender gedrückt wurde.

GetAnalog

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

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

Exception : InterfaceProblem, KeinOpen, DoEvent

Siehe auch : GetVoltage

Beispiel

```
Console.WriteLine(" AX : " + ft.GetAnalog(Inp.AX).ToString());
```

WriteLine gibt den aktuellen Wert von AX aus.

GetCounter

Auslesen des Wertes des angegebenen Counters

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

Siehe auch : SetCounter, ClearCounter, ClearCounters

Beispiel

```
Console.WriteLine("Counter für I2 : "+
ft.GetCounter(Inp.I2).ToString());
```

Der aktuelle Zählerstand, der dem I-Eingang I2 zugeordnet ist, wird ausgegeben.

GetDistance

Auslesen eines zu einem D-Eingang gehörenden Distanzsensorwertes [cm].
OpenInterface muß dazu mit Distance.UltraSonic erfolgt sein.

Value = ft.**GetDistance**(DisNr, DisValue);

Exception : InterfaceProblem, KeinOpen, DoEvent.

Siehe auch OpenInterface, DistanceMode, WaitForGreater, WaitForLess

Beispiel

```
int Abstand = ft.GetDistance(Inp.D1);
```

Der aktuelle Abstand in cm zu einem Hindernis wird bestimmt.

GetInput

Auslesen des Wertes des angegebenen I-Einganges

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

Exception : InterfaceProblem, KeinOpen. DoEvents

Siehe auch : GetInputs, Inputs, Finish, WaitForInput

Beispiel

```
if (ft.GetInput(Inp.I1)) {
  ...
}
else {
  ...
}
```

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

GetInputs

Auslesen der Werte aller I-Eingänge

IntputStatus = ft.**GetInputs**()

Exception : InterfaceProblem, KeinOpen; DoEvents

Siehe auch : GetInputs, Finish, WaitForInputs

Beispiel

```
int E13;
  E13 = ft.GetInputs();
  if ((E13 & (0x1 + 0x4)) > 0) Console.WriteLine("TRUE");
```

Der Block wird ausgeführt, wenn die I-Eingänge I1 oder I3 true sind.
Alternativ :

```
  if ((E13 & 0x1) > 0 || (E13 & 0x4) > 0) cn.WriteLine("TRUE");
```

GetIRKey

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

bool = ft.**GetIRKey**(Code, KeyNr);

Exception : InterfaceProblem, KeinOpen; DoEvents

Siehe auch : GetInputs, GetInput, Finish

Beispiel :

```
if(ft.GetIRKey(IRCode.Code1, IRKeys.M2L);
....
else ...
```

Wenn die IR_Taste M2L = true ist und Code1 aktiv ist, wird der true Zweig durchlaufen.

GetMessage

Auslesen der obersten Nachricht aus dem Empfangspuffer

MessageData = ft.**GetMessage**();

Exception : KeinOpen, NachrichtenFehler

Siehe auch : ClearMessagesIn, ClearMessagesOut, IsMessage, SendMessage, WaitForMessage

GetVoltage

Feststellen des Spannungswertes des angegebenen Spannungs-Einganges.
Bei OpenInterface mit Distance.Voltage können auch die D-Eingänge ausgelesen (Inp.D1V /.D2V) werden. Wenn bei Distance.UltraSonic Spannungen eingespeist werden, können Schäden auftreten, ggf. mit Vorwiderständen (220 – 470 Ohm) arbeiten.

Value = ft.**GetVoltage**(VoltNr);

Exception : InterfaceProblem, KeinOpen; DoEvents

Siehe auch : GetAnalog

Beispiel :

```
lblVolt.Text = ft.GetVoltage(Inp.A1).ToString();
```

Dem Label lblVolt wird der aktuelle Wert von A1 zugewiesen.

IsMessage

Befindet sich mindestens eine Nachricht im Empfangspuffer

bool = ft.**IsMessage**();

Exception : KeinOpen

Siehe auch : ClearMessagesIn, ClearMessagesOut, GetMessage, SendMessage, WaitForMessage

OpenInterface

Herstellen der Verbindung zum Interface. OpenInterface muß als erste Methode aufgerufen werden. Für das ROBO Interface an USB und Interface am COM-Port gibt es Überladungen:

Überladung USB :

ft.**OpenInterface**(ifTyp, SerialNr, DistanceMode, DoEvents);
Wenn ein RF Datalink an USB angeschlossen ist, wird ihm das erste (oder einzige) gefundene ROBO Interface zugeordnet.
– ifTyp : Interface Typ : ftROBO_IF_USB, ftROBO_RF_Datalink, ftROBO_IO_Extension. ftROBO_first_USB steht für das erste an USB gefundene Interface. Empfiehlt sich, wenn nur mit einem Interface gearbeitet wird. Die SerialNr muß dann auf 0 gesetzt werden.

- SerialNr (Standardseriennummer) : Unterscheidung gleichartiger Interfaces durch eine – freivergebbare – laufende Nummer, die in das Interface geschrieben wurde (z.Zt. durch ROBO Pro). Interfaces von unterschiedlichem ifTyp können die gleiche SerialNr haben.
- DistanceMode (Optional, default = Distance.Voltage) Betrieb mit Distanzsensor (Distance.UltraSonic).
- DoEvents (Optional, default = true), mit/ohne Application.DoEvents in den Methoden.

Exception : InterfaceProblem

Siehe auch : CloseInterface

Beispiel

```
try {
  ft.OpenInterface(IFTypen.ftROBO_first_IF_USB, 0);
......
}
catch(FishFaceException eft) {
  Console.WriteLine(eft.Message);
}
finally  {
  ft.CloseInterface();
}
```

Herstellen der Verbindung zum ersten (oder einzigen) Interface an USB, DoEvents wird ausgeführt. Im Fehlerfall wird der Text '`InterfaceProblem.Open`' ausgegeben

Überladung RF Datalink :

ft.**OpenInterface**(SerialNrInterface, DistanceMode, DoEvents);
Herstellen einer Verbindung zu einem ROBO Interface (nur an Spannung, Funk-Platine) über ein ROBO RF Datalink an USB. Beide müssen die gleiche Kanal-Nr. haben (z.B. RF 2/5 – RF2/0), die Nummer des Unterkanals ist beliebig. Nur sinnvoll, wenn noch weitere ROBO Interfaces (nur an Spannung, Funk-Platine) über Funk betrieben werden sollen und bestimmt werden soll, welches direkt über ein PC-Programm betrieben werden soll.
– SerialNrInterface : Aktuelle Seriennummer des Interfaces zu dem durch geschaltet werden soll.

- DistanceMode (Optional, default = Distance.Voltage) Betrieb mit Distanzsensor (Distance.UltraSonic).
- DoEvents (Optional, default = true), mit/ohne Application.DoEvents in den Methoden.

Beipiel :

```
try {
  ft.OpenInterface(7);
....
}
catch ....
```

Voraussetzung RF Datalink mit RF2/0 und ROBO Interface mit RF2/x und akt. Seriennummer 7. Das Interface mit Seriennummer 7 wird durch das PC-Programm über RF Datalink betrieben.

Überladung COM :

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

- ifTyp : Interface Typ : ftIntelligent_IF, ftIntelligent_IF_Slave, ftROBO_IIM, ftROBO_COM.
- ComNr : Nummer des COM-Ports an dem das Interface angeschlossen ist (z.B. Port.COM1 oder einfach 1).
- DoEvents (Optional, default = true), mit/ohne Application.DoEvents in den Methoden.

Exception : InterfaceProblem.

Siehe auch : CloseInterface

Beispiel :

```
try {
   ft.OpenInterface(IFTypen.ftIntelligent_IF, Port.COM1, 5, false);
....
}
catch(FishFaceException eft) {
  Console.WriteLine(eft.Message);
}
finally {
   ft.CloseInterface();
}
```

Herstellen einer Verbindung zu einem Intelligent Interface an COM1, alle 5 Zyklen werden die A-Eingänge upgedatet, kein Application.DoEvents.

Pause

Anhalten des Programmablauf für mSek MilliSekunden

ft.**Pause**(mSek)

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForTime

Beispiel

```
ft.SetMotor(Out.M1, Dir.Links);
ft.Pause(1000);
ft.SetMotor(Out.M1, Dir.Aus);
```

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

SendMessage

Senden einer Nachricht

ft.**SendMessage**(ref MessageData, Spez);
- MessageData : Struktur, die die Nachricht enthält.
– Spez : Spezifikation, die die Art der Versendung spezifiziert (optional, default = 0)
  0 = wird unbedingt gesendet, 1 = wird nur gesendet, wenn ungleich der letzten Nachricht in der Queque, 2 = wird nur gesendet, wenn nicht in der Queque vorhanden.

Exception : KeinOpen, NachrichtenFehler

Siehe auch : ClearMessagesIn, ClearMessagesOut, GetMessage, IsMessage, WaitForMessage

SetCounter

Setzen des Counters für den angegebenen I-Eingang

ft.**SetCounter**(InputNr, Value)

Exception : KeinOpen, InterfaceProblem

Siehe auch : GetCounter, ClearCounter, ClearCounters

SetLamp

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

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

- Power : Intensität des "Leuchtens", optional, default = 7.

Exception : InterfaceProblem, KeinOpen

Siehe auch : SetMotors, SetMotors, ClearMotors

Beispiel

```
const int lGruen = (int)Out.O1, lGelb = (int)Out.O2,
      lRot = (int)Out.O3, cEin = (int)Dir.Ein, cAus = (int)Dir.Aus;

  ft.SetLamp(lGruen, cEin);
  ft.Pause(2000);
  ft.SetLamp(lGruen, cAus);
  ft.SetLamp(lGelb, cEin);
```

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

SetMotor

Setzen eines M-Ausganges (Motor). Die Motordrehzahl kann gewählt werden (Default = Full), ebenso die Fahrstrecke in Anzahl Impulsen. Siehe auch "Anmerkungen zu den Rob-Funktionen.

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

Die Parameter ab Speed sind optional (default Speed : Speed.Full, kein Counter)

Exception : InterfaceProblem, KeinOpen; DoEvents; Counter (bei Parameter Counter)

Siehe auch : SetMotors, ClearMotors, SetLamp, Outputs.

Beipiel 1

```
ft.SetMotor(Out.M1, Dir.Rechts, Speed.Full);
ft.Pause(1000);
ft.SetMotor(Out.M1, Dir.Links, Speed.Half);
ft.Pause(1000);
ft.SetMotor(Out.M1, Dir.Aus);
```

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

Beispiel 2

```
ft.SetMotor(Out.M1, Dir.Links, 12, 123);
```

Der Motor am M-Ausgang M1 wird für 123 Impulse am I-Eingang I2 oder I1 = true mit Geschwindigkeitsstufe 12 eingeschaltet. Das Abschalten erfolgt selbsttätig, das Programm läuft solange weiter. Siehe Auch Beispiel WaitForMotors.

SetMotors

Setzen des Status aller M-Ausgänge, optional mit Geschwindigkeitsangabe (SpeedStatus) und des Betriebsmodes (ModeStatus, default = 0). Bei Betriebsmodus RobMode sind vor dem Aufruf der Methode die entsprechenden Counter zu setzen (SetCounter[m]) Siehe auch "Anmerkungen zu den Rob-Funktionen"

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

Exception : InterfaceProblem, KeinOpen; DoEvents, Counter(bei Parameter Counter)

Siehe auch : ClearMotors, SetMotors, SetLamp, Outputs

Beispiel

```
ft.SetMotors(0x1 + 0x80);
ft.Pause(1000);
ft.ClearMotors();
```

Der M-Ausgang (Motor) M1 wird auf links geschaltet und gleichzeitig M4 auf rechts. Alle anderen Ausgänge werden ausgeschaltet. Nach 1 Sekunde werden alle Ausgänge abgeschaltet.

WaitForChange

Warten auf NrOfChanges Impulse an InputNr oder TermInputNr = True

Intern wird der zu InputNr gehörende Counter genommen, der dazu zu Beginn zurückgesetzt wird.

ft.**WaitForChange**(InputNr, NrOfChanges, Optional TermInputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar.

Siehe auch : WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh.

Beispiel

```
ft.SetMotor(Out.M1, Dir.Links);
ft.WaitForChange(Inp.I2, 123, Inp.E1);
ft.SetMotor(Out.M1, Dir.Aus);
```

Der M-Ausgang (Motor) M1 wird linksdrehend geschaltet, es wird auf 123 Impulse an I-Eingang I2 oder I1 = true gewartet, der Motor wird abgeschaltet. Solange wird der Programmablauf angehalten. Siehe auch Beispiel bei SetMotors : dort läuft das Programm weiter.

WaitForGreater

Warten an DisNr auf einen Wert größer als DisValue

ft.**WaitForGreater**(DisNr, DisValue);

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : OpenInterface, WaitForLess, GetDistance

Beispiel

```
ft.SetMotor(Out.M1, Dir.Right);
ft.WaitForGreater(Inp.D1, 15);
ft.SetMotor(Out.M1, Dir.Off);
```

Der Robot (M1) fährt solange rückwärts bis der an D1 gemessene Entfernungswert größer als 15 cm ist.

WaitForHigh

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

ft.**WaitForHigh**(InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForLow, WaitForChange, WaitForInput.

Beispiel

```
ft.SetMotor(Out.M1, Dir.Ein);
ft.SetMotor(Out.M2, Dir.Links);
```

```
ft.WaitForHigh(Inp.I1);
ft.SetMotor(Out.M2, Dir.Aus);
```

Eine Lichtschranke mit Lampe an M-Ausgang M1 und Phototransistor an I-Eingang I1 wird eingeschaltet. Ein Förderband mit Motor an M2 wird gestartet, es wird gewartet bis ein Teil auf dem Förderband aus der Lichtschranke ausgefahren ist (die Lichtschranke wird geschlossen), dann wird abgeschaltet. Die Lichtschranke muß vorher false sein (unterbrochen).

WaitForInput

Warten, daß der angegebene I-Eingang den vorgegebenen Wert annimmt. (Default = true)

ft.**WaitForInput**(InputNr, OnOff)

OnOff ist optional, default = true

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar

Siehe auch : WaitForChange, WaitForLow, WaitForHigh.

Beispiel

```
ft.SetMotor(Out.M1, Dir.Links);
ft.WaitForInput(Inp.I1);
ft.SetMotor(Out.M1, Dir.Aus);
```

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

Überladung IRKey :

Warten, daß der angegebene IRKey den vorgegebenen Wert annimmt

ft.**WaitForInput**(Code, IRKey, OnOff)

Optional : OnOff (default = True)

Exception : InterfaceProblem, KeinOpen; DoEvent; Abbrechbar

Siehe auch : WaitForChange, WaitForLow, WaitForHigh.

Beispiel :

```
ft.SetMotor(Out.M1, Dir.Links);
ft.WaitForInput(IRCode.Code1, IRKeys.M2L);
ft.SetMotor(Out.M1, Dir.Off);
```

Der Motor an M-Ausgang M1 wird gestartet, es wird auf IR-Sender Code1, M2L = True gewartet, dann wird der Motoer wieder abgeschaltet.

WaitForLess

Warten an DisNr auf einen Wert kleiner als DisValue

ft.**WaitForGreater**(DisNr, DisValue);

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : OpenInterface, WaitForLess, GetDistance

Beispiel

```
ft.SetMotor(Out.M1, Dir.Left);
ft.WaitForGreater(Inp.D1, 15);
ft.SetMotor(Out.M1, Dir.Off);
```

Der Robot (M1) fährt solange vorwärts bis der an D1 gemessene Entfernungswert kleiner als 15 cm ist.

WaitForLow

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

ft.**WaitForLow**(InputNr)

Exception : InterfaceProblem, KeinOpen; DoEvent, Abbrechbar

Siehe auch : WaitForChange, WaitForInput, WaitForHigh.

Beispiel

```
ft.SetMotor(Out.M1, Dir.Ein);
ft.SetMotor(Out.M2, Dir.Links);
ft.WaitForLow(Inp.I1);
ft.SetMotor(Out.M2, Dir.Aus);
```

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).

WaitForMessage

Warten auf eine Nachricht

Wait = ft.**WaitForMessage**(Time, out Nachricht)

Exception : KeinOpen, NachrichtenFehler, DoEvent, Abbrechbar

Time in mSek : die gewartet werden soll. Bei Time = 0 wird bis zum Eintreffen einer Nachricht gewartet.

Wait : Return-Code
- Ende : es wurde eine Nachricht empfangen
- Time : die vorgegebene Wartezeit ist abgelaufen
- ESC : die Methode wurde über ESC-Taste abgebrochen
- NotHalt : die Methode wurde über NotHalt = true abgebrochen.

Nachricht : Die empfangene Nachricht (Struktur MessageData)

Siehe auch : ClearMessagesIn, ClearMessagesOut, GetMessage, IsMessage, SendMessage

Beispiel :

```
do {
  .... do something ....
} while(ft.WaitForMessage(300, out inNachricht) == Wait.Time);
... Verarbeiten der Nachricht (Wait.Ende) oder des Abbruchs
```

Am Ende einer do Schleife wird 300 MilliSekunden auf das Eintreffen einer Nachricht gewartet, dann wird die Schleife erneut durchlaufen

WaitForMotors

Warten auf ein MotorReadyEreignis oder den Ablauf von Time

WaitWert = ft.**WaitForMotors**(Time, MotorNr, ...)

Time (int) : Zeit in MilliSekunden. Bei Time = 0 wird endlos gewartet.

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.

WaitWert(Wait) : Grund warum die Methode beendet wurde
Wait.Ende : Alle betroffenen M-Ausgänge = Dir.Aus
Wait.Time : Die vorgegebene Wartezeit ist abgelaufen
Wait.NotHalt : Die Eigenschaft NotHalt = True, alle betroffenen Motoren wurden angehalten
Wait.ESC : Die ESC-Taste wurde betätigt, alle betroffenen Motoren wurden angehalten.

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : SetMotor

Beispiel

```
ft.SetMotor(Out.M4, Dir.Links, Speed.Half, 50);
ft.SetMotor(Out.M3, Dir.Rechts, Speed.Full, 40);
do {
   cn.WriteLine(ft.GetCounter(Inp.I6).ToString() + " - " +
                ft.GetCounter(Inp.I8).ToString());
} while (ft.WaitForMotors(300,
                  (int)Out.M4, (int)Out.M3) == Wait.Time);
cn.WriteLine(ft.GetCounter(Inp.I6).ToString() + " - " +
             ft.GetCounter(Inp.I8).ToString());
```

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. Die do .. while-Schleife wartet auf das Ende der Motoren (ft.WaitForMotors). Alle 300 MilliSekunden wird in der Schleife die aktuelle Position angezeigt (300 ... = Wait.Time). Wenn die Position erreicht ist (<> Time), ist der Auftrag abgeschlossen, die Motoren haben sich selber beendet. Achtung hier wurde nicht auf NotHalt, oder ESC abgefragt, es könnte also auch vor Erreichen der Zielposition abgebrochen worden sein. In der Schleife wird auf Label lblPos die aktuelle Position angezeigt. Zusätzlich nach Ende der Schleife (Differenz zu 300 MSek).

WaitForPositionDown

Warten auf Errreichen einer vorgegebenen Position durch Abwärtszählen von der aktuellen

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

Ausgegangen wird von der aktuellen Position, die in Counter gespeichert ist, es werden solange Impulse von Counter abgezogen, bis der in Position angegebenen Stand erreicht ist. Counter enthält zusätzlich die dann tatsächlich erreichte Position (kann um einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = True beendet. Counter und Position müssen immer postive Werte (einschl. 0) enthalten.

TermInputNr ist optional

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForPositionUp, WaitForChange

Beispiel

```
int Zaehler = 12;
  ft.SetMotor(Out.M1, Dir.Links);
  ft.WaitForPositionDown(Inp.I2, ref Zaehler, 0, Inp.I1);
  ft.SetMotor(Out.M1, Dir.Aus);
  cn.WriteLine("Zählerstand : " + Zaehler.ToString());
```

Die aktuelle Position ist 12 (Zaehler), der Motor an M1 wird linksdrehend gestartet. WaitForPositionDown wartet dann auf Erreichen der Postion 0, der Motor wird dann ausgeschaltet. Wenn vorher I1 = true wird, wird ebenfalls abgeschaltet.

WaitForPositionUp

Warten auf Erreichen einer vorgegebenen Position durch Aufwärtszählen von der aktuellen.

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

Ausgegangen wird von der aktuellen Postion in Counter, es werden solange Impulse auf Counter aufaddiert, bis der in Position angegebene Stand erreicht ist. Counter enthält zusätzlich die dann tatsächlich erreichte Position (kann einen Wert höher liegen, wenn der Motor nochmal "geruckt" hat). Alternativ wird die Methode durch E-Eingang TermInputNr = true beendet. Counter und Position müssen immer postive Wrte (einschl. 0) enthalten.

TermInputNr ist optional.

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar

Siehe auch : WaitForPositionDown, WaitForChange

Beipiel

```
int Zaehler = 0;
  ft.SetMotor(Out.M1, Dir.Rechts);
  ft.WaitForPositionUp(Inp.I2, Zaehler, 24);
  ft.SetMotor(Out.M1, Dir.Aus);
  cn.WriteLine("Zähler : " + Zaehler.ToString());
```

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

WaitForTime

Anhalten des Programmablaufes für mSek MilliSekunden.

ft.**WaitForTime**(mSek)

Synonym für Pause

Exception : InterfaceProblem, KeinOpen; DoEvents; Abbrechbar.

Siehe auch : Pause

Beispiel

```
do {
  ft.SetMotors(0x1);
  ft.WaitForTime(555);
  ft.SetMotors(0x4);
  ft.WaitForTime(555);
} while (!ft.Finish());
```

In der Schleife do .. while wird erst M-Ausgang (Lampe) M1 eingeschaltet und alle anderen abgeschaltet (binär : 0001), dann gewartet, M2 (Lampe) eingeschaltet (Rest aus, binär 0100) und gewartet. Ergebnis ein Wechselblinker. Ende der Schleife durch ESC-Taste.

Allgemeine Anmerkungen

Die Methoden erwarten ein vorhergehendes OpenInterface. 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. Die Wait-Methoden setzen bei Bedarf den zugehörenden **Counter** zurück.

Die SetMotor(s)-Methoden sind **asynchron** d.h. der oder die angesprochenen Motoren (Lampen) werden mit der Methode gestartet. Sie laufen dann unabhängig vom Programm weiter. Sie werden durch ein weiteres SetMotors mit Direction = 0 (Aus) beendet. Ausnahme : SetMotor mit Count-Parameter. Diese Methode beendet sich nach Erreichen der vorgegebenen Position selber.

Die Wait-Methoden koordinieren – meist in Verbindung mit End- bzw. ImpulsTastern den asynchronen 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 zum Funkbetrieb

Am Funkbetrieb beteiligt sind das ROBO RF Datalink und ROBO Interfaces mit RF-Platine

Es gibt drei unterschiedliche Typen des Funkbetriebes :

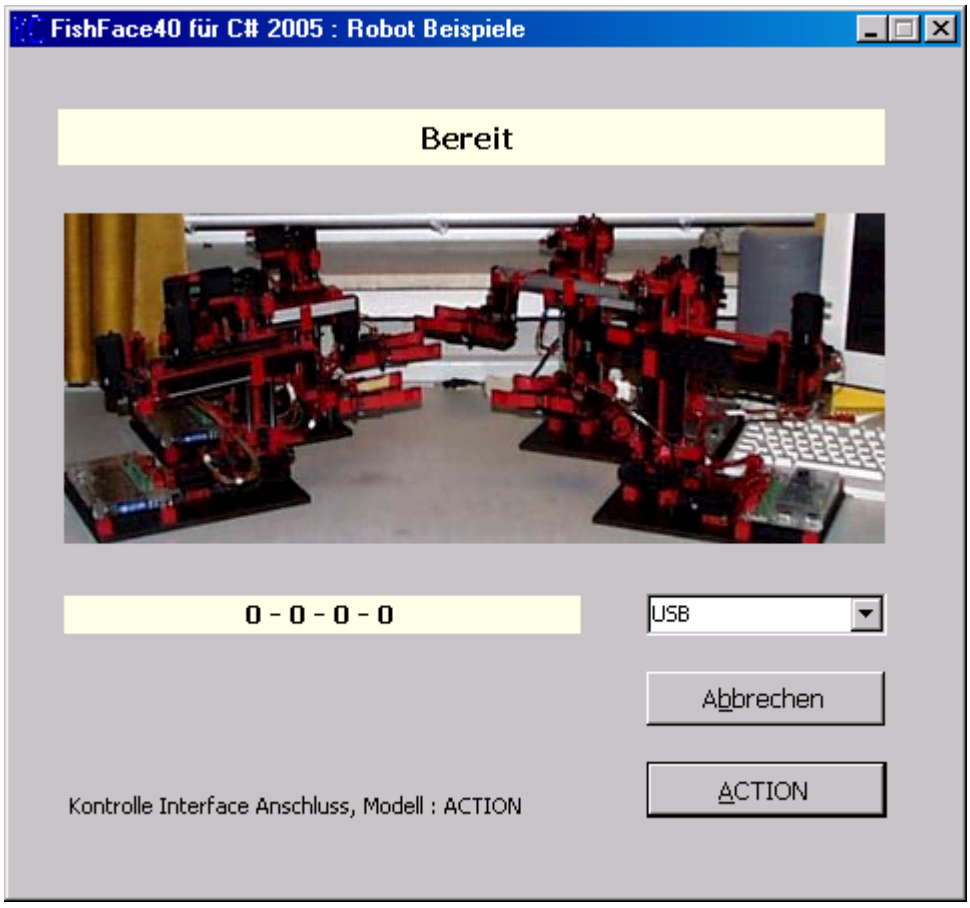
1. **Route Through** : Ein Interface mit RF-Platine ist über das RF Datalink an den PC angeschlossen. Das Anwendungsprogramm läuft im PC ohne die Anschlußart kennen zu müssen. Vorteil : Ein mit dem Interface + RF-Platine ausgerüstetes Modell kann sich frei im Raum bewegen, Kontrolle und Bedienoberfläche liegen auf dem PC
   Wird voll durch C# / FishFace2005.DLL unterstützt.
2. **Autonom** : Mehrere Interfaces mit RF-Platine kommunizieren miteinander über Funk Das RF Datalink übernimmt die Rolle des **Messages Routers**. Die Anwendungsprogramme laufen in den Interfaces.
   Wird durch C# / FishFace2005.DLL nicht unterstützt.
3. **Route Through** und **Message Router**. Das erste Interface wird durch eine PC-Anwendung unterstützt, die weiteren laufen autonom. Sie können aber von der Anwendung des ersten aus dem PC über Funk erreicht werden.
   Dieser Funkverkehr wird durch C# / FishFace2005.DLL von der PC-Seite aus unterstützt. Die Programmierung der Interfaces erfolgt in ROBO Pro oder Renesas C

Für (3.) gibt es einige speziellen Methoden :
- SendMessage zum Senden einer gepufferten (Broadcast) Nachricht.
– GetMessage, IsMessage und WaitForMessage zum entgegennehmen einkommender Nachrichten aus dem Empfangspuffer.
– Struktur MessageData zur Darstellung der Funk-Daten. Bis auf HwId das die Versandart enthält (RF Broadcast über Funk, Code 2). Können die Felder der Struktur in Abstimmung der am Funkverkehrbeteiligten frei vereinbahrt werden.

Siehe auch Beispiel bei Tips & Tricks.

Klasse FishRobot

Die Klasse FishRobot ist von FishFace abgeleitet und bietet zusätzlich zu den FishFace Eigenschaften und Methoden eine Reihe von Methoden, die speziell für den Betrieb von Robots des Typs "Industry Robot" geeignet sind. Charakteristika : Der Antriebsmotor treibt auf einer geeigneten Welle noch ein zusätzliches Impulsrad an, das einen Taster betätigt. Die Schaltvorgänge (Einschalten, Ausschalten separat) werden ab Robot Null gezählt. Der Robot Null wird durch einen weiteren Taster markiert, der bei einer Linksdrehung (Dir.Links) vom Robot angefahren wird. Die Taster sind dem M-Ausgang fest zugeordnet (M1 : I1 Endtaster, I2 Impulszähler). Siehe auch Abschnitt Anmerkungen | Rob-Funktionen.



Im Verzeichnis \Templates\FishRobotWindows findet sich ein gleichnamiges Projekt, das zum Erproben der Beispiele und zum Erstellen einer Vorlage dienen kann (siehe "Anmerkungen zu C#").

Konstruktor

FishRobot(int[,] MotList)

MotList : int[,] Liste der für den Robot-Betrieb eingesetzten Motoren. Jeweils Nummer des M-Ausganges und max. Fahrweg ab Endtaster in Impulsen.
Beispiel :

```
private FishRobot ft = new FishRobot(new int[,]{{1,123},{4,456}});
```

oder :

```
private int[,] MotList = new int[,] {{1,123),{4,456}});
private FishRobot ft = new FishRobot(MotList);
```

In beiden Fällen werden die Motoren an M1 (mit I1 Endtaster, I2 Impulstaster) und M4(mit I7 Endtaster, I8 Impulstaster) in den Robbetrieb einbezogen. Der Fahrweg beträgt 123 bzw. 456 Impulse ab Endtaster. Zu beachten ist, daß die anzufahrenden Positionen bei den Methoden MoveTo / MoveDelta in dieser Reihenfolge anzugeben sind.

Eigenschaften

MotCntl

Liste mit den Daten der bei der Instanzierung übergebenen Motordaten

ft.**MotCntl**[n].Nr Nummer des zugehörenden M-Ausganges

ft.**MotCntl**[n].maxPos Maximaler Fahrweg in Impulsen

ft.**MotCntl**[n].actPos Aktuelle Position ab Home (0) in Impulsen

Alle Werte vom Typ int. Zusammengefaßt in der struct MotWerte.

n bezieht sich auf die Motor-Position in der MotList der Instanzierung.

Methoden

MoveDelta

Simultanes Anfahren einer vorgegebenen Position relativ zur aktuellen.

ft.**MoveDelta**(params int[] DeltaList)

int DeltaList : Liste der anzufahrenden Positionen gezählt ab der aktuellen Positionen, bei negativen Werten wird die Fahrrichtung umgekehrt. Die Werte können als Aufzählung einzelner Werte oder alternativ als Array angegeben werden.

Exception : InterfaceProblem; DoEvent, Abbrechbar
Ereignis : PositionChange.

Siehe auch : MoveTo

Beispiel

```
FishRobot ft = new FishRobot(new int[,]{{3,234}, {4,123}});
.....
ft.MoveDelta(34, -12);
```

Der Motor an M3 fährt 34 Impulse nach rechts, der Motor an M4 fährt 12 Impulse nach links.

MoveHome

Simultanes Anfahren der Home Position

ft.**MoveHome**()

Betroffen sind die Motoren, die bei der Instanzierung angegeben wurden. Es wird nach links (Dir.Links) gefahren, bis der zugehörende Endtaster erreicht wird. Die erreichte Position wird auf 0 gesetzt.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Beispiel : siehe MoveTo

MoveTo

Simultanes Anfahren einer vorgegebenen Position bezogen auf die Home Position.

ft.**MoveTo**(params int[] PosList)

int PosList : Liste der anzufahrenden Positionen gezählt ab Home Position (zug. Endtaster bei Dir.Links). Wertebereich 0 – max. (Angabe bei der Instanzierung). Die Werte können als Aufzählung einzelner Werte oder alternativ als Array angegeben werden. Während der jeweils letzten 6 Impulse wird gebremst.

Exception : InterfaceProblem; DoEvent, Abbrechbar
Ereignis : PositionChange.

Siehe auch : MoveDelta

Beispiel

```
FishRobot ft = new FishRobot(new int[,]{{3,234}, {4,123}});
int [] PosList = new int[] {100, 200);
.....
ft.MoveHome();
ft.MoveTo(PosList);
```

Der gesamte Robot fährt mit den Motoren an M3 und M4 zunächst die Home Position an und dann die durch MoveTo vorgegebene. Der Motor an M3 fährt auf Position 100, der an M4 auf Position 123, da er bei Erreichen der max. zul. Position gestoppt wird.

Alternative Schreibweise :

```
FishRobot ft = new FishRobot(new int[,]{{3,234}, {4,123}});
.....
ft.MoveTo(100, 200);
```

Schreibweise 1 ist sinnvoll, wenn die Werte bestehenden Tabellen entnommen werden können (Abarbeiten einer Instruktionsliste). Schreibweise 2, wenn Einzelwerte vorliegen (z.B. beim TeachIn).

Ereignisse

PositionChange

Aufruf bei einer Veränderung der Position eines Motors durch die Methoden MoveTo / MoveDelta

ft.**PositionChange** += new **FishRobot.CommonDelegate**(name_event_ routine);

void name_event_routine(object sender, int[] actPos)

{ .... }

actPos : Liste mit den aktuellen Positionen ab Home für die Motoren nach der MotList der Instanzierung.

Beispiel

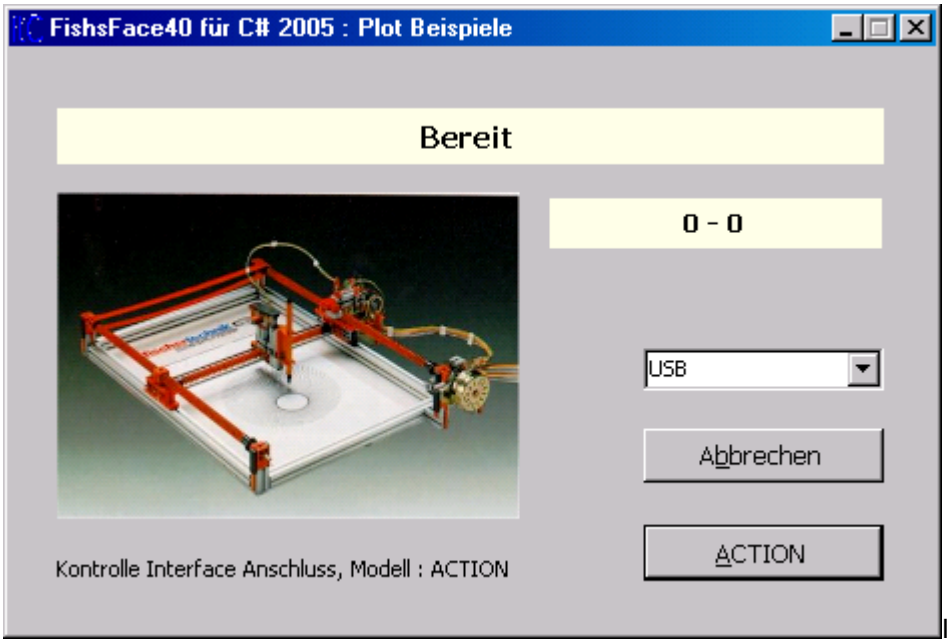
```
FishRobot ft = new FishRobot(new int[,]{{3,222},{4,88}});
.....
ft.PositionChange += new FishRobot.CommonDelegate(PositionsAusgabe);
.....
static void PositionsAusgabe(object sender, int[] actPos) {
  cn.WriteLine(actPos[0].ToString() + " - " + actPos[1].ToString());
}
```

ft.PositionChange : Für das Ereignis PositionChange wird in die Liste des zuständigen Delegate CommonDelegate die Ereignisroutine PositionsAusgabe eingetragen.

static void ... : Die Ereignisroutine. sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt. actPos eine Liste mit den aktuellen Positionen der laut Instanzierung übergebenen MotList. Die Positionen zählen in Impulsen ab Home positiv.

Klasse FishStep

Die Klasse FishStep ist von FishFace abgeleitet und bietet zusätzlich zu den FishFace Eigenschaften und Methoden eine Reihe von Methoden, die speziell für den Betrieb von Schrittmotoren geeignet sind. Dabei wird zwischen dem Betrieb einzelner Schrittmotoren (Anschlußbelegung : zwei aufeinanderfolgende M-Ausgänge, Methoden Step...) und dem Simultanbetrieb zweier zusammenhängender Schrittmotoren im XY-Verbund (Anschlußbelegung : drei aufeinanderfolgende M-Ausgänge für die beiden Motoren, Methoden Plot) unterschieden. Die Positionierung erfolgt in Zyklen (in der Regel vier Schaltvorgänge von 7,5°). Zu den M-Ausgängen gehören fest vorgegeben I-Eingänge zur Feststellung der Home Position. Dazu siehe auch "Anmerkungen zu den Step-Funktionen" am Ende des Dokumentes.



Im Verzeichnis \Templates\FishPlotWindows findet sich ein gleichnamiges Projekt, das zum Erproben der Beispiele und zum Erstellen einer Vorlage dienen kann (siehe "Anmerkungen zu C#").

Konstruktor

FishStep(int[,] MotList)

MotList : int[,] Liste der für den Step-Betrieb eingesetzten Motoren. Jeweils Nummer des M-Ausganges und max. Fahrweg ab Endtaster in Zyklen.

Beispiel:

```
private FishStep ft = new FishStep(new int[,]{{1,123},{3,456}});
```

oder :

```
private int[,] MotList = new int[,] {{1,123),{3,456}});
private FishStep ft = new FishStep(MotList);
```

In beiden Fällen werden die Motoren an M1/M2 (Endtaster I1) und M3/M4(Endtaster I5) in den Betrieb mit Step-Methoden einbezogen. Der Fahrweg beträgt 123 bzw. 456 Zyklen ab Endtaster. Beim Betrieb mit Plot-Methoden im XY-Verbund werden die Motoren an M1/M2 (Endtaster I1) und M3/M1(Endtaster I5) in den Betrieb einbezogen. Eine fließende Verteilung auf Interface und Extension Module ist möglich.

Eigenschaften

MotCntl

Liste mit den Daten der bei der Instanzierung übergebenen Motordaten

ft.**MotCntl**[n].maxPos	int Maximaler Fahrweg in Zyklen
ft.**MotCntl**[n].actPos	int Aktuelle Position ab Home (0) in Zyklen
ft.**MotCntl**[n].outPos	bool Angabe, ob einer der Motoren die actPos überschritten hat nur bei PlotTo/PlotDelta

n bezieht sich auf die Motor-Position in der MotList der Instanzierung.

Methoden

PlotDelta

Fahren eines Motorenpaares im XY-Verbund um Xrel / Yrel Zyklen bezogen auf die aktuelle Position

ft.**PlotDelta**(MotNr, int Xrel, int Yrel)

MotNr : Nummer (Nr. oder int) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.
Xrel / Yrel : Increment in Zyklen. Positive Werte rechtsdrehend (weg vom Endtaster, begrenzt durch den Wert der maxPos), negative linksdrehend (in Richtung Home Position / Endtaster).

Exception : InterfaceProblem; DoEvent, Abbrechbar.
Ereignis : PlotChange.

Siehe auch : PlotTo

Beispiel :
```
ft.PlotDelta(Out.M1, 100, -50);
```

Von der aktuellen Position wird um 100 Zyklen in X- und um 50 Zyklen (hin zum Endtaster) Y-Richtung gefahren.

PlotHome

Anfahren der HomePosition für einen XY-Verbund von zwei Schrittmotoren.

ft.**PlotHome**(MotNr)

MotNr : Nummer (Nr. oder int) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.

Gefahren wird in Richtung der zugeordneten Endtaster. Nach Erreichen der Endtaster wird um zwei Zyklen in Gegenrichtung freigefahren.

Exception : InterfaceProblem; DoEvent, Abbrechbar.

Beispiel :
```
private FishStep ft = new FishStep(new int[,]{{1,123},{3,456}});
ft.Home(Out.M1);
```

Bei der Instanzierung werden die M-Ausgänge, die von den Motoren genutzt werden sollen, und die max. Fahrwege angegeben. Hier wird der X-Motor an M1/M2 (Endtaster I1, Fahrweg 123 Zyklen) und der Y-Motor an M3/M1(Endtaster I5, Fahrweg 456 Zyklen) angeschlossen. Anschließend wird auf die Home Position gefahren.

PlotTo

Fahren eines Motorenpaares im XY-Verbund auf die Position Xabs / Yabs.

ft.**PlotTo**(MotNr, int Xabs, int Yrel)

MotNr : Nummer (Nr. oder int) des ersten M-Ausganges, der dem XY-Verbund bei der Instanzierung zugeordnet wurde.
Xabs / Yabs : Position ab Home Position (0) in Zyklen. Begrenzung durch Endtaster.

Exception : InterfaceProblem; DoEvents, Abbrechbar.
Ereignis : PlotChange

Beispiel :
```
ft.PlotTo(Out.M1, 150, 333);
```

Gefahren wird auf Position 123 / 273, wenn die Instanzierung (max 123 / 456)des Beispiels von PlotHome angenommen wird. Die Fahrwegbegrenzung hat hier also zugeschlagen. Der Fahrbefehl wird mit Erreichen von Xmax abgebrochen, daraus ergibt sich dann die erreichte Y-Position (123/150 * 333 = 273).

StepDelta

Fahren eines einzelnen Schrittmotors um Xabs Zyklen bezogen auf die aktuelle Position

ft.**StepDelta**(MotNr, int Xabs)

MotNr : Nummer (Out. oder int) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Bei positiven Werten wird weg vom Endtaster gefahren bei negativen Werten hin zum Endtaster. Fahrwegbegrenzung bzw. Endtaster werden beachtet.

Exception : InterfaceProblem; DoEvent, Abbrechbar.
Ereignis : StepChange.

Beispiel :
```
ft.StepDelta(Out.M5, 123);
```

Der Schrittmotor an M5/M6 fährt von der aktuellen Position rechtsdrehend (weg vom Endtaster I9) 123 Zyklen.

StepHome

Anfahren der Home Position des über MotNr angegebenen Schrittmotors.

ft.**StepHome**(MotNr)

MotNr : Nummer (Ou. oder int) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Gefahren wird in Richtung des zugeordneten Endtasters (I9).

Exception : InterfaceProblem; DoEvents, Abbrechbar:

Beispiel :
```
FishStep ft = new FishStep(new int[,]{{1,123},{3,456}});
ft.StepHome(Out.M3);
```

Der Motor an M3/M4 wird in Richtung des Endtasters (I9) gefahren, die aktuelle Position wird auf 0 gesetzt.

StepTo

Fahren eines einzelnen Schrittmotors auf Position Xabs.

ft.**StepTo**(MotNr, int Xabs)

MotNr : Nummer (Out. oder int) des ersten M-Ausganges, der dem Motor bei der Instanzierung zugeordnet wurde.

Endtaster und Fahrwegbegrenzung werden beachtet.

Exception : InterfaceProblem; DoEvent, Abbrechbar.
Ereignis : StepChange.

Beispiel :

```
const int mAufzug = 1;
ft.StepTo(mAufzug, 123);
```

Der Motor an M1/M2 fährt auf Position 123.

Ereignisse

PlotChange

Aufruf bei einer Veränderung der Position des Motorenpaares des XY-Verbundes durch die Methoden PlotTo/PlotDelta.

ft.**PlotChange** += new **FishStep.PlotDelegate**(name_event_routine)

void name_event_routine(object sender, int xPos, int yPos)
{ ... }

Beispiel :

```
FishStep ft = new FishStep(new int[,]{{1,123},{3,456}});
....
ft.PlotChange += new FishStep.PlotDelegate(PlotPosition);
....
static void PlotPosition(object sender, int MotNr,
                                              int xPos, int yPos) {
  cn.WriteLine("Position : " + xPos.ToString() + " / " +
                               yPos.ToString());
}
```

ft.PlotChange : Für das Ereignis PlotChange wird in die Liste des zuständigen Delegate – PlotDelegate – die Ereignisroutine PlotPosition eingetragen.

static void : Die Ereignisroutine. sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt. xPos / yPos die aktuelle Position des XY-Verbundes.

StepChange

Aufruf bei einer Veränderung der Position eines einzelnen Schrittmotors durch die Methoden StepTo/StepDelta.

ft.**StepChange** += new **FishStep.StepDelegate**(name_event_routine)

void name_event_routine(object sender, int actPos)

{ ... }

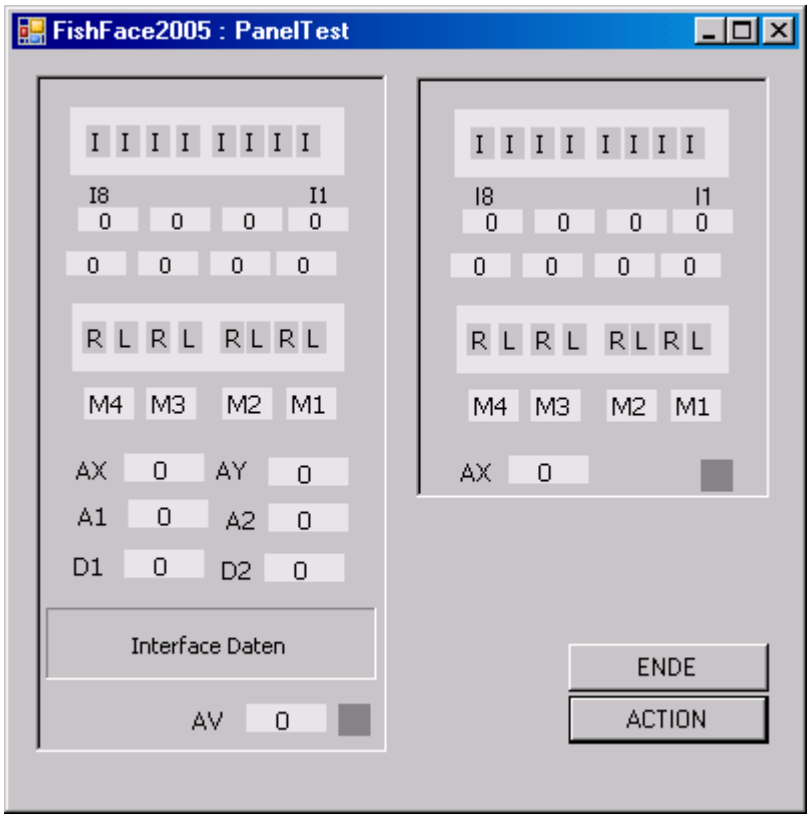
Beispiel :

```
FishStep ft = new FishStep(new int[,]{{1,123},{3,456}});
....
ft.StepChange += new FishStep.StepDelegate(StepPosition);
....
static void StepPosition(object sender, int MotNr, int actPos) {
  cn.WriteLine("Position : " + actPos.ToString() yPos.ToString());
}
```

ft.StepChange : Für das Ereignis StepChange wird in die Liste des zuständigen Delegate – StepDelegate – die Ereignisroutine StepPosition eingetragen.

static void : Die Ereignisroutine. sender enthält, wie gewohnt, einen Hinweis auf das rufende Objekt. actPos die aktuelle Position des Schrittmotors.

Control FishPanel und FishExtPanel



FishPanel FishExtPanel

FishPanel und FishExtPanel sind Controls zur Darstellung der aktuellen Werte des jeweiligen Interfaces bzw. der Extension. Zusätzlich können die M-Ausgänge über die Maus geschaltetet werden (Eigenschaft AllowInteract = true). Es können mehrere Controls entsprechend den angeschlossenen Interfaces eingerichtet werden.

FishPanel unterstützt alle über FishFace2005.DLL nutzbaren Interfaces, wenn sie direkt an den PC angeschlossen sind (Auch über RF Datalink bzw. Connect Box). Das FishExtPanel unterstützt die ROBO I/O Extension (Sowohl Anschluß an USB wie auch über Extensionkabel).

Die Anzeige der Interfacedaten ist optional (Eigenschaft ShowAll = true).

Einrichten der Controls in der Toolbox

1. Verweise auf FishFace2005 (ab 4.2.65.2005) wie normal
2. RechtsClick auf eine freie Stelle der Toobox : Registereinrichten (FishFace)
3. RechtsClick auf das neue Register : Elemente auswählen, Durchsuchen auf FishFace2005.DLL
4. Einfügen der Controls wie gewohnt.

Das wars.

Elemente der Controls

I-Eingänge (beim Intelligent/Universal Interface E-Eingänge)

Über die Leiste der I-Elemente

Counter Zähler für die Betätigung der I-Eingänge, Ein- und Ausschalten wird separat gezählt.

Leiste mit den Nullen. Obere Reihe von links : Counter für I8 .. I5, darunter I4 .. I1

M-Ausgänge Anzeige der M- bzw. der O-Ausgänge

Bei Eigenschaft AllowInteract = true auch Schalten der Ausgänge über Maus :
LinksClick auf R / L : Ein für die Dauer von MouseDown,
RechtsClick : Dauerhaftes Ein,
Ausschalten über M1 .. M4.

Analog Eingänge (Eingänge AX .. A2 bzw. EX /EY)

Wertebereich 0 – 1023. Rohdaten, bei A1 und A2 werden hundertstel Volt angezeigt.

Distance Eingänge

Messwerte der angeschlossenen Distanzsensoren in cm bzw. von Spannungen in hundertstel Volt.

Interface Daten

Bezeichnung des Interfaces (nach Enum IFTypen). In Klammern die num. Typbezeichnung und die eingestellte Seriennummer. In der zweiten Zeile die Version der Firmware (soweit feststellbar. Bei RF Datalink und der Connect Box deren Version).

AV und Knopf

Betriebsspannung in hundertstel Volt, und Betriebsstatus (grün OK, rot Fehler, grau nicht in Betrieb).

Nutzung

Einrichten einer FishFace Instanz für jedes direkt an den PC angeschlossene Interface. Meist im Zusammenhang mit der weiteren Programmierung. z.B.

```
FishFace ft = new FishFace();
```

Ebenfalls wie gewohnt : Open und Close der Interfaces, zusätzlich Start / Stop Panel :

```
ft.OpenInterface(IFTypen.ftROBO_first_USB, 0);
fishPanel1.Start(ft);
...
fishPanel1.Stop();
ft.CloseInterface();
```

Programmierung ansonsten wie gewohnt.

Tips & Tricks

Programmrahmen

Die in den folgenden Kapiteln Techniken.. angeführten Programmausschnitte benötigen einen Programmrahmen innerhalb dessen sie ablaufen können. Man kann die gleichen wie im Abschnitt Referenz nutzen, sie befinden sich im Verzeichnis \Templates40. Der Rahmen wird in den Kapiteln Techniken dann nicht mehr extra angegeben.

Allgemeine Techniken

Diese Techniken basieren auf der Klasse FishFace. Sie lassen sich natürlich auch bei Nutzung der davon abgeleiteten Klassen FishRobot und FishStep einsetzen.

Blinker/Schleife

Lampe an M1 blinkt im Sekundentakt :

```
const int mGelb = (int)Out.M1, cEin = (int)Dir.Ein,
                               cAus = (int)Dir.Aus;

do {
  ft.SetMotor(mGelb, cEin);
  ft.Pause(555);
  ft.SetMotor(mGelb, cAus);
  ft.Pause(444);
} while (!ft.Finish());
```

Die Parameter für die FishFace Methoden sollten benannt werden. Für einige Standardwerte gibt es bereits Namen (Enums) : Out.M1 ... Inp.I16, Dir.Ein, Dir.Aus, Dir.Links, Dir.Rechts ... und Wait.Ende, Wait.Time, Wait.ESC für die Methode WaitForMotors. Weitere sollte man selber erfinden. Zu beachten ist, daß es hier ein entweder/oder gibt : Verwenden der Enums oder eigener Namen.

Meistens enthält das Programm ein große Schleife in der alle Befehle wiederholt durchlaufen werden. Hier ist das `do .. while (!ft.Finish());` : Die Methode Finish prüft, ob ein Abbruchwunsch vorliegt und meldet dann true zurück.
Beenden der Schleifte duch ESC-Taste. Es kann auch zusätzlich ein I-Eingang angegeben werden : `ft.Finish(Inp.I8)`. Beendigung durch ESC-Taste oder I8.

WechselBlinker

Lampen an O1 und O2 blinken im Wechsel.

Alternative 1 :

```
do {
  ft.SetLamp(Out.O2, Dir.Aus);
  ft.SetLamp(Out.O1, Dir.Ein);
  ft.Pause(444);
  ft.SetLamp(Out.O1, Dir.Aus);
  ft.SetLamp(Out.O2, Dir.Ein);
  ft.Pause(444);
} while (!ft.Finish());
```

Alternative 2 kompakter:

```
do {
  ft.SetMotors(0x1);
  ft.Pause(444);
  ft.SetMotors(0x2);
  ft.Pause(444);
} while (!ft.Finish());
```

Hier werden alle M/O-Ausgänge gleichzeitig geschaltet. Jeweils 1 bit pro O-Ausgang. Also 00000001 für O1 Ein und 00000010 für M2 ein. Alle anderen Ausgänge sind Aus.

Abfrage eines I-Einganges

Wenn I1 geschaltet ist lblStatus = "---EIN---" sonst "---AUS---" :

```
if (ft.GetInput(Inp.I1)) lblStatus.Text = "--- EIN ---";
else lblStatus.Text =  "--- Aus ---";
```

Warten auf einen I-Eingang

Wenn I1 geschlossen ist, wird in lblStatus "--- Es geht los ---" ausgegeben :

```
lblStatus.Text = "--- Zum Programmstart I1 drücken ---";
ft.WaitForInput(Inp.I1);
lblStatus.Text = "--- Es geht los ---";
```

Anzeige des Status der I-Eingänge

Status von I1 :

```
lblStatus.Text = "I1 : " & ft.GetInput(Inp.I1)
```

Laufende Anzeige des Status aller E-Eingänge :

```
do {
   string EWerte = "";
   for(int i = 0x80, E = ft.Inputs; i > 0; i >>= 1)
       EWerte += (E & i) > 0 ? "1" : "0";
   lblStatus.Text = EWerte.ToString();
   ft.Pause(1234);
} while (ft.Finish());
```

Die Angelegenheit benötigt nur ein paar Zeilen, sieht aber etwas vertrackt aus – und ist es auch : In EWerte (beim ROBO Interface wäre wohl IWerte besser) werden die

Schaltzustände der (hier 8) I-Eingänge gesammelt. Das geschieht in einer for-Schleife mit der Laufvariablen i, die gleichzeitig auch noch eine logische Maske ist. i wird auf den Ausgangswert 0x80 (das bit für I8 im InputStatus) gesetzt, gleichzeitig auch E auf den aktuellen InputStatus. Die Schleife läuft solange i > 0 ist. Bei jedem Schleifendurchlauf wird das Maskenbit in i um eine Position nach rechts geschoben (z.B. von I8 -> I7). An die Variable EWerte wird bei jedem Schleifendurchlauf der Status eines I-Einganges ("1" bzw. "0") gehängt. Der Status wird durch ein logische Und (&) von E und i bestimmt, die Auswertung geschieht mit einem Bedingungsoperator (?) und einer verkürzten Zuweisung.

Es geht auch einfacher, wenn man sich mit einer schlichten Hexa-Ausgabe begnügt :

```
do {
   lblStatus.Text = ft.Inputs.ToString("X");
   ft.Pause(1234);
} while (ft.Finish());
```

Analog-Anzeige

Laufende Anzeige der beiden Analog-Eingänge AX und AY :

```
do {
  lblStatus.Text = "AX : "
             + ft.GetAnalog(Inp.AX).ToString()
             + " AY : " + ft.GetAnalog(Nr.AY).ToString();
  ft.Pause(1111);
while (!ft.Finish());
```

Fahren für eine bestimmte Zeit

Der Motor an M3 soll 3,5 Sekunden nach links laufen :

```
ft.SetMotor(Out.M3, Dir.Links);
ft.Pause(3500);
ft.SetMotor(Out.M3, Dir.Aus);
```

Fahren zum Endtaster

Der Motor an M3 soll den Endtaster I5 anfahren und dann abschalten :

```
ft.SetMotor(Out.M3, Dir.Links);
while(!ft.GetInput(Inp.I5));
ft.SetMotor(Out.M3, Dir.Aus);
```

Das Beispiel ist nicht durch ESC-Taste abbrechbar und auch etwas umständlich.
besser :

```
ft.SetMotor(Out.M3, Dir.Links);
ft.WaitForInput(Inp.I5);
ft.SetMotor(Out.M3, Dir.Aus);
```

Fahren um eine vorgegebene Anzahl von Schritten

WaitForChange

Motor an M3 mit Impulstaster an I6 soll um 12 Impluse fahren.

```
ft.SetMotor(Out.M3, Dir.Links);
ft.WaitForChange(Inp.I6, 12);
ft.SetMotor(Out.M3, Dir.Aus);
```

WaitForPositionDown

Motor an M3 fährt von IstPosition = 12 auf ZielPosition = 0,
Impulszählung an I6 in Richtung 0 (Endtaster = I5) :

```
int IstPosition = 12;
ft.SetMotor(Out.M3, Dir.Links);
ft.WaitForPositionDown(Inp.I6, ref IstPosition, 0, Inp.I5);
lblStatus.Text = "Istposition : " + IstPosition.ToString();
```

Die tatsächlich erreichte Position (kann um einen Impuls von der Vorgabe abweichen) steht nach dem Vorgang in IstPosition. Wird der Endtaster I5 vorher true, wird vorzeitig abgebrochen.

WaitForPositionUp

Motor an M3 fährt von IstPosition = 12 auf ZielPosition = 24,
Impulszählung an I6 in Richtung weg von Endtaster :

```
int IstPosition = 12;
ft.SetMotor(Out.M3, Dir.Rechts);
ft.WaitForPositionUp(Inp.I6, ref IstPosition, 24);
lblStatus.Text = "Istposition : " + IstPosition.ToString();
```

Die tatsächlich erreichte Position (kann um einen Impuls von der Vorgabe abweichen) steht nach dem Vorgang in IstPosition.

WaitForMotors

Der Motor an M3 fährt für 12 Impulse an I6 mit verminderter Geschwindigkeit nach Links.

```
ft.SetMotor(Out.M3, Dir.Links, Speed.L5, 12);
ft.WaitForMotors(0, 3);
```

Es wird gewartet, bis das Ziel erreicht wurde. Es geht auch ohne WaitForMotors, wenn das Programm anderweitig beschäftigt ist (Die Motoren schalten bei Erreichen der Zielposition selbsttätig ab). Siehe auch "Anmerkungen zu den Rob-Funktionen".

Zwei Motoren simultan mit laufender Positionsanzeige

Zwei Motoren (M3, M4) fahren simultan (gleichzeitig), die Impulszählung erfolgt an I6 und I8 (siehe auch Rob-Funktionen). Parallel dazu wird die aktuelle Position angezeigt :

```
ft.SetMotor(Out.M3, Dir.Links, Speed.Full, 121);
ft.SetMotor(Out.M4, Dir.Rechts, Speed.L8, 64);
do {
  lblStatus.Text = "Position M3 - M4 : " +
         ft.GetCounter(Inp.I6).ToString() + " - " +
         ft.GetCounter(Inp.I8).ToString();
} while (ft.WaitForMotors(300, 3, 4) == Wait.Time);
lblStatus.Text = "Position M3 - M4 : " +
         ft.GetCounter(Inp.I6).ToString() + " - " +
         ft.GetCounter(Inp.I8).ToString() + "--- Final ---
";
```

Motor M3 fährt mit voller Geschwindigkeit um 121 Impulse nach Links
Motor M4 fährt mit halber Geschwindigkeit um 64 Impulse nach Rechts
WaitForMotors wartet auf beide, alle 0,3 Sekunden wird die aktuelle Position angezeigt. Zum Schluß wird die tatsächlich erreichte Position angezeigt. Zur Positionsanzeige wird mit GetCounter die aktuelle Position ausgelesen.

Lampen

Lampen werden meistens genauso behandelt wie Motoren (mit zwei Polen an einem M-Ausgang, z.B. `ft.SetMotor(1, Dir.Ein))`, da sie aber nur ein oder ausgeschaltet werden können, ist auch die Schaltung an einem Pol eines M-Ausganges und Masse möglich man kann so bis zu acht Lampen an ein Interface anschließen (gilt nur für ROBO Interfaces) :

```
ft.SetLamp(Out.O1, Dir.Ein);
ft.SetLamp(Out.O4, Dir.Ein);
ft.Pause(1000);
ft.SetLamp(Out.O1, Dir.Aus);
ft.SetLamp(Out.O4, Dir.Aus);
```

Die Lampen an O1 und O4 und Masse werden für 1 Sekunde eingeschaltet.

Lichtschranken

Warten auf Lichtschranke

Lampe an M1, Phototransistor an I1. Es wird auf eine Unterbrechung der Lichtschranke gewartet :

```
const int mLicht = (int)Out.O1, ePhoto = (int)Inp.I1,
          cEin = (int)Dir.Ein;
ft.SetMotor(mLicht, cEin);
ft.Pause(555);
ft.WaitForInput(ePhoto, false);
```

Lampe wird eingeschaltet, danach 0,5 Sekunden Pause um den Phototransistor "anzuwärmen", dann wird auf eine Unterbrechung der Lichtschranke gewartet.

Warten auf Einfahrt in eine Lichtschranke

Lampe an M1, Förderbandmotor an M3, Phototransistor an I1 :

```
const int mBand = 2, ePhoto = 1;
ft.SetMotor(mBand, (int)Dir.Links);
ft.WaitForLow(ePhoto);
ft.SetMotor(mBand, (int)Dir.Aus);
```

Der Motor M1 läuft solange bis ein Teil auf dem Band in die vorher nicht unterbrochene Lichtschranke einfährt. Die Lichtschranke wurde bereits vorher eingeschaltet. Die Konstanten Zuweisung ist eher "unordentlich" es werden hier anstelle von enums num. Werte zugewiesen, das schreibt sich aber schneller.

Warten auf Ausfahrt aus einer Lichtschranke

Lampe an M1, Förderbandmotor an M3, Phototransistor an I1 :

```
const int mBand = 2, ePhoto = 1,
          Links = (int)Dir.Links, Aus = (int)Dir.Aus;
ft.SetMotor(mBand, Links);
ft.WaitForHigh(ePhoto);
ft.SetMotor(mBand, Aus);
```

Der Motor M1 läuft solange bis ein Teil auf dem Band, das die Lichtschranke unterbricht, aus der Lichtschranke herausgefahren ist.

Gleichzeitiges Schalten aller M-Ausgänge

Mit SetMotors können alle M-Ausgänge mit einem Befehl geschaltet werden. Dazu muß der Parameter MotorStatus entsprechend besetzt werden. Im MotorStatus sind pro M-Ausgang jeweils 2bit resierviert : 00 00 00 00 (bei Einsatz eines Extension Modules nochmals jeweils 4). 00 bedeutet ausgeschaltet, 01 Drehrichtung links bzw.
Ein, 10 Drehrichtung rechts.
00 01 00 00 demnach M3 links und 01 00 00 00 M4 links.

Einfache Ampel

Ein einfaches Ampelspiel sieht so aus : Grün – Gelb – Rot – RotGelb.
Die Lampen dazu M1 : Grün, M2 : Gelb, M3 : Rot und die Konstanten dazu :
mGruen = 00 00 00 01, mGelb = 00 00 01 00, mRot = 00 01 00 00,
dezimal = 1, 4, 16.

```
const int mGruen = 1, mGelb = 4, mRot = 16;
while (!ft.Finish()) {
  ft.SetMotors(mGruen);
  ft.Pause(1000);
  ft.SetMotors(mGelb);
  ft.Pause(250);
  ft.SetMotors(mRot);
  ft.Pause(1000);
  ft.SetMotors(mRot+mGelb);
  ft.Pause(250);
}
```

Listengesteuerte Ampel

Wenn man einen festen Ampeltakt vorgibt, kann man den Ablauf auch listengesteuert machen :

```
const int mGruen = 1, mGelb = 4, mRot = 16;
int[] Phase = {mGruen, mGruen, mGruen, mGruen,
               mGelb, mRot, mRot, mRot, mRot, mRot+mGelb};
while (!ft.Finish()) {
  foreach (int p in Phase) {
    ft.SetMotors(p);
    ft.Pause(250);
  }
}
```

Hier wird mit einer festen Taktung von 250 MilliSekunden gearbeitet. Das Verfahren lohnt bei komplexeren Steuerungen.

Lauflicht

Wenn an einem Interface gerade 4 Lampen angeschlossen sind, kann man ganz einfach ein Lauflicht programmieren :

```
while (!ft.Finish()) {
  for (int Phase = 1; Phase < 0xFF; Phase <<= 2) {
    ft.SetMotors(Phase);
    ft.Pause(555);
  }
}
```

Phase ist gleichzeitig Laufvariable und MotorStatus. Es wird jeweils ein M-Ausgang eingeschaltet. Dazu werden zyklisch 2 bit durch die Phase geschoben.

Funk-Betrieb

Das auf dem PC laufende Programm betreibt ein über das RF Datalink zugeordnetes Interface transparent, d.h. es besteht aus Sicht des Programms kein Unterschied zwischen einem Anschluß über Kabel und dem "Anschluß" über Funk. Es ist allerdings möglich zusätzliche Interfaces, die ebenfalls mit einer Funk-Karte ausgestattet sein müssen, über Funk Nachrichten zu senden und von ihnen Nachrichten zu empfangen.

Die Funk-Nachrichten sind Broadcast-Nachrichten. d.h. alle Teilnehmer des Funkverkehrs (mit der gleichen Frequenz, hier 2) können sie hören, eine Quittung wird nicht gesendet.

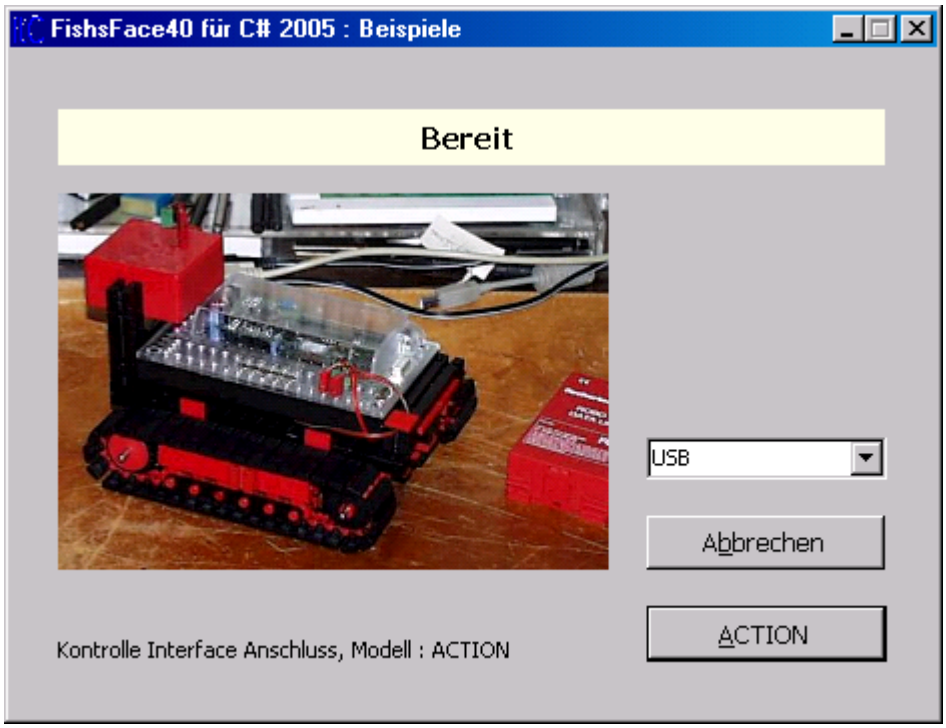
Senden und Empfangen von Nachrichten

Hier ein Beispiel : Steuerung einer Raupe der Power Bulldozer durch ein zweites Interface. Durch Betätigen des Taster an I1 – RF 2/1 wird auf RF 2/2 ein "Schaulaufen" gestartet. Während des Schaulaufen leuchtet an RF 2/1 ein rote Lampe, bei Bereitschaft eine grüne.

Konfiguration :

- PC-Programm zur Steuerung des Gesamtablaufs und zum Betrieb des zugeordneten Interfaces (RF 2/1) über das RF Datalink (RF 2/0)
- ROBO RF Datalink über USB , Kennung RF 2/0. Aufgabe : Message Routing
- ROBO Interface mit Funk-Karte, Kennung RF 2/1. "Fernsteuerung" der Raupe Taster an I1 zum Start des Schaulaufens, grüne Lampe an O1, rote an O2
- ROBO Interface mit Funk-Karte, Kennung RF 2/2
  Power Motoren an M1 und M2 zum Antrieb der Raupen.

Das PC-Programm : FunkRaupe.CS



Der Programmrahmen wurde auf Basis der Vorlage FishFaceWindows erstellt.

Startet den Gesamtbetrieb und steuert das Interface RF 2/1 und sendet Nachrichten an die Raupe auf dem Bild.

Die Source-Liste FunkRaupe.CS

```
public partial class FunkRaupe : Form {
  FishFace ft = new FishFace();
  private void Action() {
    MessageData inN;
    MessageData outN;
    outN.HwId = 0;
    outN.Msg = 0;
    outN.MsgId = 0;
    outN.SubId = 0;

    ft.SetLamp(Out.O1, Dir.Ein);
    lblStatus.Text = "Warten auf Auftrag : I1";
    do {
      if (ft.GetInput(Inp.I1)) {
      ft.SendMessage(ref outN);
      ft.SetLamp(Out.O1, Dir.Aus);
      ft.SetLamp(Out.O2, Dir.Ein);
      lblStatus.Text = "Schaulaufen der Raupe";
      ft.WaitForMessage(0, out inN);
      ft.WaitForMessage(0, out inN);
      ft.SetLamp(Out.O1, Dir.Ein);
      ft.SetLamp(Out.O2, Dir.Aus);
      lblStatus.Text = "Warten auf neuen Auftrag : I1";
    }
  } while (!ft.Finish());
}
```

Gezeigt wird das komplette Betriebsprogramm, das DrumRum fehlt.

Ablauf :

- Deklarieren der Nachrichten Strukturen inN und outN, outN wird mit Leerwerten besetzt
- Einschalten der grünen Lampe
- Endloschleife (Ende durch HALT-Button oder ESC-Taste)
- Abfrage ob I1 gedrückt, Ready von RF 2/2 wird vorausgesetzt.
- I1 = true : Senden einer Nachricht zum Start des Schaulaufens der Raupe
  Rote Lampe an
- RF 2/2 sendet an RF 2/0 die Nachricht Busy und nach beenden des Schaulaufen Ready. Das wird hier ohne Prüfung des Inhaltes der Nachricht (man ist ja unter sich) abgewartet.
- Dann wieder Lampe grün
- Und das immer wieder

Source-Liste FactoryRaupe.c

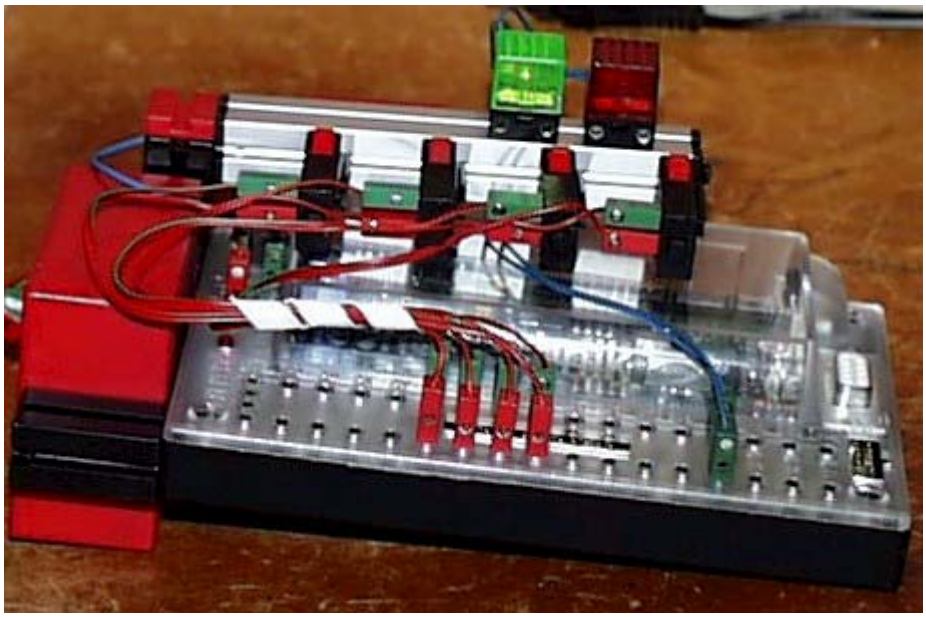
Ein Renesas C Programm, das im Prozessor des Raupen-Interfaces RF 2/2 läuft :

```
void Schaulaufen() {
  sTrans.M_Main = 0x05;
  FtDelay(888);
  sTrans.M_Main = 0x0A;
  FtDelay(888);
  sTrans.M_Main = 0x06;
  FtDelay(888);
  sTrans.M_Main = 0x09;
  FtDelay(888);
  sTrans.M_Main = 0x00;
}

UCHAR main(void) {
  SMESSAGE inMessage, outMessage;
  UCHAR    res, i;

  InitMessage();
  SetFtMessageReceiveAddress((void far*)&WriteMessageToBuffer);
  sTrans.MPWM_Update = 0x01;
  for(i = 0; i<8; i++) sTrans.MPWM_Main[i] = 7;
  do {
    if(GetMessageFromBuffer(&inMessage) == ERROR_SUCCESS) {
      outMessage.W.uiMsgId = 17;
      outMessage.W.uiMsg   = 0x00;
      res = SendFtMessage(MSG_HWID_RF, 2, outMessage.L.ulMsg, 55);
      Schaulaufen();
      outMessage.W.uiMsgId = 18;
      outMessage.W.uiMsg   = 0x01;
      res = SendFtMessage(MSG_HWID_RF, 2, outMessage.L.ulMsg, 55);
    }
  } while(1);
  return(0);
}
```

Das main Programm wartet in einer Endlos-Schleife auf das Eintreffen (irgendeiner) Nachricht (GetMessageFromBuffer) und quittiert das durch ein SendFtMessage Busy. Anschließend wird das Schaulaufen gestartet, nach dessen Ende wird ein Ready Nachricht gesendet.

Die Routine Schaulaufen macht nichts weiter als die beiden Motoren der Raupe laufen zu lassen : Vor, Rück, Rechtsdrehen, Linksdrehen, Aus.

Das PC-Programm : FunkRaupe4.CS



Der Programmrahmen ist wie bei FunkRaupe.CS. Wenn bei der FunkRaupe durch I1 ein Schaulaufen ausgelöst wurde, hat hier RF 2/1 (jetzt mit vier Tastern an I1 – I4) eine Fernbedienung a la IR-Sender programmiert. Die Raupe verzichtet aufs Schaulaufen und fährt nur noch in die vorgegebene Richtung. Auf eine Quittierung des Auftrags wird verzichtet.

Die Source-Liste FunkRaupe4.CS

```
    private void Action() {
      MessageData outCommand;
      outCommand.HwId  = 2;
      outCommand.SubId = 1;
      outCommand.MsgId = 0x01;
      outCommand.Msg = cAus;
      ft.SetLamp(Out.O1, Dir.Ein);
      lblStatus.Text = "Ready, waiting for your orders";

      do {
        if (ft.GetInputs() != 0) {
          if (ft.GetInput(Inp.I1)) {
            if (outCommand.Msg == cVor) outCommand.Msg = cAus;
            else outCommand.Msg = cVor;
          }
          else if (ft.GetInput(Inp.I2)) {
            if (outCommand.Msg == cLinks) outCommand.Msg = cAus;
            else outCommand.Msg = cLinks;
          }
          else if (ft.GetInput(Inp.I3)) {
            if (outCommand.Msg == cRechts) outCommand.Msg = cAus;
            else outCommand.Msg = cRechts;
          }
          else if (ft.GetInput(Inp.I4)) {
            if (outCommand.Msg == cRueck) outCommand.Msg = cAus;
            else outCommand.Msg = cRueck;
          }
          ft.SendMessage(ref outCommand);
```

```
          if (outCommand.Msg != cAus) {
            ft.SetLamp(Out.O1, Dir.Aus);
            ft.SetLamp(Out.O2, Dir.Ein);
            lblStatus.Text = "--- Busy ---";
          }
          else {
            ft.SetLamp(Out.O1, Dir.Ein);
            ft.SetLamp(Out.O2, Dir.Aus);
            lblStatus.Text = "--- Ready ---";
          }
        }
        ft.Pause(555);
      } while (!ft.Finish());
    }
```

Gezeigt wird wieder das komplette Betriebsprogramm, das DrumRum fehlt, Ablauf :

- Deklarieren der ausgehenden Nachricht und Setzen Grüne Lampe : Ready.
- Endlos-Schleife (Ende durch HALT-Button oder ESC-Taste)
- Abfrage, ob irgendeine Taste gedrückt ist.
- Wenn ja : Wenn gleiche Taste wie in der letzten Runde : Abschalten der Motoren
  Sonst : Schalten in die durch Taster vorgegebene Richtung.
- Senden der Nachricht und Schalten der Status-Lampen.
- Pause damit man die Taste in Ruhe wieder loslassen kann.

Die Konstanten im Text sind die Fahrbefehle für beide Motoren, die an die Raupe gesendet werden.

Source-Liste FunkPoorSlave.C

Ein Renesas C Programm, das im Prozessor des Raupen Interfaces RF 2/2 läuft :

```
UCHAR main(void) {
  SMESSAGE inMessage;
  UCHAR res;
  UCHAR i;

  InitMessage();
  SetFtMessageReceiveAddress((void far*)&WriteMessageToBuffer);
  sTrans.MPWM_Update = 0x01;
  for(i = 0; i<8; i++) sTrans.MPWM_Main[i] = 7;
  do {
    if(GetMessageFromBuffer(&inMessage) == ERROR_SUCCESS) {
      sTrans.M_Main = inMessage.B.ucB3;
      FtDelay(111);
    }
  } while(1);
  return(0);
}
```

Das Programm in RF 2/2 ist deutlich geschrumpft. Es gibt nur noch eine Schleife in der unkontrolliert eine Nachricht entgegengenommen wird und das entscheidende Byte daraus an die TranferArea weitergereicht wird um die Motoren der Raupe zu schalten.

Betrieb eines Robots

Die Klasse FishRobot ist speziell auf den Betrieb von Robot-Motoren ausgerichtet. Als Robot-Motor wird ein Motor dann bezeichnet, wenn auf einer Motorwelle ein Impulsrad mitläuft, das einen Taster betätigt über den die Umdrehungen der Motorwelle in Form von Impulsen gezählt werden. Hinzu kommt ein Endtaster zur Bestimmung der Home-Position. Endtaster und Impulstaster sind dem jeweiligen M-Ausgang fest zugeordnet. Außerdem kann der max. Fahrweg vorgegeben werden (s.a. Anmerkungen zu den Rob-Funktionen). Es können bis zu vier (mit Extension Module bis zu acht, ROBO 16) Motoren simultan (gleichzeitig) betrieben werden.

Die Testroutine ist eine Windows.Form, sie entspricht der im Kapitel Programmrahmen angeführten, die Instanzierung wird auf FishRobot umgestellt.

Robot-Fahren

Das Robot-Fahren geschieht über die Methoden MoveTo und MoveDelta. Als Parameter enthalten sie eine Liste der anzufahrenden Positionen (absolut von Home oder relativ zur aktuellen Position). Die Liste der zugehörenden M-Ausgänge und die Begrenzung wird bei der Instanzierung festgelegt. Die Home-Position (die Position an den Endtastern) wird mit MoveHome angefahren. Die Motoren müssen so gepolt sein, daß sie linksdrehen (mit Dir.Links den zugehörenden Endtaster anfahren). Bei Erreichen der Endposition werden die aktuellen Motorpositionen auf 0 gesetzt.

```
private FishRobot ft = new FishRobot(new int[,]{{3,222},{4,88}});
.......
private void cmdAction_Click(object sender, System.EventArgs e){
  try {
    ft.OpenInterface(IFTypen.ftROBO_first_IF_USB, 0);
     lblStatus.Text = "--- gestartet ---";
     ft.MoveHome();
     lblStatus.Text = "M3 auf Pos : " +
                              ft.MotCntl[0].actPos.ToString();
     ft.Pause(1234);
     ft.MoveTo(23, 34);
     ft.MoveDelta(-13, 6);
     ft.MoveTo(50,80);
   }
   catch(FishFaceException eft) {
     lblStatus.Text = eft.Message;
   }
   finally {
     ft.CloseInterface();
   }
}
```

Bei der Instanzierung wird die Roboter-Konfiguration festgelegt : Motoren an M3 und M4 mit Fahrwegbegrenzung auf 222 bzw. 88 Impulse.

Nach dem OpenInterface wird die Home-Position angefahren und die aktuelle Position auf 0 gesetzt. Das wird kurz angezeigt.

Anschließend wird auf die Position M3 = 23 und M4 = 34 gefahren. Dann wird M3 um 13 Impulse zurück auf Position 10 und M4 um 6 Positionen vor auf Position 40 gefahren. Zum Schluß wird die Position 50/80 angefahren.

Beim Erproben mit dem Projekt FishRobotWindows sind nur noch die Statements ab ft.MoveHome(); ... ft.MoveTo(50,80), in Methode Action() einzutragen.

Positionsanzeige

Die aktuelle Position kann nach einer Move-Methode den entsprechenden Werten von MotCntl entnommen werden, wie im vorhergehenden Beispiel geschehen. Die aktuelle Position kann aber auch während der Ausführung der Methoden MoveTo/MoveDelta über eine Ereignis-Routine angezeigt werden.

Dazu ist nach der Instanzierung (am besten noch im Konstruktor der Form-Klasse) eine entsprechende Routine anzumelden :

```
ft.PositionChange += new FishRobot.CommonDelegate(PositionsAusgabe);
```

In die Liste des von MoveTo/MoveDelta ausgelösten Ereignisses PositionChange wird der Delegate CommonDelegate mit dem Namen der zugehörenden Ereignisroutine (PositionsAusgabe) eingetragen :

```
private void PositionsAusgabe(object sender, int[] actPos) {
  lblStatus.Text = "Position : " + actPos[0].ToString() + " - " +
         actPos[1].ToString();
}
```

Die Ereignisroutine zeigt die aktuelle Position der beiden Motoren in lblStatus an. Natürlich könnten hier auch noch weitere Aufgaben wahrgenommen werden.

Betrieb von Schrittmotoren

Die Klasse FishStep ist von FishFace abgeleitet und unterstützt zusätzlich besonders den Einsatz von Schrittmotoren. Mit den Methoden StepHome / StepTo / StepDelta werden einzelne Schrittmotoren, die an zwei aufeinander folgende M-Ausgänge angeschlossen sind, unterstützt. Und mit den Methoden PlotHome / PlotTo / PlotDelta der Betrieb von zwei Schrittmotoren im XY-Verbund unterstützt. Die Schrittmotoren belegen 3 aufeinanderfolgende M-Ausgänge. Jedem Schrittmotor ist ein Endtaster fest zugeordnet.

Die hier verwendete Testroutine ist eine einfache Windows.Form, sie entspricht der im Kapitel Programmrahmen angeführten, die Instanzierung wird auf FishStep umgestellt.

Einzelner Schrittmotor

Beispiel : Fahrstuhl aus einem Schrittmotor mit einer (langen) Schneckenwelle senkrecht nach oben und einem "Korb" an der Schneckenmutter. Der Endtaster liegt auf E1.

```
private const int mFahrstuhl = 1;
private FishStep  ft = new FishStep(new int[,]{{mFahrstuhl,456}});
......
ft.StepChange += new FishStep.StepDelegate(PositionsAusgabe);
......
private void cmdAction_Click(object sender, System.EventArgs e) {
   try {
      ft.OpenInterface(IFTypen.ftROBO_first_IF_USB, 0);
      lblStatus.Text = "--- fährt auf Home (Keller) ---";
      ft.StepHome(mFahrstuhl);
      lblStatus.Text = "--- fährt auf Etage 4 ---";
      ft.StepTo(mFahrstuhl, 200);
      ft.Pause(1234);
      lblStatus.Text = "--- fährt eine Etage tiefer ---";
      ft.StepDelta(mFahrstuhl, -50);
      lblStatus.Text = "Das war's : Die Rufknöpfe nachrüsten";
   }
   catch(FishFaceException eft) {
      lblStatus.Text = eft.Message;
   }
   finally{
      ft.CloseInterface();
   }
}
```

Und die Routine zur Positionsausgabe :

```
private void PositionsAusgabe(object sender, int MotNr, int actPos) {
  lblStatus.Text = "Fahrstuhl auf Etage : " +  _
                   (actPos/50).ToString()+
                    " | " + (actPos%50).ToString();
}
```

Bei der Instanzierung wird der Fahrstuhlmotor mFahrstuhl (M1/M2) mit einem max. Fahrweg von 456 Zyklen der Instanz zugeordnet.

Im Konstruktor der Form-Klasse wird noch die Routine für die PositionsAusgabe in die Ereignisliste von StepChange eingeklinkt.

In der Klick-Routine für den ACTION-Button läuft das Steuer-Programm :

- Nach OpenInterface : Anfahren der Home Position (StepHome)
- Fahren zu Etage 4 (StepTo)
- Eine Etage tiefer mit StepDelta
- Laufende Anzeige der Position mit Routine PositionsAusgabe.

Zwei Motoren im XY-Verbund : Plotten

Plotter mit zwei Schrittmotoren an M1 – M3 und den Endtastern I1 und I5. Für Testzwecke reichen die nackten Motoren mit Scheibenrädern drauf, damit man etwas sehen kann.

Instanzierung :

```
private const int mPlotter = 1;
private FishStep  ft = new FishStep(new
                                    int[,]{{mPlotter,456},{3,456}});
.....
```

Zuordnung der Ereignisroutine zur PostionsAusgabe :

```
ft.PlotChange += new FishStep.PlotDelegate(PositionsAusgabe);
......
```

Das Steuer-Programm in der ACTION-Button Klickroutine :

```
private void cmdAction_Click(object sender, System.EventArgs e) {
   try  {
      ft.OpenInterface(IFTypen.ftROBO_first_IF_USB, 0);
      lblStatus.Text = "--- fährt auf Home (linke untere Ecke) ---";
      ft.PlotHome(mPlotter);
      lblStatus.Text = "--- Anfahren 50/50 ----";
      ft.PlotTo(mPlotter, 50, 50);
      lblStatus.Text = "--- Zeichnen eines Quadrats ---";
      Vieleck(new int[,] {{50,0},{0,50},{-50,0},{0,-50}});
      lblStatus.Text = "--- Das war's ---";
   }
   catch(FishFaceException eft) {
      lblStatus.Text = eft.Message;
   }
   finally {
      ft.CloseInterface();
   }
}
```

Die laufende Positionsanzeige :

```
private void PositionsAusgabe(object sender, int MotNr,
                                          int xPos, int yPos)  {
   lblStatus.Text = "Position X/Y : " + xPos.ToString()+
                                            " / " + yPos.ToString();
}
```

Zeichnen des Quadrats :

```
private void Vieleck(int[,] RelList) {
   for(int i = 0; i < RelList.GetLength(0); i++)
          ft.PlotDelta(mPlotter, RelList[i,0], RelList[i,1]);
}
```

Diesmal sind die Kommentare in der Source und der zugehörende Beispielrahmen in \Templates\FishPlotWindows, dort sind nur noch die Plot-Methoden einzutragen.

Anmerkungen zu den Interfaces

Übersicht

Zur ROBO Serie gehören folgende Komponenten :

- **ROBO Interface**
  Anschluß über USB oder COM. Bei aufgesteckter RF-Karte auch über Funk.
  Bei Anschluß über COM wahlweise ROBO oder Intelligent Interface Betriebs-Mode
  Download-Betrieb über ROBO Pro und Renesas C, wenn die ROBO Interfaces mit RF-Karte ausgerüstet sind auch über Funk.
  An das ROBO Interface können bis zu drei ROBO Extensions angeschlossen werden.
- **ROBO IO Extension**
  Anschluß über USB an den PC oder über ein Sonderkabel direkt an ein ROBO Interface.
  Es können dann noch zwei weitere über das Extension Module selber "durchgeschleift" werden.
- **ROBO RF-Datalink**
  Anschluß an den PC über USB
  - Zum Betrieb eines zugeordneten ROBO Interfaces über Funk (Route Through),
  - Als Message-Router zur Koordination des Funkverkehrs mehrer ROBO Interfaces.
  - Zusätzlich in einer Kombination von beidem. Dabei muß immer ein ROBO Interface im Route Through laufen.
- **ROBO Connect Box**
  Anschluß an den PC über USB
  – Zum Betrieb eines (ganz alten) Universal Interface mit Centronics-Schnittstelle. Der Betrieb eines Slaves am Interface wird nicht unterstützt.
  – Ein Anschluß des Intelligent Interfaces ist ebenfalls möglich (wenn z.B. am Rechner keine seriellen Anschlüsse vorhanden sind.
  Das jeweils angeschlossene Interface stellt sich dem Programm gegenüber wie ein ROBO Interface (mit eingeschränkten Möglichkeiten) dar.

Zu den älteren Intelligent Interface gehören zwei Komponenten :

- **Intelligent Interface**
  Anschluß über COM, Download-Betrieb über LLWin
- **Extension Module**
  Anschluß an das Intelligent Interface

Die Interfaces (nicht RF-Datalink) sind mit 4 Motorausgängen ausgestattet, die bei Bedarf einzeln in "Lampen"-Ausgänge geteilt werden können. Hinzu kommen 8 digitale Eingänge und einige Analog-Eingänge. Weitere Details siehe www.ftcomputing.de/ftdetail.htm

Interface Anschluß

Die Verbindung zwischen PC-Programm und Interface wird über die FishFace.Methode `OpenInterface` hergestellt. Dabei wird als Parameter der Interface-Typ (Siehe auch enum IFTypen) angegeben, davon abhängig noch weitere Parameter :

- ft**ROBO_first_USB** (0)
  Erstes ROBO Device an USB, Default-Seriennummer = 0
- ft**Intelligent_IF** (10)
  Intelligentes Interface, Anschluß an wählbaren COM-Port
- ft**Intelligent_IF_Slave** (20)
  Intelligentes Interface mit Extension Module, Anschluß an wählbaren COM-Port
- ft**ROBO_IF_IIM** (50)
  ROBO Interface an wählbaren COM-Port, das Intelligent Interface wird simuliert
- ft**ROBO_IF_USB** (60)
  ROBO Interface an USB, Angabe der Seriennummer ist erforderlich
- ft**ROBO_IF_COM** (70)
  ROBO Interface am wählbaren COM-Port im Native Mode
- rf**ROBO_IF_Over_RF** (80)
  interner Typ, wenn ein ROBO Interface mit Funk-Platine über ein ROBO RF Datalink betrieben wird
- ft**ROBO_IO_Extension** (90)
  ROBO IO Extension Module direkt an USB, Angabe der Seriennummer erforderlich
- ft**ROBO_RF_Datalink** (110)
  ROBO RF Datalink an USB, Angabe der Seriennummer erforderlich
- **ftROBO_Connect_Box**(200)
  ROBO Connect Box.

Bei dem ROBO Devices, die an USB angeschlossen werden sollen, reicht die Angabe ftROBO_first_USB, wenn nur ein ROBO Device an USB angeschlossen wurde. Der aktuell angeschlossene Device-Typ kann dann, bei Bedarf, vom Programm über die FishFace.Eigenschaft ActDevice abgefragt werden.

Sensoren am ROBO Interface

Neben den einfache Sensoren an I-Eingängen (Taster, Phototransistor ..) ... sind mit dem Kasten ROBO Explorer komplexere Sensoren ins Spiel gekommen deren Anschluß und Einsatz nicht mehr so einfach ist :

Distanzsensor

Anschluß Kabel rot/grün an D1 / D2, Polung beliebig.
Betrieb mit GetDistance, WaitForGreater, WaitForLess, OpenInterface(... Distance.UltraSonic ..) ist erforderlich.

Farbsensor

Anschluß Kabel rot/grün an +/Masse, schwarz an A1/A2 (einpolig, innen).
Betrieb : GetVoltage.

Spursensor

Anschluß Kabel rot/grün an +/Masse, gelb/blau an zwei verschiedene I-Eingänge (einpolig, innen). Auswertung entsprechend der Farbzuordnung am Sensor.
Betrieb : GetInput / GetInputs

Interface Konfigurationen

In den vielen Fällen wird nur ein Interface vorhanden sein, dann ist die Lage ganz einfach : Interface an USB oder COM anschließen und durch OpenInterface beim Programm anmelden. Es können aber auch deutlich umfangreichere Konfigurationen zusammengestellt werden :

1. **PC** – (USB / COM) – **Interface**, ggf. mit Extensions
   Betrieb durch ein PC-Programm (C# - FishFace2005.DLL, ROBO Pro)
2. **PC** – (USB) – **ROBO I/O Extension**
   Betrieb durch ein PC-Programm (C# - FishFace2005.DLL, ROBO Pro)
3. **PC** : Simultaner Betrieb weiterer Interfaces nach **(1)**, **(2)** im gleichen Programm
4. **PC** – (USB) – **ROBO RF Datalink** (RF 2/0) – (Funk)
   – **ROBO Interface mit RF Platine** (RF 2/x), ggf. auch mit Extensions
   Betrieb durch ein PC-Programm (C# - FishFace2005.DLL, ROBO Pro)
   Das Interface ist dem RF Datalink fest zugeordnet. Das PC-Programm sieht nur das Interface.
5. **PC** : Simultaner Betrieb **weiterer Devices** nach (1), (2) oder (4)
6. **Interface**, ggf. mit Extensions
   Betrieb über **im Interface gespeichertes Programm**
   Erstellung ROBO Interface : ROBO Pro oder Renesas C
   Intelligent Interface : LLWin
7. **Mehrere Interfaces** nach (6), die Interfaces sind zusätzlich mit jeweils einer **RF Platine** ausgerüstet. Es ist dann zusätzlich eine **Kommunikation** untereinander über Funk möglich. Die Interfaces werden durch ihre RF Nummer (RF 2/1 ... RF 2/8) identifiziert. Zusätzlich ist als Message Router ein ROBO RF Datalink (RF 2/0) erforderlich, er muß zur Stromversorgung an USB angeschlossen sein, benötigt aber kein PC-Programm
8. **PC mit Datalink nach** (4) und **Interfaces nach (7)**
   Zusätzlich ist jetzt ein Funkbetrieb zwischen dem PC-Programm und den Interfaces nach (7) möglich.

Die hier angegebene Frequenz 2 ist die voreingestellte. Das RF Datalink wird mit RF 2/0 ausgeliefert, die RF Platinen der Intefaces mit RF 2/1. Die Frequenz kann mit ROBO Pro im Bereich 2 .. 80 eingestellt werden und die Subkanäle der RF Platinen im Bereich 1..8.
Ein parallel Betrieb auf mehreren Frequenzen ist möglich.

Treiber, Firmware und DLLs

Die FishFace-Software baut auf der FtLib von fischertechnik auf. Dazu gehören die USB-Treiber und die in den ROBO Devices installierte Firmware. Sie wird nicht mit den FishFace-Downloads ausgeliefert. Die einfachste Art sie zu bekommen ist der Kauf und die Installation von ROBO Pro (ca. 20€). Damit erhält man gleichzeitig (aus Sicht C# 2005) ein schönes Rapid-Prototyping Tool. www.fischertechnik.de/robopro/update.html . ROBO Pro enthält auch Möglichkeiten zum Update der Firmware in den ROBO Devices, außerdem kann man mit ROBO Pro Seriennummer und Funkfrequenzen der Geräte einstellen.

COM

Keine Treiber erforderlich

USB

ftusb.inf und ftusb.sys werden bei Anschluß eines ROBO Devices automatisch installiert. Der aktuelle Stand ist in Treiber Software FTLIB enthalten :
www.fischertechnik.de/computing/download/zip/2005-05-FtLib.zip

Kommt mit ROBO Pro

Firmware

Die ROBO Devices (Interface, Extension, Datalink) benötigen zum Betrieb Firmware sie ist ebenfalls in FTLIB enthalten :

www.fischertechnik.de/computing/download/zip/2005-05-FtLib.zip

außerdem kommt sie mit den ROBO Pro Updates.

FishFace2005.DLL / umFish40.DLL (25.04.06) nutzen :

- ROBO Interface : RoboIf_Ver_01_64_00_03.FW1
- ROBO I/O Extension : Ft_RoboExt_Ver_00_12_03.FW1
- ROBO RF Datalink : RfDatLink_Ver_00_44_00_03.FW1

Update Firmware mit ROBO Pro

1. ROBO Interface an USB (ggf. COM) anschließen, sonst keine USB Devices
2. ROBO Pro Werkzeugleiste Test-Symbol aufrufen, Info-Tab einstellen
3. Kasten Firmware aktualisieren : Anweisungen folgen.
4. Wenn RF-Platine auf dem Interface : nochmal für diese.
5. Bei Bedarf Seriennummer einstellen. Am einfachsten ist es seine ROBO Devices laufend durch zu numerieren. Ist nur ein Interface vorhanden, nicht erforderlich
6. Wenn RF-Platine vorhanden, USB-Funk einstellen. Die Frequenz 2 kann so bleiben (wenn man nicht mehrere Funkkreise betreiben will). Beim ersten Interface stellt man am besten auf RF 2/1 ein, wenn man nicht mehrere gleichzeitig im Funk-Verbund nutzen will, kann man das auch bei den weiteren tun, sonst muß die Funkruf nummer im Verbund eindeutig sein (1..8).
7. ROBO RF Datalink nach gleichem Schema. Standard RF 2/0, Funkrufnummer ist immer 0. Keine besondere RF-Platine vorhanden.
8. ROBO I/O Extensions : Firmware Update, keine RF-Platine möglich, Seriennummer

umFish40.DLL

Der Zugriff auf das Interface erfolgt indirekt über eine FtLib von fischertechnik(in FishFace40/umFish40.DLL integriert, z.Zt. (25.04.06) ist das die Version 0.59), die in regelmäßigen Abständen die Werte des Interface ausliest und gleichzeitig den Status der M-Ausgänge setzt (schaltet).

Die ausgelesenen Werte werden in einem internen Kontrollblock abgestellt bzw.die Werte für die M-Ausgänge werden dort entnommen. Der Kontrollblock enthält darüberhinaus alle Werte die für den Betrieb eines Interfaces erforderlich sind. Ein Parallel-Betrieb mehrerer Interfaces (z.B. eins an USB, ein weiteres an COM1) ist somit möglich.

Zusätzlich werden die Impulse an den I-Eingängen gezählt (Veränderung am true/false-Status eines Einganges, in umFish40.DLL), die Geschwindigkeitssteuerung (durch zyklisches Ein/Ausschalten der M-Ausgänge - PWM) und im RobMode das Abschalten eines M-Ausganges, wenn der zugehörende Impuls-Counter den Wert null erreicht hat.

Die angebotenen Zugriffsfunktionen sind ein Mix aus Notwendigkeit und Komfort. Open/CloseInterface stellen die Verbindung zum Interface her und beenden sie. Die GetInput-Funktion liest lediglich den Wert für einen I-Eingang aus einem Kontrollblock. Dabei wird das zutreffend bit maskiert, ähnliches gilt für SetMotor und SetLamp in der Gegenrichtung. Es erfolgt auch hier keine direkte Ansteuerung des Interfaces.

SetMotor(s) arbeiten nur mit dem Kontrollblock zusammen, führen aber (über das reine Setzen der M-Ausgänge hinaus) etwas komplexere Operationen aus.

Zusätzlich stehen besondere Funktionen für den Funk-Betrieb zur Verfügung

FishFace2005.DLL

Die C# 2005 Assembly FishFace2005.DLL mit dem Namensraum FishFace40 setzt auf der umFish40.DLL auf und unterstützt die ROBO Devices und das Intelligent Interface im Online-Betrieb. Zusätzlich werden Methoden für den Funk-Betrieb angeboten.

Sie enhält die Klassen FishFace, FishRobot:FishFace und FishStep:FishFace sowie die Klasse FishFaceException. Außerdem enthält sie eine Reihe von enum's zur Bezeichnung der Parameter der Methoden.

FishFace2005.DLL basiert auf dem Source-File FishFace2005.cs das alle Klassen enthält. Da XML-Kommentare für die Anzeigen von erklärenden Texte in ObjectBrowser und IntelliSense genutzt werden, ist für die compilierte DLL noch ein daraus generiertes FishFace2005.xml erforderlich, das im Verzeichnis der DLL liegen muß.

Counter, Geschwindigkeit, RobMotoren und Stepper

Die Counter - Impulstaster

Ein wesentliches Element zur Positionsbestimmung sind die Counter. Sie sind den I-Eingängen zugeordnet. In den Countern wird von einer zentralen Routine in umFish40.DLL jede Veränderung des Zustandes der I-Eingänge gezählt. Also z.B. das Öffnen oder auch das Schließen eines Tasters, der z.B. durch ein Impulsrad betätigt wird.

Die Counter sind Teil eines internen Kontrollblocks. Sie können mit entsprechenden Methoden gesetzt und abgefragt werden. Die Counter werden auch intern von einigen Funktionen/Methoden (z.B. SetMotor mit Parameter Counter und den meisten Wait-Methoden) genutzt, es kann also nicht damit gerechnet werden, daß sie über den Programmablauf Bestand haben.

Geschwindigkeitssteuerung

Die Geschwindigkeitssteuerung beruht auf einem zyklischen Ein- und Ausschalten der betroffenen M-Ausgänge (Motoren). Dazu wird intern für jede Geschwindigkeitsstufe eine entsprechende Schaltliste vorgehalten. Die Geschwindigkeit wird durch den Parameter Speed für einen Motor und den Parameter SpeedStatus für alle Motoren angewählt. Sie geschieht über FtLib.

Rob-Funktionen - RobMotoren

Hier werden allgemeine Anmerkungen zu den Rob-Funktionen und deren Nutzung durch Methoden der Klasse FishFace gemacht. Die spezielle Klasse FishRobot wird separat beschrieben.

Die Rob-Funktionen laufen in einem besonderen Betriebsmodus, dem RobMode. In diesem Modus werden die betroffenen Counter decrementiert. Bei Erreichen des Wertes 0 wird der betroffen Motor abgeschaltet. Gelegentlich kann es vorkommen, daß noch um einen Impuls weiter gefahren wird. Das kann man durch Abfrage des entsprechenden ImpulsCounters (wert > 0) feststellen und bei der Speicherung der aktuellen Position entsprechend berücksichtigen.

Der Betrieb eines Motors mit den Rob-Funktionen setzt ein festes Anschlußkonzept voraus. Zum jeweiligen Motor gehören je ein Impulstaster und ein Endtaster. Dazu folgende Tabelle :

Motor	Endtaster	Impulstaster
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10
6	11	12
7	13	14
8	15	16

Und so weiter bis Motor 16, wenn entsprechend viele Extensions angeschlossen sind.

Die Motoren sind „linksdrehend“ d.h. sie drehen bei Dir.Links in Richtung Endtaster.

Die Motoren können einzeln über SetMotor oder alle gemeinsam über SetMotors geschaltet werden. Das Argument Counter gibt die Anzahl der zu fahrenden Impulse an.Die Argumente

ActPosition und ZielPosition beschreiben den Fahrauftrag. GetCounter /SetCounter greifen direkt auf den intern verwendeten Counter zu.

Die Motoren können auch alle mit einem Befehl geschaltet werden : SetMotors. Dazu müssen vorher die Parameter aufbereitetet werden.
MotorStatus : pro Motor 2bit, mit M1 : bit 0 und 1 beginnend.
00 : aus, 01 links, 10 rechts.
SpeedStatus : pro Motor 4bit, mit M1 : bit 0-3 beginnend,
0000 aus, 1000 halbe Kraft, 11111 voll. Ab M9 in SpeedStatus16.
ModeStatus : proMotor 2 bit, mit M1 : bit 0-1 beginnend,
00 Normal-Mode, 01 Rob-Mode. Der Rest z.Zt. nicht besetzt
(vorgesehen z.B. für Schrittmotorenbetrieb).

Beispiel : SetMotors(0x9, 0x74, 0x0, 0x5);
0x steht für Hexa, binär : MotorStatus 1001, SpeedStatus 0111 0100 ModeStatus 0101 -> M2 = rechts, Speed im Rob-Mode, M1 = links, Speed 4 im RobMode. Der Rest steht. Die zugehörenden Counter sind vorher mit SetCounter auf die gewünschte Fahrstrecke zu setzen.

Direction = 0 bzw. die Angabe im MotorStatus hält den Motor unabhängig von den Speed-Werten an.

Die Motoren laufen simultan (ggf. auch alle 16 - sechzehn), sie können der Reihe nach mit SetMotor geschaltet werden. Sie starten dann beim nächsten Abfragezyklus automatisch und laufen asynchron (d.h. unabhängig von den Aktionen des rufenden Programms) bis sie die vorgegebene Position erreicht haben. Sie werden dann ebenfalls einzeln abgeschaltet.

Um Festzustellen, ob die Motoren ihr Ziel erreicht haben und um das Programm mit den durch durch die Rob-Funktionen ausgelösten Aktionen wieder zu synchronisieren ist ein WaitForRobMotor(s) erforderlich.

Step-Funktionen - Schrittmotoren

Der Betrieb von Schrittmotoren über ein fischrtechnik Interface ist möglich. Dazu ist die Klasse FishStep der Assembly FishFace40.DLL vorgesehen.

Schrittmotoren können mit FishStep **einzeln** oder im **XY-Verbund** paarweise betrieben werden. In beiden Fällen werden sie synchron betrieben, d.h. das Programm wartet, bis die vorgegebene Position erreicht ist. Im Falle des XY-Verbundes werden die beiden dazugehörenden Schrittmotoren simultan (gleichzeitig) betrieben.

Die Schrittmotoren erfordern zum Anschluß zwei aufeinanderfolgende M-Ausgänge (Einzel-Motoren) bzw. drei aufeinanderfolgende M-Ausgänge (XY-Verbund). Die M-Ausgänge können sich über Master und Slave (Extension Module) erstrecken. Der Betrieb erfolgt in Zyklen zu vier Schritten. Ein Zyklus ist damit auch die Positioniereinheit für einen Motor. Da die Motoren pro Schritt eine 7,5° Drehung machen, ergeben 48 Schritte eine volle Umdrehung. Das entspricht dann 12 Zyklen.

Diese Betriebsart wurde besonders in Hinblick auf die beschränkte Anzahl von M-Ausgängen am Interface gewählt. So ist ein Plotterbetrieb mit nur einem (Master) Interface möglich. Der freie M-Ausgang wird hier für den Stift-Antrieb genutzt. Da führt zu einem "Zittern" des Motors, der gerade nichts zu tun hat (Vor-/Rückschritt im Wechsel). Diese Zittern stört den Betrieb aber nicht weiter, da es von dem üblichen Spiel (Schneckenantrieb) des Modells aufgefangen wird.

Zeiten

Bei Schrittmotoren werden häufig Schnecken zum Modellantrieb genutzt. Die 'große' Schnecke hat eine Steigung von ca. 4,77 mm. d.h. bei einer Umdrehung der Motorwelle (mit der aufgesteckten Schnecke) legt die Schneckenmutter einen Weg von 4,77 mm zurück. Bei 12 Zyklen/Umdrehung sind das pro Zyklus 0,4 mm.

Bei einem Win2000 Rechner mit 1700 MHz und einer Zykluszeit von 10 MilliSekunden ergeben sich folgenden Zeiten und Geschwindigkeiten :

200 Zyklen : 12,5 Sekunden. Pro Zyklus also 62.5 MilliSekunden.

100 mm Weg entsprechen 250 Zyklen. Das sind dann ca. 15 Sekunden für die 100 mm.

Anschluß der Schrittmotoren

Die verwendeten Schrittmotoren haben vier Kabelanschlüsse : rot, grün, schwarz, grau.

Zwei Schrittmotoren im XY-Verbund

		vorn	hinten
Motor X-Achse	Ma	rot	schwarz
	Mb	grün	grau
Motor Y-Achse	Ma	rot	schwarz
	Mc	grün	grau

vorn heißt die Stiftreihe an der Außenkante.
Mit Ma-Mc sind aufeinanderfolgende M-Ausgänge gemeint.
z.B. M1-M3. Aber M4-M6 sind auch möglich.
Die Motoren drehen bei PlotHome in Richtung 0 auf den zugehörenden Endtaster (Schließer, Kontakte 1 und 3). Für X ist der zu Ma gehörende, für Y der zu Mc gehörende
z.B. M1 : I1, M3 : I5. (Alle von M1 : I1, I3, I5, I7, I9, I11, I13, I15 (für M8)).

Ein einzelner Schrittmotor

		vorn	hinten
Motor A	Ma	rot	schwarz
	Mb	grün	grau

vorn heißt die Stiftreihe an der Außenkante.
Mit Ma-Mb sind aufeinanderfolgende M-Ausgänge gemeint.
z.B. M1-M2. Aber M4-M5 sind auch möglich.
Der Motor dreht bei StepHome in Richtung 0 auf den zugehörenden Endtaster (Schließer, Kontakte 1 und 3). Das ist der zu Ma gehörende.
z.B. M1 : I1 (Alle von M1 : I1, I3, I5, I7, I9, I11, I13, I15 (für M8)).

Anmerkungen zu C#

Programmrahmen

Aufbau eines ftComputing-Programms

1. try – catch(FishFaceException e) (- finally) Block
   sollte die gesamt Anwendung umfassen. Wenn mehr Detailierung erforderlich ist, natürlich mehr, ggf. auch weitere catch – Klauseln.
2. OpenInterface – CloseInterface
   umschließt die Anwendung. Häufig reicht der feste Deviceeintrage ftROBO_first_USB, 0 für das erste ROBO Interface an USB.
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(!ft.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 :

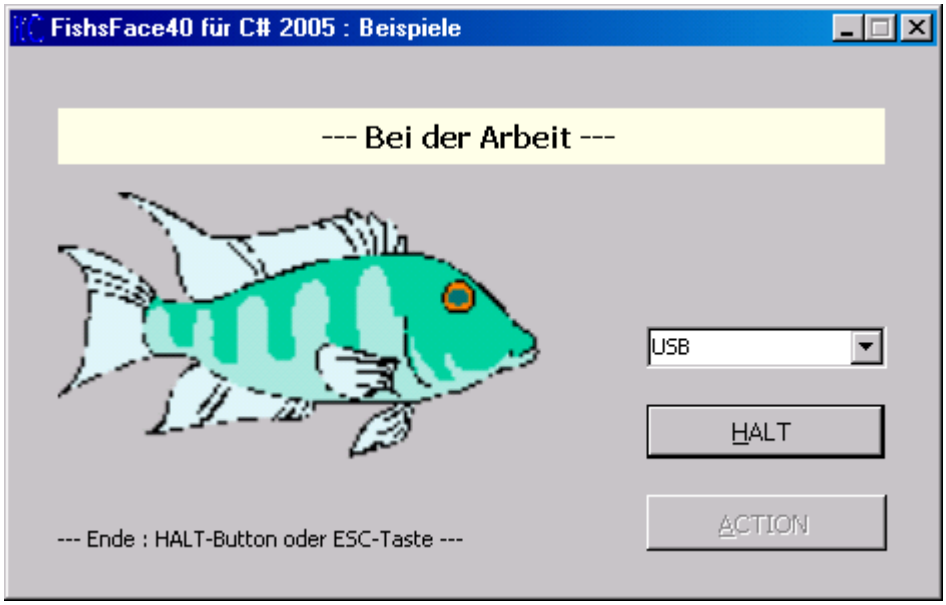
1. Menü Datei | Neues Projekt... | Konsolenanwendung
2. Projektnamen wählen : FishFaceConsole
3. Ggf. Program.cs umbenennen
   Projektmappen Explorer : FishFaceConsole
4. Projektmappen Explorer : Verweise
   Hinzufügen : FishFace2005.DLL (über Tab Durchsuchen oder Aktuell)
5. In FishFaceConsole.cs
   `using FishFace40;` 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 : FishFace2005.DLL (über Tab Durchsuchen oder Aktuell)
5. In der Source :
   `using FishFace40;` ergänzen.
7. Speichern
   Projektmappenverzeichnis anlegen, nicht erforderlich, wenn nur Einzelprojekt (man spart dann eine Verzeichnis-Ebene).

Vorlagen



Vorlage : FishFaceWindows

Bei der Standard Installation von cs2005fishface40setup.exe wird ein Verzeichnis ..\Templates\CS2005 angelegt, in dem sich eine Reihe von Projekten befinden, die sich gut als Programmrahmen für das Ausprobieren von Codeausschnitten dieses Handbuch, aber auch für das Ausprobieren eigener Ideen eignen mitgeliefert. Um sie als Vorlagen auf der eigenen C# 2005 Installation nutzen zu können, sind sie vorher noch als Vorlagen zu speichern. Sie erscheinen dann anschließen in dem Auswahlfenster für neue Projekte.

FishFaceConsole

```
class FishFaceConsole1 {
  static FishFace ft = new FishFace();

  static void Main(string[] args) {
    try {
      cn.WriteLine("--- Main gestartet ---");
      ft.OpenInterface(IFTypen.ftROBO_first_USB, 0);
      cn.WriteLine("--- Beenden : Escape-Taste ---");
      do {

          // --- Die Anwendungsbefehle ------

      } while (!ft.Finish());
      ft.ClearMotors();
    }
    catch (FishFaceException eft) {
      cn.WriteLine(eft.Message);
    }
    finally {
      ft.CloseInterface();
      cn.WriteLine("--- FINITO : RETURN ---");
      cn.Read();
    }
  }
}
```

Ist eine ganz einfache Konsolen-Anwendung. Mit einem try – catch – finally Block zum Abfangen von FishFace-Fehlern und einem cn.WriteLine
(`using cn = System.Console;`) für Programmausgaben.
Das OpenInterface fest auf IFTypen.ftROBO_first_USB eingestellt, also bei Bedarf ändern.

FishFaceWindows

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

Programm-Struktur

```
using FishFace40;

namespace BeispieleFishFace {
  public partial class FishFaceWindows1 : Form {
    FishFace ft = new FishFace();

    private void Action(){
      do {
      } while (!ft.Finish());
    }
          --- Programm-Kontrolle ---
  }
}
```

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
- cboPortName : ComboBox mit den Namen der möglichichen Interface-Anschlüsse
- 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.

```
private void cmdAction_Click(object sender, EventArgs e) {
  try {
    if (cboPortName.SelectedIndex == 0)
        ft.OpenInterface(IFTypen.ftROBO_first_USB, 0);
    else ft.OpenInterface(IFTypen.ftIntelligent_IF,
                          cboPortName.SelectedIndex, 5);
    cmdAction.Enabled = false;
    cmdEnde.Text = "&HALT";
    lblStatus.Text = "--- Bei der Arbeit ---";
    lblHinweis.Text = "--- Ende : HALT-Button oder ESC-Taste ---";

        Action();        // --- Haupt-Routine der Anwendung

    ft.ClearMotors();
  }
  catch (FishFaceException eft) {
    MessageBox.Show(eft.Message, this.Text,
          MessageBoxButtons.OK, MessageBoxIcon.Stop);
  }
  finally {
    ft.CloseInterface();
    cmdAction.Enabled = true;
    cmdEnde.Text = "&ENDE";
    lblStatus.Text = "--- Auf ein Neues ---";
    lblHinweis.Text = "Nochmal : ACTION, sonst ENDE oder ESC-Taste";
    cmdAction.Focus();
  }
}
```

cmdAction enthält einen try – catch – finally Block, der die Fehler aus FishFace40 abfängt. Das OpenInterface öffnet, nach Vorgabe der ComboBox, das erste ROBO Interface an USB bzw. ein Intelligent Interface an COMx. 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") ft.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.

FishRobotWindows

Wie FishFaceWindows, aber mit Instanzierung der Klasse FishRobot und einer Ereignisroutine zur Anzeige der Robot-Position

FishStepWindows

Wie FishFaceWindows, aber mit Instanzierung der Klasse FishStep und einer Ereignisroutine zur Anzeige der Plotter-Position.

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