

---

fischertechnik Interfaces

# umFish40.DLL

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

Dokumentation zu Version 4.0

Ulrich Müller



# Inhaltsverzeichnis

<b>umFish40.DLL</b>	<b>3</b>
Allgemeines	3
Funktionen	4
Konventionen	4
Fehlerbehandlung	5
Befehlsliste	6
Anmerkungen zur umFish40.DLL Source	8
FtLib-Funktionen	8
umFish40.DLL-Funktionen	8
umFish40.DLL-Details	9
<b>Nutzung</b>	<b>10</b>
Allgemeines	10
InterfacePanel	11
VC++	12
C#	13
VB.NET	14
Delphi	15
Visual Basic 6	16
Java	17
Anmerkungen zu den Counters	18
Anmerkungen zu den Rob-Funktionen	18

Copyright © 2005 Ulrich Müller. Dokumentname : umFish40.DOC. Druckdatum : 15.06.2005  
Bild Einfügen | Grafik | Aus Datei | Office | Fisch6.WMF

# umFish40.DLL

---

## Allgemeines

umFish40.DLL basiert auf der von fischertechnik zur Verfügung gestellten **FtLib** v38, die in die DLL eingebunden wurde. Sie unterstützt die unten aufgeführten Interfaces im **Online-Modus** (ständige Verbindung mit dem PC) :

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

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 mehreren 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 erforderliche USB-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 (InterfacePanel).

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.

---

# Funktionen

## Konventionen

### Parameterbezeichnungen und deren Wertebereich

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

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

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

## 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	<b>rbOpenInterfaceUSB</b> (ifTyp, SerialNr) Herstellen einer Verbindung zu einem Interface an USB <i>ROBO Pro – Entsprechung : 7.1.1 Start (so beinahe)</i>
iHandle	<b>rbOpenInterfaceCOM</b> (ifTyp, ComNr, AnalogZyklen) Herstellen einer Verbindung zu einem Interface an COM <i>ROBO Pro – Entsprechung : 7.1.1 Start (so beinahe)</i>
Fehlercode	<b>rbCloseInterface</b> (iHandle) Beenden der Verbindung zum Interface <i>ROBO Pro – Entsprechung : 7.1.e Ende (so beinahe)</i>
OnOff	<b>rbGetInput</b> (iHandle, InputNr) Auslesen des Zustandes des angegebenen I-Einganges <i>ROBO Pro – Entsprechung : 7.1.3 Verzweigung &amp; If</i>
int	<b>rbGetInputs</b> (iHandle) InputStatus : Status des Zustandes aller I-Eingänge (I1 rechts, 1bit)
int	<b>rbGetAnalog</b> (iHandle, AnalogNr) Auslesen des AnalogWertes des angegebenen A-Einganges (AX, AY bzw. EX, EY und ggf. AXS1, AXS2, AXS3) <i>ROBO Pro – Entsprechung : 7.1.4 Verzweigung Analog &amp; If</i>
int	<b>rbGetIRKey</b> (iHandle, Code, KeyNr) Auslesen des Zustandes des angegebenen IR-Keys des IR-Senders
int	<b>rbGetVoltage</b> (iHandle, VoltNr) Auslesen des VoltWertes des angegebenen A-Einganges (A1 – A2) <i>ROBO Pro – Entsprechung : 7.1.4 Verzweigung Analog &amp; If</i>
Fehlercode	<b>rbSetMotor</b> (iHandle, MotNr, Dir) Setzen eines M-Ausganges mit Speed = 7 <i>ROBO Pro – Entsprechung : 7.1.6 Motorausgang</i>
Fehlercode	<b>rbSetMotorEx</b> (iHandle, MotNr, Dir, Speed) Setzen eines M-Ausganges einschl. Angabe der Drehzahlstufe (Speed), wird beim Intelligenten Interface nicht ausgewertet. <i>ROBO Pro – Entsprechung : 7.1.6 Motorausgang</i>
int	<b>rbGetMotors</b> (iHandle) Einschaltstatus aller M-Ausgänge (M1 rechts, 2bit)
Fehlercode	<b>rbSetMotors</b> (iHandle, MotorStatus) Schalten aller M-Ausgänge (M1 rechts, 2bit), NormalMode, Speed = 7
Fehlercode	<b>rbSetMotorsEx</b> (iHandle, MotorStatus, SpeedStatus, SpeedStatus16) Setzen aller M-Ausgänge einschl. Speed im NormalMode
int	<b>rbGetModeStatus</b> (iHandle, MotNr) Abfrage des ModeStatus eines M-Ausganges (Normal = 0, RobMode = 1)
Fehlercode	<b>rbSetModeStatus</b> (iHandle, MotNr, Mode) Setzen des ModeStatus eines M-Ausganges (Normal = 0, RobMode = 1)
Fehlercode	<b>rbSetLamp</b> (iHandle, LampNr, OnOff) Setzen eines O-Ausganges mit Power = 7 <i>ROBO Pro – Entsprechung : 7.1.7 Lampenausgang</i>
Fehlercode	<b>rbSetLampEx</b> (iHandle, LampNr, OnOff, Power) Setzen eines O-Ausganges einschl. Angabe der Intensität Nur ROBO Interface / Extension <i>ROBO Pro – Entsprechung : 7.1.7 Lampenausgang</i>

Fehlercode	<b>rbRobMotor</b> (iHandle, MotNr, Dir, Speed, ICount) Starten eines M-Ausganges im RobMode (mit angeschlossenem Motor und Impulsrädchen an den zugehörigen I-Eingängen. Der Befehl läuft asynchron und beendet sich bei ICount = 0 bzw. Endtaster (linksdrehend) selber. mit rbGetCounter kann das abgefragt werden. <i>ROBO Pro – Entsprechung : 7.1.6 Motorausgang und 7.1.9 Impulszähler (so in etwa)</i>
Fehlercode	<b>rbRobMotors</b> (iHandle, MotorStatus, SpeedStatus, SpeedStatus16, ModeStatus) Setzen des vollständigen Status aller M-Ausgänge Die evtl. zugehörigen Counter müssen separat gesetzt werden.
ICount	<b>rbGetCounter</b> (iHandle, CounterNr) Lesen eines ImpulsCounter-Standes <i>ROBO Pro – Entsprechung : 7.1.9 Impulszähler (so in etwa)</i>
Fehlercode	<b>rbSetCounter</b> (iHandle, CounterNr, ICount) Setzen eines ImpulsCounter-Standes <i>ROBO Pro – Entsprechung : 7.1.9 Impulszähler (so in etwa)</i>
Fehlercode	<b>rbClearCounter</b> (iHandle) Löschen aller ImpulsCounter auf 0. <i>ROBO Pro – Entsprechung : 7.1.9 Impulszähler (so in etwa)</i>
Auskunftsfunktionen : <i>ROBO Pro – Entsprechung : Interface Test</i>	
int	<b>rbGetActDeviceType</b> (iHandle) Auslesen des aktiven DeviceTyps. Erfolgreiches rbOpenInterface erforderlich
int	<b>rbGetActDeviceSerialNr</b> (iHandle) Auslesen der SerialNr des aktiven Device. Erfolgreiches rbOpenInterface erforderlich
int	<b>rbGetActDeviceFirmwareNr</b> (iHandle) Auslesen der FirmwareNr des aktiven Device. Erfolgreiches rbOpenInterface erforderlich
LPCSTR	<b>rbGetActFirmware</b> (iHandle) Auslesen des Firmwarestrings des aktiven Device Erfolgreiches rbOpenInterface erforderlich
LPCSTR	<b>rbGetActName</b> (iHandle) Auslesen des Namens des aktiven Device Erfolgreiches rbOpenInterface erforderlich

---

## 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 Ausgänge 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 RobolInstanz 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

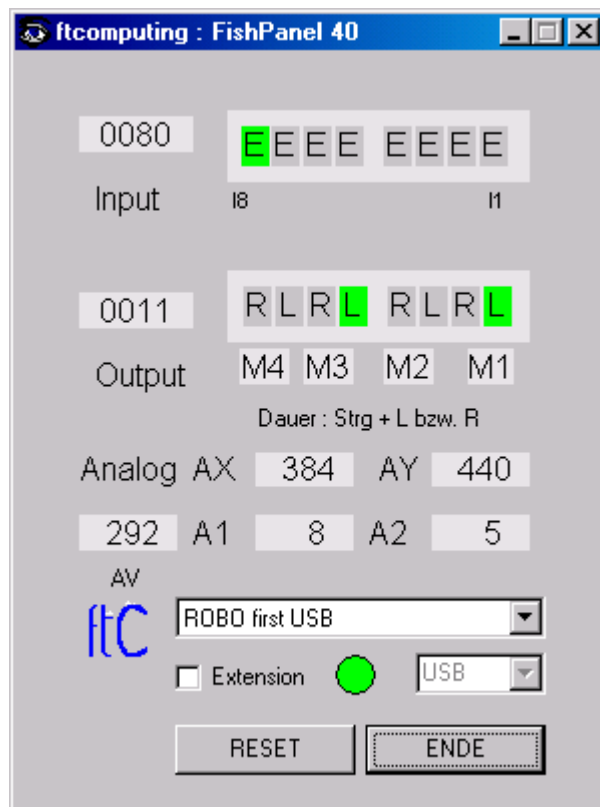
umFish40.DLL liegt in kompilierter Form und als VC++ Projekt vor.

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).

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](mailto: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.

## 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 : Il 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
```



---

# Java

```
package testrahmen;
import ftcomputing.robo.*;

public class TestJF {

    public static void main(String[] args) {
        TestJF testJF1 = new TestJF();
        System.out.println("--- FishROBO-Test gestartet (TestJF) ---");
        testJF1.Action();
        System.out.println("--- FishROBO-Test beendet ---");
    }

    public void Action() {
        JavaFish ft = new JavaFish();
        int iHandle;
        System.out.println("JavaFish angekommen");
        iHandle = ft.jrOpenInterfaceUSB(0, 0);
        int Runde = 0;
        do {
            System.out.println("Runde : " + Runde++);
            ft.jrSetMotors(iHandle, 0x01);
            JavaFish.sleep(333);
            ft.jrSetMotors(iHandle, 0x03);
            JavaFish.sleep(333);
            ft.jrSetMotors(iHandle, 0x07);
            JavaFish.sleep(333);
            ft.jrSetMotors(iHandle, 0x00);
            JavaFish.sleep(333);
        } while((ft.jrGetInput(iHandle, 1) == 0)
                && (JavaFish.escape() == 0));
        ft.jrCloseInterface(iHandle);
    }
}
```

Erstellt auf Basis der Klasse ftcomputing.robo.JavaFish. JavaFish enthält die native Deklarationen für die Wrapper.DLL javaFish40.DLL, die die umFish40.DLL Funktionen javakonform 1:1 kapselt. Hinzukommen die Win32-API Funktionen escape, getTickCount, sleep.

Im Beispielprogramm werden die Lampen an O1 – O3 nacheinander ein- und dann gemeinsam ausschaltet. Das geschieht in einer Endlosschleife, die durch I1 = true oder die ESC-Taste beendet werden kann. Genutzt wird das erste ROBO-Interface an USB.

---

## 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 besonderen Betriebsmodus, dem RobMode. In diesem Modus werden die betroffenen Counter decrementiert. Bei Erreichen des Wertes 0 wird der betroffene 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 zugehörigen 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 aufbereitet 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 : pro Motor 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 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).