fischertechnik Interfaces

umFish40.DLL

mit VC++, C#, VB.NET, Delphi, Visual Basic 6
& dem FishPanel40

Dokumentation zu Version 4.1

Ulrich Müller

Inhaltsverzeichnis

 Dokumentname : umFish41.DOC. Druckdatum : 20.12.2006
Bild Einfügen | Grafik | Aus Datei | Office | Fisch6.WMF

umFish40.DLL

Allgemeines

umFish40.DLL basiert auf der von fischertechnik zur Verfügung gestellten **FtLib** v59, die in die DLL eingebunden wurde. Sie unterstützt die unten aufgeführten Interfaces im **Online-Modus** (ständige Verbindung mit dem PC) :

- Erstes ROBO Interface an USB(ifTyp 0)
- Intelligent Interface (ifTyp 10)
- Intelligent Interface mit Slave (ifTyp 20)
- ROBO Interface im Intelligent Interface Mode (ifTyp 50)
- ROBO Interface an USB (ifTyp 60)
- ROBO Interface an COM (ifTyp 70)
- ROBO Interface über RF (ifTyp 80)
- ROBO I/O Extension an USB (ifTyp 90)
- ROBO RF Datalink an USB (ifTyp 110)

Das ROBO Interface kann als ifTyp 60, 70 und 80 mit bis zu 3 ROBO I/O Extensions ausgestattet werden.
Z. Zt. werden hier die M- und O-Ausgänge sowie die I-, IR und A-Eingänge unterstützt.

Der parallele Betrieb von mehren Interfaces (auch COM / USB gemischt) wird unterstützt. Im Programm wird das einzelne Interface durch ein Handle (iHandle) unterschieden.

Zusätzlich zu den aus FtLib genutzten Funktionen bietet umFish40.DLL einen ImpulsCounter für jeden I-Eingang, gezählt werden steigende und fallende Flanken. Auf dieser Basis auch eine Unterstützung für RobMotoren : Eine feste Kombination von Motor, Endtaster und Impulstaster : M1, I1, I2 .... M4, I7, I8.

Ziel der Entwicklung von umFish40.DLL war es, eine Basis für den Einsatz zusammen mit möglichst vielen Programmiersprachen zu schaffen. Deswegen wurde auch auf C++ spezifische Datentypen verzichtet. Es werden durchgängig int Variablen (32bit mit Vorzeichen) genutzt, die in vielen Programmiersprachen zu finden sind, trotzdem sind für einige Programmiersprachen noch besondere Anstrengungen erforderlich (z.B. Java : eine Wrapper.DLL nach JNI-Standard, Script-Sprachen : Angebot einer ActiveX.DLL).
Die Return-Werte vom Typ LPCSTR sind ein Fehltritt, sie können wohl nur aus VC++ genutzt werden.

Nicht im Download-Päckchen enthalten sind die Firmware für die ROBO Interfaces, der erforderlicheUSB-Treiber und die FtLib-Dateien.Diese können von fischertechnik (http://www.fischertechnik.de/computing/software.html) bezogen werden. Ebenso werden noch keine Dienstprogramme zum Aufspielen der Firmware angeboten.

In beiden Fällen ist der Einsatz von ROBO Pro zu empfehlen. Da die FtLib in umFish40.DLL integriert ist, wird diese nicht separat benötigt.

Für den Test von frisch aufgebauten Modellen empfiehlt sich das beiliegende Tool umFishDP40.EXE (FishPanel).

HINWEIS : Bei gewaltsamem Abbruch einer Anwendung im Test kann es vorkommen, dass der requirierte Speicher nicht wieder vollständig freigegeben wird. Mit dem Speicher kann es also im Eifer des Testens immer weniger werden : BOOTEN.

Erweiterungen gegenüber Version 4.0

Unterstützung für den Nachrichtenaustausch zwischen einer PC-Anwendung und einer Anwendung im Interface selber (z.B. mit Renesas C erstellt).

Für diese Einsatzart ist ein aktueller Firmwaresatz für ROBO Interface (v1.64.0.03, Funkplatine v0.44.0.03) und ROBO RF Datalink (v0.44.0.03) erforderlich, sonst gehts auch mit älterer Firmware.

Ein Umstieg auf umFish40.DLL v4.1.59.1 von älteren Versionen kann kompatibel erfolgen. Er ist nur erforderlich, wenn die neuen Funk-Funktionen genutzt werden sollen.

Siehe auch Kapitel : Anmerkungen zu den Funk-Funktionen.

.

Funktionen

Konventionen

Parameterbezeichnungen und deren Wertebereich

Beim Wertebereich wird in Klammern der Maximalwert bei vollem Ausbau eines USB Interfaces mit 3 Extensions angegeben.

iHandle	Handle zur Identifizierung des aktuellen Interfaces im Programm 1 - 8
MotNr	Nr eines M-Ausganges 1 – 4 (16)
LampNr	Nr eines O-Ausganges 1 – 8 (32). "Halber" M-Ausgang Beim Intelligent Interface nicht nutzbar.
InputNr	Nr eines I-Einganges 1 – 8 (32). Intelligent Interface : E-Eingang 1 – 8 (16)
Inputwert	Wert eines I-Einganges 0 / 1
InputStatus	Status aller (max. 32) I-Eingänge, I1 rechts, je 1bit.
CounterNr	Nummer des Zählers für einen I-Eingang 1 – 8 (32)
AnalogNr	AX / AY / AXS1 / AXS2 / AXS3 : 1 – 2 (5) Intelligent Interface EX / EY : 1 – 2
Analogwert	Wert eines A-Einganges 0 - 1023
VoltNr	A1 / A2 / AV / AZ : 1 – 4 nicht beim Intelligent Interface
Dir	Drehrichtung bei den M-Ausgängen : Aus = 0, Links = 1, Rechts = 2
MotorStatus	Dir-Werte aller Motoren, M1 rechte bits, 2bit
Mode	Betriebsmode eines Motors. Normal = 0, RobMode = 1 vorgesehen : Step1, Step2
ModeStatus	**(**Betriebs)Modealler Motoren, M1 rechte bits, 2bit
Speed	PWM Geschwindigkeitsstufe (M-Ausgänge) : 0 – 7
SpeedStatus	Geschwindigkeitsstufe von 8 aufeinanderfolgenden Motoren M1 – M8 und M9 – M16, M1 bzw. M9 rechte bits, 4bit
Power	PWM Intensität (O-Ausgänge) : 0 – 7
OnOff	Ein / Aus : 1 / 0
ICount	Wert eines Impulszählers
ifTyp	Interfacetyp (siehe Allgemeines)
SerialNr	Standardseriennummer eines USB-Interfaces bei SerialNr = 0 wird das erste USB-Device der Liste genommen.
ComNr	Nummer des COM-Ports an dem das Interface angeschlossen ist. 1 – 4
Fehlercode	Fehlercode rbFehler oder 0
int	Allgemeiner 32bit Integer-Wert mit Vorzeichen.

Alle Parameter sind vom Typ int (32 bit mit Vorzeichen)

Nachrichten

Werden in der Struktur MessageData transportiert :

```
typedef struct {
  BYTE HwId;        Angabe über die Sendeart (hier immer 2 : Senden an alle anderen
                    Teilnehmer),
  BYTE SubId;       Klassifizierung der Nachricht
  USHORT MsgId;     Nummer der Nachricht
  USHORT Msg;       Die Nachricht selber.
} MessageData;
```

Mit Ausnahme von HwId können die Bestandteile der Nachricht frei genutzt werden, eine andere Gliederung ist im Rahmen der Wordeinteilung also möglich. Wenn auf der Interface-Seite ein mit ROBO Pro erstelltes Programm eingesetzt wird, muß man sich allerdings an dessen Eigenheiten anpassen.

Fehlerbehandlung

Alle Funktionen haben einen Rückgabewert, der im Fehlerfall rbFehler (0xE0000001) liefert, im Erfolgsfall den gewünschten Rückgabewert bzw. 0, wenn es keinen gibt. Die zahlreichen Fehlercodes, die FtLib liefert, wurden nicht genutzt, da sie im praktischen Betrieb selten weiterhelfen. Meist reicht die kontextabhängige Interpretation der Situation. Beim Test unter VC++ IDE kann man sie natürlich begutachten.

Befehlsliste

iHandle **rbOpenInterfaceUSB**(ifTyp, SerialNr)
Herstellen einer Verbindung zu einem Interface an USB
ggf. über ein ROBO RF Datalink, in diesem Fall wird das erste passende Interface mit gleicher RF-Kanalnummer und beliebiger Unterkanalnummer genutzt.
ifTyp = 0 : erstes Interface an USB, dann auch SerialNr = 0.
Bei RF Datalink dann mit der ersten passenden Interface Typ 80
weitere ifTypen : 60 und 90
ROBO Pro – Entsprechung : 7.1.1 Start (so beinahe)

iHandle **rbOpenInterfaceRF**(SerialNrInterface)
Herstellen einer Funkverbindung zu einem wählbarenROBO Interface über ein ROBO RF Datalink (das erste irgendwo an USB). Hier muß zur Identifikation des Interface dessen Serialnummer angegeben werden. Das Interface steht "unter Strom", ist aber nicht an USB angeschlossen.
ifTypen 110 mit 80

iHandle **rbOpenInterfaceCOM**(ifTyp, ComNr, AnalogZyklen)
Herstellen einer Verbindung zu einem Interface an COM
ifTypen 10, 20, 50 und 70
ROBO Pro – Entsprechung : 7.1.1 Start (so beinahe)

Fehlercode **rbCloseInterface**(iHandle)
Beenden der Verbindung zum Interface
ROBO Pro – Entsprechung : 7.1.e Ende (so beinahe)

A- und I-Eingänge

OnOff **rbGetInput**(iHandle, InputNr)
Auslesen des Zustandes des angegebenen I-Einganges
ROBO Pro – Entsprechung : 7.1.3 Verzweigung & If

int **rbGetInputs**(iHandle)
InputStatus : Status des Zustandes aller I-Eingänge (I1 rechts, 1bit)

int **rbGetAnalog**(iHandle, AnalogNr)
Auslesen des AnalogWertes des angegebenen A-Einganges
(AX, AY bzw. EX, EY und ggf. AXS1, AXS2, AXS3)
ROBO Pro – Entsprechung : 7.1.4 Verzweigung Analog & If

int **rbGetIRKey**(iHandle, Code, KeyNr)
Auslesen des Zustandes des angegebenen IR-Keys des IR-Senders

int **rbGetVoltage**(iHandle, VoltNr)
Auslesen des VoltWertes des angegebenen A-Einganges (A1 – A2)
ROBO Pro – Entsprechung : 7.1.4 Verzweigung Analog & If

M- und O-Ausgänge :

Fehlercode **rbSetMotor**(iHandle, MotNr, Dir)
Setzen eines M-Ausganges mit Speed = 7
ROBO Pro – Entsprechung : 7.1.6 Motorausgang

Fehlercode **rbSetMotorEx**(iHandle, MotNr, Dir, Speed)
Setzen eines M-Ausganges einschl. Angabe der Drehzahlstufe (Speed), wird beim Intelligenten Interface nicht ausgewertet.
ROBO Pro – Entsprechung : 7.1.6 Motorausgang

int **rbGetMotors**(iHandle)
Einschaltstatus aller M-Ausgänge (M1 rechts, 2bit)

Fehlercode	**rbSetMotors**(iHandle, MotorStatus) Schalten aller M-Ausgänge (M1 rechts, 2bit), NormalMode, Speed = 7
Fehlercode	**rbSetMotorsEx**(iHandle, MotorStatus, SpeedStatus, SpeedStatus16) Setzen aller M-Ausgänge einschl. Speed im NormalMode
int	**rbGetModeStatus**(iHandle, MotNr) Abfrage des ModeStatus eines M-Ausganges (Normal = 0, RobMode = 1)
Fehlercode	**rbSetModeStatus**(iHandle, MotNr, Mode) Setzen des ModeStatus eines M-Ausganges (Normal = 0, RobMode = 1)
Fehlercode	**rbSetLamp**(iHandle, LampNr, OnOff) Setzen eines O-Ausganges mit Power = 7 *ROBO Pro – Entsprechung* : 7.1.7 Lampenausgang
Fehlercode	**rbSetLampEx**(iHandle, LampNr, OnOff, Power) Setzen eines O-Ausganges einschl. Angabe der Intensität Nur ROBO Interface / Extension *ROBO Pro – Entsprechung* : 7.1.7 Lampenausgang
Fehlercode	**rbRobMotor**(iHandle, MotNr, Dir, Speed, ICount) Starten eines M-Ausganges im RobMode (mit angeschlossenem Motor und Impulsrädchen an den zugehörenden I-Eingängen. Der Befehl läuft asynchron und beendet sich bei ICount = 0 bzw. Endtaster (linksdrehend) selber. mit rbGetCounter kann das abgefragt werden. *ROBO Pro – Entsprechung* : 7.1.6 Motorausgang und 7.1.9 Impulszähler (so in etwa)
Fehlercode	**rbRobMotors**(iHandle, MotorStatus, SpeedStatus, SpeedStatus16, ModeStatus) Setzen des vollständigen Status aller M-Ausgänge Die evtl. zugehörenden Counter müssen separat gesetzt werden.

ImpulsCounter :

ICount	**rbGetCounter**(iHandle, CounterNr) Lesen eines ImpulsCounter-Standes *ROBO Pro – Entsprechung* : 7.1.9 Impulszähler (so in etwa)
Fehlercode	**rbSetCounter**(iHandle, CounterNr, ICount) Setzen eines ImpulsCounter-Standes *ROBO Pro – Entsprechung* : 7.1.9 Impulszähler (so in etwa)
Fehlercode	**rbClearCounter**(iHandle) Löschen aller ImpulsCounter auf 0. *ROBO Pro – Entsprechung* : 7.1.9 Impulszähler (so in etwa)

Funk-Funktionen :

Fehlercode	**rbClearMessagesIn**(int iHandle) Löschen der Queque mit den einkommenden Nachrichten
Fehlercode	**rbClearMessagesOut**(int iHandle) Löschen der Queque mit den ausgehenden Nachrichten
Fehlercode	**rbGetMessage**(int iHandle, MessageData* inNachricht) Abrufen einer Broadcast Nachricht aus der inQueque
int	**rbIsMessage**(int lHandle) Abfrage, ob Nachrichten in der Queque der einkommenden Nachrichten vorliegen. 0 = keine, > 0 = Anzahl der Nachrichten oder Fehlercode
Fehlercode	**rbSendMessage**(int iHandle, MessageData* outNachricht) Senden (Stellen in die outQueque) einer BroadcastNachricht.
Fehlercode	**rbSendMessageEx**(int iHandle, MessageData* outNachricht, int Spez) Bedingtes Senden (Stellen in die outQueque) einer Nachricht. Spez : 0 = immer, 1 wenn gegenüber letzten in der outQueque neu, 2 = wenn nicht in der outQueque enthalten

Auskunftsfunktionen : *ROBO Pro – Entsprechung : Interface Test*

int	**rbGetActDeviceType**(iHandle) Auslesen des aktiven DeviceTyps. Erfolgreiches rbOpenInterface erforderlich, sonst Fehlercode
int	**rbGetActDeviceSerialNr**(iHandle) Auslesen der SerialNr des aktiven Device. Erfolgreiches rbOpenInterface erforderlich, sonstFehlercode
int	**rbGetActDeviceFirmwareNr**(iHandle) Auslesen der FirmwareNr des aktiven Device. Erfolgreiches rbOpenInterface erforderlich, sonst Fehlercode
LPCSTR	**rbGetActDeviceFirmware**(iHandle) Auslesen des Firmwarestrings des aktiven Device Erfolgreiches rbOpenInterface erforderlich, sonst NULL
LPCSTR	**rbGetActDeviceName**(iHandle) Auslesen des Namens des aktiven Device Erfolgreiches rbOpenInterface erforderlich, sonst NULL

Anmerkungen zur umFish40.DLL Source

umFish40.DLL ist eine in VC++ 6.0 geschriebene systemkonforme.DLL, die für den eigentlichen Interfacezugriff die Funktionen der von fischertechnik bereitgestellten FtLib_Static_LIBCMT_Release.lib nutzt. Sie bietet eine Reihe von Basisfunktionen für den Zugriff auf die Interfaces der ROBO Serie und des Intelligent Interface. Zusätzlich zur einfachen Abfrage der Eingänge und dem Setzen der Ausgänge werden eine Counter-Funktion (Zählen der Häufigkeit der Betätigung der I-Eingänge) und RobMotoren (Betreiben eines M-Ausganges für eine bestimmte Anzahl von Impulsen an einem I-Eingang) geboten.

Ziel der Entwicklung war eine DLL, die aus möglichst vielen Programmiersprachen genutzt werden kann. Es werden deswegen an den Schnittstellen durchgängig nur einfache Konstrukte verwendet. Extern verwendete Variable sind durchgängig von Typ int (32bit mit Vorzeichen), Strukturen ... Überladungen nicht verwendet.

Die Source besteht in ihrem Kern aus folgenden Dateien :

- umFish40.DEF : Deklarationen der Entries der DLL
- umFish40.H : Deklaration der externen Funktionen, interna
- umFish40.CPP : Die eigentliche Source
- umFtLib.H : Die vereinfachte FtLib.H von fischertechnik für den Zugriff auf die FtLib.

FtLib-Funktionen

Die FtLib kontrolliert den direkten Zugang zu den Interfaces. Dazu bietet sie eine Reihe von Funktionen, die den Verbindungsauf- und -abbau mit dem jeweiligen Interface regeln (OpenFtUsbDevice / OpenFtCommDevice ... StartFtTransferArea ...) und einige Auskunftsfunktionen (GetFtDeviceType / GetFirmwareStrg ...) sowie Funktionen zum Download von Assembler-Programme in ein ROBO-Interface (z.Zt. hier nicht genutzt).

Die eigentliche Kommunikation mit dem Interface erfolgt über einen Kommunikationsbereich – die TransferArea – in der im 10 ms Takt die Werte der Ein- und Ausgänger des Interfaces aktualisiert werden. In diesem Zusammenhang wird über die Struktur NOTIFICATION_EVENTS ein CallBack-Ausgang geboten, der von umFish40.DLL für dessen Zusatzfunktionen genutzt wird.

umFish40.DLL-Funktionen

Die wesentliche Aufgabe von umFish40.DLL ist die Aufbereitung der in der TransferArea vorliegenden Daten und deren Überführung in Funktionen. Aus 'unsigned char E_Main' wird so eine Funktion rbGetInput(InputNr), die Auskunft über den Zustand eines einzelnen I-Eingang gibt. Die Ein- und Ausgänge werden im Gegensatz zu manchen Teilen der FtLib auch durchgängig fortlaufend gezählt (I-Eingänge von 1 – max. 32, M-Ausgänge von 1 – max. 16).

Der recht aufwändige Verbindungsaufbau mit einem Interface wird auf eine Funktion (rbOpenInterfaceUSB bzw. rbOpenInterfaceCOM) reduziert. Ebenso der Verbindungsabbau.

Die Zusatzfunktionen von umFish40.DLL werden in einer CallBack-Routine der NOTIFICATION_EVENTS von FtLib gewonnen und in entsprechenden Funktionen weitergereicht.

Die Auskunftsfunktionen werden nur vereinzelt angeboten, soweit allgemeiner Bedarf angenommen werden kann.

Die Funktionen für Code-Download und weiteres werden z.Zt. nicht angeboten.

umFish40.DLL-Details

umFish40.DLL kann max. bis zu acht Interface-Verbindungen simultan betreiben. Die erforderlichen Instanzdaten werden in dem Array RoboInstanz gehalten. Das jeweilige OpenInterface reicht anstelle eines echten Handle (um Zeiger zu vermeiden) einen Index nach außen, der die Instanz identifiziert.

Ein typischer Zugriff auf die TransferArea sieht dann so aus :

```
rbI[iHandle].ftDCB->M_Main |= MLinks[n];
```

Einschalten eines M-Ausganges in der TransferArea (ftDCB)

Das Konstrukt :

```
if(rbI[iHandle].ftDCB == NULL) return rbFehler;
```

wird zur Abfrage auf ein gültiges OpenInterface genutzt.

```
if(!(IsFtTransferActiv(rbI[iHandle].ftHandle) ==
     FTLIB_ERR_THREAD_IS_RUNNING)) return rbFehler;
```

fragt nach einer bestehende Verbindung zum Interface.

Die Ein- und Ausgänge werden nach außen mit 1 beginnend gezählt (intern ab 0), deswegen findet sich oft z.B. ein MotNr-- zu Beginn einer Routine.

Die Maskierung der Ein- und Ausgabefelder ist "historisch" gewachsen, sie erfolgt teils über Tabellen, teils über Shiften.

Nutzung

Allgemeines

umFish40.DLL liegt in kompilierter Form und als VC++ Projekt vor.

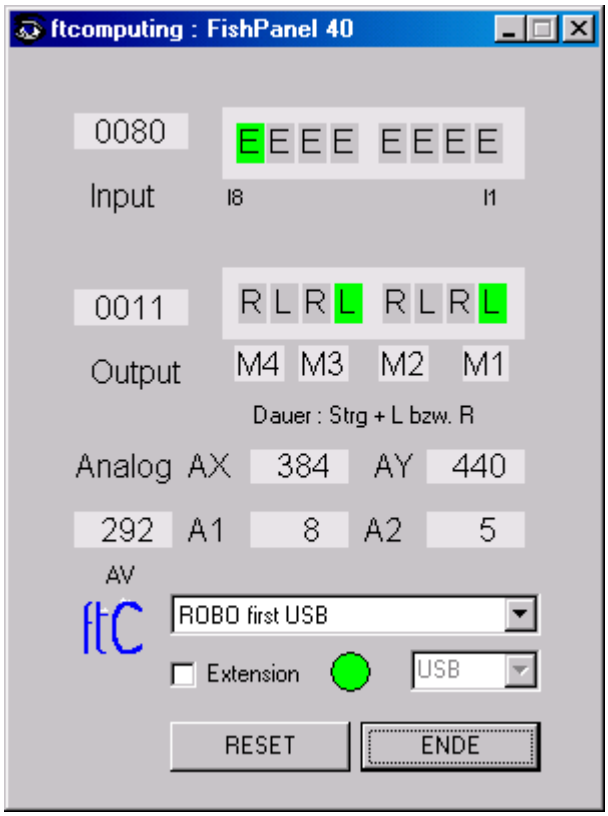
Zusätzlich gibt es für eine Reihe von Sprachen die erforderlichen "H"-Files (Deklarationen, teils in Form von Klassen). Und – diese nutzend – ein kleines Hello ROBO (5mal Blinken an M1 nach Start über I1 = true) am ersten ROBO Interface an USB.

umFish40.DLL ist in erster Linie als Basis für die Entwicklung eigener Bibliotheken gedacht, eine direkte Nutzung kann problematisch sein, da z.B. Unterbrechbarkeit(DoEvents) und Abbrechbarkeit (ESC-Taste) nicht gegeben sind. Einige Programmiersprachen (Scriptsprachen, Logo und Java ... kommen mit einer systemkonformen DLL.Schnittstelle auch nicht zurecht).

In die Hello-Projekte wurde das Senden einer einzelnen (willkürlichen) Nachricht eingebaut um zu zeigen wie mans überhaupt macht. Das Hello läuft aber auch unverändert mit einer simplen ROBO I/O Extension so ganz ohne Funk. Es wird halt gefunkt ohne eine Kontrolle ob auch etwas ankommt. Siehe auch "Anmerkungen zu den Funk-Funktionen"

Für die direkte Anwendungsprogrammierung eignet sich je nach Sprache eine sprachspezifische Klassenbibliothek (FishFace40...) oder eine ActiveX.DLL (FishFa40AX.DLL) deutlich besser. Wenn man da eine Sprache vermißt, lohnt sich eine Kontaktaufnahme mit um@ftcomputing.de

InterfacePanel



Liegt in kompilierter Form vor (**umFishDP40.EXE)** und kann zur Kontrolle frisch aufgebauter Modelle und zum 'Geraderücken' von in die Irre gelaufenen Modellen genutzt werden.

VC++

Projekt mit VC++ 6.0 anlegen

1. Neu | Projekt | Win32-Konsolen-Anwendung : leeres Projekt
2. Dem Projekt hinzufügen : neue C++ Quelldatei oder vorhandene einfügen.
3. Und die Dateien umFish40.H und umFish40.lib einfügen
4. Die umFish40.DLL (Release Version) muß in einem erreichbaren Pfad liegen : Verzeichnis Debug des Projektes oder WinNT\System32
5. Projekteinstellungen : Win32Debug | C/C++ Kategorie CodeGeneration | LaufzeitBibliothek : Multithreaded.DLL

```
#include <Windows.h>
#include <iostream.h>
#include "../umFish40VC.H"

void main() {
  char Ende;
  cout << "--- HalloROBO : es geht los ---" << endl;
  int iHandle = rbOpenInterfaceUSB(ftROBO_first_USB, 0);
  if (iHandle == rbFehler) {
    cout << "Da stimmt etwas nicht : ENDE (Enter-Taste)" << endl;
    cin.get(Ende);
    return;
  }
  cout << "Interface : " << rbGetActDeviceName(iHandle)
       << ", Typ : " << rbGetActDeviceType(iHandle)
       << ", SerialNr : " << rbGetActDeviceSerialNr(iHandle) <<
endl;
  cout << "Firmware  : " << rbGetActDeviceFirmware(iHandle) << endl;
  cout << endl << "Start : I1 druecken" << endl;
  while(!rbGetInput(iHandle, 1)) {Sleep(123);}
  for (int i = 0; i < 5; i++) {
    rbSetMotor(iHandle, 1, 1);
    Sleep(333);
    rbSetMotor(iHandle, 1, 0);
    Sleep(333);
  }
  rbCloseInterface(iHandle);
  cout << endl << "--- FINIS : Enter-Taste ---" << endl;
  cin.get(Ende);
}
```

C#

```
using System;
using System.Threading;
using cs = System.Console;
using um = HelloCSROBO.umFish40CS;

namespace HelloCSROBO {
   class Rahmen {
      uint iHandle;
      [STAThread]
      static void Main(string[] args)  {
         Rahmen rt = new Rahmen();
         cs.WriteLine("--- Hello ROBO gestartet ---");
         rt.Action();
         cs.WriteLine("--- Hello ROBO beendet (Return-Taste) ---");
         cs.Read();
      }
      private void Action() {
         iHandle =
              um.rbOpenInterfaceUSB((int)IFTypen.ftROBO_first_USB, 0);
         if(iHandle == um.rbFehler) {
            cs.WriteLine("Da stimmt was nicht : ENDE (Return-Taste)");
            return;
         }
         cs.WriteLine("IN ACTION : Start I1 drücken");
         while(um.rbGetInput(iHandle, 1) == 0) {um.Sleep(123);};
         for(int i = 0; i < 5; i++) {
            cs.WriteLine("Blinker : " + i);
            um.rbSetMotor(iHandle, 1, 1);
            um.Sleep(333);
            um.rbSetMotor(iHandle, 1, 0);
            um.Sleep(333);
         }
         um.rbCloseInterface(iHandle);
      }
   }
}
```

Console Projekt.Die Elemente der Klasse umFish40CS sind static, deshalb keine Instanziierung.

VB.NET

```
Imports cs = System.Console
Imports um = HelloVBNETRobo.umFish40VBNET

Module HelloMain

  Sub Main()
    Dim i%, iHandle%
    cs.WriteLine("--- Hello VB.NET gestartet ---")
      iHandle = um.rbOpenInterfaceUSB(IFTypen.ftROBO_first_USB, 0)
      If iHandle = um.rbFehler Then
        cs.WriteLine("Da stimmt was nicht : ENDE (Return-Taste)")
        Return
      End If
      cs.WriteLine("Interface : " & _
                um.rbGetActDeviceType(iHandle) & " / " _
                & um.rbGetActDeviceSerialNr(iHandle))
      cs.WriteLine("mit Firmware : " & _
                 um.rbGetActDeviceFirmwareNr(iHandle).ToString("X"))
      cs.WriteLine("IN ACTION : Start I1 drücken")
      While um.rbGetInput(iHandle, 1) = 0
        um.Sleep(123)
      End While
      For i = 1 To 5
        cs.WriteLine("Blinker : " & i)
        um.rbSetMotor(iHandle, 1, 1)
        um.Sleep(333)
        um.rbSetMotor(iHandle, 1, 0)
        um.Sleep(333)
      Next
      um.rbCloseInterface(iHandle)
      cs.WriteLine("--- Hello VB.NET beendet (Return-Taste) ---")
      cs.Read()
  End Sub

End Module
```

Console Projekt.Die Elemente der Klasse umFish40VBNET sind static, deshalb keine Instanziierung.

Delphi

Verwendet wurde hier Delphi4, es müßte aber von Delphi 2 – Delphi 7 reichen.

```
program HalloDelphiROBO;

uses
  Windows, SysUtils,
  umFish40 in 'umFish40.PAS';
var
  ft, i: LongInt;
begin
  ft := rbOpenInterfaceUSB(ftiROBO_first_USB, 0);
  if ft = ftiFehler then begin
    WriteLn('Hier stimmt etwas nicht : ENDE (Enter-Taste)');
    ReadLn;
    exit;
  end
  else WriteLn('HalloDelphiROBO in Action');
  WriteLn('Interface : ' + IntToStr(rbGetActDeviceType(ft)) +
    ' / ' + IntToStr(rbGetActDeviceSerialNr(ft)));
  WriteLn('mit Firmware : ' +
IntToStr(rbGetActDeviceFirmwareNr(ft)));
  WriteLn('Start : I1 druecken');
  while rbGetInput(ft, 1) = 0 do Sleep(123);
  for i := 1 to 5 do begin
    WriteLn('Runde : ' + IntToStr(i));
    rbSetMotor(ft, 1, ftiEin);
    Sleep(333);
    rbSetMotor(ft, 1, ftiAus);
    Sleep(333);
 end;
  rbCloseInterface(ft);
  WriteLn('HalloDelphiROBO beendet'); ReadLn;
end.
```

Visual Basic 6

```
Option Explicit

Dim iHandle&

Private Sub Form_Load()
  iHandle = rbOpenInterfaceUSB(ftROBO_first_USB, 0)
  If iHandle = rbFehler Then
    MsgBox "Hello ROBO : Da stimmt etwas nicht"
    End
  End If
End Sub

Private Sub cmdAction_Click()
Dim i&
  lstAus.AddItem "Hello Robo gestartet"
  lstAus.AddItem "Interface : " & rbGetActDeviceType(iHandle) & _
      " / " & rbGetActDeviceSerialNr(iHandle)
  lstAus.AddItem "Firmware : " & rbGetActDeviceFirmwareNr(iHandle)
  lstAus.AddItem "Start : I1 drücken"
  Do: DoEvents: Sleep 123: Loop Until rbGetInput(iHandle, 1) = 1
  For i = 1 To 5
  lstAus.AddItem "Runde : " & i
    rbSetMotor iHandle, 1, 1
    DoEvents
    Sleep 333
    rbSetMotor iHandle, 1, 0
    DoEvents
    Sleep 333
  Next i
  lstAus.AddItem "FINITO : Das wars (x-Klicken)"
End Sub

Private Sub Form_Unload(Cancel As Integer)
  rbCloseInterface (iHandle)
End Sub
```

Anmerkungen zu den Counters

Ein wesentliches Element zur Positionsbestimmung sind die Counters. Sie sind den E-Eingängen zugeordnet In den Countern wird jede Veränderung des Zustandes der I-Eingänge gezählt. Also z.B. das Öffnen oder auch das Schließen eines Tasters.

Es werden entsprechende Funktionen/Methoden zur Handhabung der Counter angeboten. Die Counter werden auch intern von einigen Funktionen/Methoden (z.B. SetMotor mit Parameter Counter und den meisten Wait-Methoden) genutzt

Anmerkungen zu den Rob-Funktionen

Die Rob-Funktionen laufen in einem bsonderen Betriebsmodus, dem RobMode. In diesem Modus werden die betroffenen Counter decrementiert. Bei Erreichen des Wertes 0 wird der betroffen Motor abgeschaltet. Während der letzten 6 Impulse fahren sie nur noch mit halber Geschwindigkeit um ein sicheres Erreichen der Endposition zu erreichen. Gelegentlich kann es trotzdem 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 bei genügend vielen Extensions weiter bis Motor 16

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

Die Motoren können einzeln über rbRobMotor bzw. eine Variante von SetMotors geschaltet werden. Das Argument ICount/Counter gibt die Fahrstrecke in Impulsen (true/false- oder false/true-Durchgang am zugehörenden Impulstaster) an. Sie werden während des Pollens auf Null heruntergezählt. Sie sind über den ftiDCB.Counter bzw. die Funktion csGetCounter zugänglich. Also Achtung, etwa durch die Anwendung gesetzte Counter können in diesem Fall geändert werden.

Die Motoren können auch alle mit einem Befehl geschaltet werden : rbRobMotors bzw. einer Variante von SetMotor. 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, 0111 voll.
ModeStatus : proMotor 2 bit, mit M1 : bit 0-1 beginnend,
00 Normal-Mode, 01 Rob-Mode, Rest z.Zt. nicht besetzt
(vorgesehen z.B. für Schrittmotorenbetrieb).

Beispiel : rbRobMotors(ft, 0x9, 0x74, 0x0, 0x05);
0x steht für Hexa, binär : MotorStatus 1001 SpeedStatus 01110100 ModeStatus 0101 -> M2 = rechts, Speed 7 im Rob-Mode, M1 = links, Speed 4 im RobMode. Der Rest steht. Vor rbRobMotors sind für jeden Motor einzeln die Impuls-Counter 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 acht oder auch 16), sie können der Reihe nach mit rbRobMotor geschaltet werden. Sie starten dann beim nächsten Zyklus 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) während des Zyklus 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. Bei den FishFace-Klassen ist eine entsprechende Methode zu finden, umFish40.DLL bietet keine entsprechende Funktion. Bei Einsatz von umFish40-Funktionen ist sie selber zu programmieren (siehe auch Beispiele, Grund für das Fehlen : umFish40.DLL soll keine sprachabhängigen Elemente enthalten).

Anmerkungen zum Funk-Betrieb

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 durch umFish40.DLL voll 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 umFish40.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 umFish40.DLL von der PC-Seite aus unterstützt.
   Die Programmierung der Interfaces erfolgt in ROBO Pro oder Renesas C (ab dem zweiten Interface).

Dazu gibt es einige speziellen Funktionen :
- rbSendMessage zum Senden einer gepufferten (Broadcast) Nachricht.
– rbGetMessage und rbIsMessage zum entgegennehmen einkommender Nachrichten aus dem Empfangspuffer. Sowie rbClearMessagesIn, rbClearMessagesOut zum Löschen der Messages-Queques.
– Die 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.

In die Hello-Demos ist ein rbSendMessage für einen ganz einfachen Versuch für Funk-Betrieb Typ 3 eingebaut. Dazu braucht man allerdings trotzdem einiges an Hardware :

- ROBO RF Datalink Einstellung RF2/0 an USB
  Das Datalink muß einfach nur da sein.
- ROBO Intelligent Interface mit FunkPlatine (Einstellung RF2/1) nur an Strom mit Taster an I1 und Lampe an M1. Das Interface wird durch das PC-Programm gesteuert : Es wird gesendet und geblinkt.
- ROBO Intelligent Interface mit FunkPlatine (Einstellung RF2/2) nur an Strom mit Motor an M1. Ausgerüstet mit einem Programm (Renesas C / ROBO Pro), das in einer Endlosschleife auf Nachrichten wartet und dann unabhängig vom Inhalt der Nachricht den Motor an M1 für 1 sec links laufen läßt

Zu Beginn der Demo sollte dann der Motor an RF2/2 für ca. 1 sec laufen, danach wird dann geblinkt.