fischertechnik Interfaces

umFish40.DLL

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

Documentation for Version 4.3

Ulrich Müller
Contents

umFish40.DLL

Common
Extensions to Version 4.0 3
Extensions to Version 4.1 4
Extensions to Version 4.2 4
Versions 4
Connection of some newer Sensors 4
Functions
Notations 5
Messages 6
Error Handling 6
Function List 7
Notes to the umFish40.DLL Source 10
FtLib Functions 10
umFish40.DLL Functions 10
umFish40.DLL Details 11

Using umFish40.DLL

Common 12
InterfacePanel 13
VC++ 14
C# 15
VB.NET 16
Delphi 17
Visual Basic 6 18
Java 19
Notes to the Counters 20
Notes to the Rob Functions 20
Notes to Radio Control 21

Copyright © 2006 - 2009 Ulrich Müller. Document Name : umFish43e.DOC.
Print Date : 23.09.2009
Bild Einfügen | Grafik | Aus Datei | Office | Fisch6.WMF
umFish40.DLL

Common

umFish40.DLL v4.3.75.0 is based on the FtLib module v1.70a supplied by fischertechnik integrated in umFish40.DLL. The listed Interface are supported in the so called Online Mode (permanent connection to the PC):

- First ROBO Interface at USB (iTyp 0)
- Intelligent Interface (iTyp 10)
- Intelligent Interface with Slave (iTyp 20)
- ROBO Interface, Intelligent Interface Mode (iTyp 50)
- ROBO Interface at USB (iTyp 60)
- ROBO Interface at COM (iTyp 70)
- ROBO Interface via RF (iTyp 80)
- ROBO I/O Extension at USB (iTyp 90)
- ROBO RF Datalink at USB (iTyp 110)
- ROBO Connect Box at USB (iTyp 200)

The ROBO Interface can be attached with up to 3 ROBO I/O Extensions.

Up to now the M- and O-Outputs and the I-, IR-, D- and A-Inputs are supported.

The ROBO Connect Box supports the operating of the Universal - and the Intelligent Interface. Extensions are not supported by the Connect Box.

Several Interfaces can be operated simultaneous (COM / USB mixed too). The special Interface is identified by a handle (iHandle).

In addition to the functions offered by FtLib umFish40.DLL supports an impulse counter for each I-Input, opening and closing of an input are counted separately. Based on this there are RobMotors: A fix combination of motor, end switch an impuls switch -> M1, I1, I2 ... M4, I7, I8.

Goal of the umFish40.DLL developement is building a base for using it from most of the common programming languages. Therefore no C++ typical data types are used. Only int (32bit, signed) variables are used. Never the less some programming languages nee special constructs: Java -> a wrapper.DLL (JNI), script languages -> a ActiveX.DLL. The return values of the type LPCSTR are an expection, they only can be used with VC++.

Not contained in the download package is the ROBO Interface firmware, the USB driver and the FtLib files. They can be downloaded from http://www.fischertechnik.de/computing/software.html.
For installing firmware and USB driver the best is to use ROBO Pro. The FtLib is not needed separately. Alternatively the drivers and new Firmware for the ROBO Connect Box can be downloaded from http://www.ftcomputing.de/zip/RoboConnectBox.zip.

For testing purposes of a new model umFishDP40.EXE is recommended (part of umFish43.ZIP). Note : If a running application is aborted from the IDE it is possible not all storage is released. On too less storage : Boot him.

**Extensions to Version 4.0**

Support for the message exchange between a PC application and an application on the interface itself (e.g. written in Renesas C). Therefore the ROBO Interface firmware v1.64.0.03, for radio layer v0.44.0.03 and ROBO RF Datalink v0.44.0.03 is needed. In other case elder firmware and umFish40.DLLs will do it.

**Extensions to Version 4.1**

In Addition the ROBO Connect Box for operating the Universal- and the Intelligent Interface is supported.

Usually the Connect Box is used as "first ROBO Device to USB". On the side of the application programs that devices work like ROBO Interfaces. Of course they only support the Inputs AX/AY (now Ex/EY) and E1…E8 (now I1 … I8).

**Extensions to Version 4.2**

The inputs D1 and D2 now can be used alternatively as additional voltage inputs (default) or for operating the actual UltraSonic sensor. This version supports Java again.

**Versions**

ROBO Interface Firmware v1.75.0.04 with RF v0.47.0.03
FishFace2005.DLL v4.3.75.2005, umFish40.DLL v4.3.75.0 with FtLib v1.70a

**Connection of some newer Sensors**

See also the fischertechnik documentation for the ROBO Explorer

**DistanceSensor**
Connection wire red/green to D1 or D2, no pole directions.
rbOpenInterfaceUSBdis must be called with ftiDisUltra for use with rbGetDistanceValue(ftiD1 / ftiD2).

**ColorSensor**
Connection wire red to + and green to ground, black to A1/A2 (inner side) for use with rbGetVoltage(ftiA1 / ftiA2)

**TrackSensor**
Connection wire red to + and green to ground. yellow / blue to two different I-Inputs (inner side)
For use via rbGetInputs(), rbGetInput(InputNr)
Functions

Notations

Parameter names and their value range

Value range in brackets: ROBO Interface with 3 Extensions connected.

iHandle Handle to identify the actual Interface (1 – 8).
MotNr Number of an M Output 1 – 4 (16)
LampNr Number of an O Output 1 – 8 (32). "Half" M Output
     Not with the Intelligent Interface.
InputNr Number of an I Input 1 – 8 (32).
     Intelligent Interface : E Input 1 – 8 (16)
Inputwert Value of an I Input 0 / 1
InputStatus State of all (max. 32) I Inputs, I1 right, each 1bit.
CounterNr Number of the Counter of an I Input 1 – 8 (32)
AnalogNr AX / AY / AXS1 / AXS2 / AXS3 : 1 – 2 (5)
     Intelligent Interface EX / EY : 1 – 2
Analogwert Value of an A Input 0 - 1023
VoltNr A1 / A2 / AV / AZ : 1 – 4
     Not with the Intelligent Interface
Dir Rotation direction of the M Outputs:
     Off, Aus = 0, Left, Links = 1, Right, Rechts = 2
MotorStatus Direction values of all motors, M1 right bits, 2bit
Mode Operating mode of a motor. Normal = 0, RobMode = 1
ModeStatus Operation mode of all motors, M1 right bits, 2bit
Speed PWM speed level (M Outputs) : 0 – 7
SpeedStatus Speed level of 8 motors M1 – M8 and M9 – M16, M1 / M9 right bits, 4bit
Power PWM level (O-Ausgänge) : 0 – 7
OnOff On, Ein / Off, Aus : 1 / 0
ICount Value of an impulse counter
ifTyp Type of the Interface (look: Common remarks)
SerialNr Standard serial number of an ROBO Interface.
     Serial number = 0 means first Interface found on USB.
ComNr Number of the COM port the Interface is connected to. 1 – 4
RbFehler Error code rbFehler or 0
int Common 32bit int value, signed.

All parameters are of the type int (32bit, signed)
Messages

Are handle within the structure MessageData:

typedef struct {
    BYTE HwId; Sending mode (always 2 : Send to all other participants)
    BYTE SubId; Class of the message
    USHORT MsgId; Number of the message
    USHORT Msg; The message itself.
} MessageData;

With exception of HwId the parts of the message can be used by own suggestions. A special use besides word borders is possible. Mostly the interpretation of the context is the right way. Testing with VC++ IDE they could be inspected.

Error Handling

All functions have an return value, which is rbFehler (0xE0000001). In case of success 0 or the special return value of the function. The return codes of FtLib are not used because on normal testing they are not very helpfull.
Function List

**iHandle**

**rbOpenInterfaceUSB**(ifTyp, SerialNr)
Making of a connection to an Interface at USB (via ROBO RF Datalink included, ROBO Interface with the same channel number is used).
ifTyp = 0 : first Interface at USB, in this case SerialNr = 0.
*ROBO Pro : 7.1.1 (nearly)*

**iHandle**

**rbOpenInterfaceUSBdis**(iTyp, SerialNr, DistanceMode)
Making of a connection to an Interface at USB (see above).
In addition : DistanceMode = ftiDisUltra - operating of UltraSonicsensors at the inputs D1 / D2, = ftiDisVolt - operating as VoltageInput, = ftiDisOff - deactivated.

**iHandle**

**rbOpenInterfaceRF**(SerialNrInterface)
Making of a RF connection to an Interface via ROBO RF Datalink (first Datalink on USB). For identification of the Interface SerialNrInterface. The Interface must have power but no USB connection.

**iHandle**

**rbOpenInterfaceRFdis**(SerialNrInterface, DistanceMode)
Making of a RF connection to an Interface (see above).
In addition : DistanceMode = ftiDisUltra - operating of UltraSonicsensors at the inputs D1 / D2, = ftiDisVolt - operating as VoltageInput, = ftiDisOff - deactivated.

**iHandle**

**rbOpenInterfaceCOM**(ifTyp, ComNr, AnalogZyklen)
Making of a connection to an Interface at COM ROBO and Intelligent Interface.
*ROBO Pro : 7.1.1 (nearly)*

**rbFehler**

**rbCloseInterface**(iHandle)
Ending an Interface connection
*ROBO Pro : 7.1.e (nearly)*

**A- and I Inputs**

**OnOff**

**rbGetInput**(iHandle, InputNr)
Read the state of the addressed IInput
*ROBO Pro : 7.1.3*

**int**

**rbGetInputs**(iHandle)
InputStatus : State of all I Inputs  (I1 right, 1bit)

**int**

**rbGetAnalog**(iHandle, AnalogNr)
Read of the actual analogous valus of the addressed A Input (AX, AY bzw. EX, EY and AXS1, AXS2, AXS3)
*ROBO Pro : 7.1.4*

**int**

**rbGetIRKey**(iHandle, Code, KeyNr)
Read the state of the address IR Keys of the IR sender

**int**

**rbGetVoltage**(iHandle, VoltNr)
Read of the voltage value of the addressed A Input (A1 – A2, AV)
*ROBO Pro : 7.1.4*

**D-Inputs**

**int**

**rbGetDistanceValue**(iHandle, SensorNr)
Read the actual distance value (cm) of the noted UltraSonicSensor (ftiD1, ftiD2)
Note : Must be preceded by rbOpenInterfaceUSBdis or rbOpenInterfaceRFdis with DistanceMode = ftiDisUltra.
M- und O-Ausgänge :

rbFehler  
rbSetMotor(iHandle, MotNr, Dir)  
Setting of an M Output with Speed = 7  
ROBO Pro : 7.1.6

rbFehler  
rbSetMotorEx(iHandle, MotNr, Dir, Speed)  
Setting of an M Output with specifying the Speed  
Speed will be not work with the Intelligent Interface  
ROBO Pro : 7.1.6

int  
rbGetMotors(iHandle)  
State of all M Outputs (M1 right, 2bit)

rbFehler  
rbSetMotors(iHandle, MotorStatus)  
Setting of all M Outputs (M1 right, 2bit), normal mode, speed = 7

rbFehler  
rbSetMotorsEx(iHandle, MotorStatus, SpeedStatus, SpeedStatus16)  
Setting of all M Outputs, speed included, normal mode

int  
rbGetModeStatus(iHandle, MotNr)  
State of the ModeStatus of one M Output (Normal = 0, RobMode = 1)

rbFehler  
rbSetModeStatus(iHandle, MotNr, Mode)  
Setting the ModeStatus of one Moutput (Normal = 0, RobMode = 1)

rbFehler  
rbSetLamp(iHandle, LampNr, OnOff)  
Setting of an Output with Power = 7  
ROBO Pro : 7.1.7

rbFehler  
rbSetLampEx(iHandle, LampNr, OnOff, Power)  
Setting of an O Output, Intensty included  
ROBO Interface / Extension only  
ROBO Pro : 7.1.7

rbFehler  
rbRobMotor(iHandle, MotNr, Dir, Speed, ICount)  
Starting of a M Output with RobMode (Motor, end switch, impulse switch).  
The function run asynchronous and ends if ICount = 0 or end switch is  
true (on left turning). ICount state can be controlled with rbGetCounter.  
ROBO Pro : 7.1.6 and 7.1.9

rbFehler  
rbRobMotors(iHandle, MotorStatus, SpeedStatus, SpeedStatus16,  
ModeStatus)  
Setting of the complete state of all M Outputs, used counters must be set  
separately.

Impulse Counter :

ICount  
rbGetCounter(iHandle, CounterNr)  
Read the value of an impulse counter  
ROBO Pro : 7.1.9

rbFehler  
rbSetCounter(iHandle, CounterNr, ICount)  
Setting the value of an impulse counter  
ROBO Pro : 7.1.9

rbFehler  
rbClearCounter(iHandle)  
Clear all impulse couters to 0.  
ROBO Pro : 7.1.9
Radio Functions:

- `rbClearMessagesIn(int iHandle)`
  Clearing of the queue of incoming messages
- `rbClearMessagesOut(int iHandle)`
  Clearing of the queue of outgoing messages
- `rbGetMessage(int iHandle, MessageData* inNachricht)`
  Peek of a broadcast message from the queue
- `int rbIsMessage(int IHandle)`
  Check for incoming messages
  - 0 = no, > 0 = number of message or rbFehler
- `rbSendMessage(int iHandle, MessageData* outNachricht)`
  Poke a broadcast message to the output queue
- `rbSendMessageEx(int iHandle, MessageData* outNachricht, int Spez)`
  Conditioned poke of a broadcast message:
  - 0 = always, 1 if new inspect of the last one in the queue,
  - 2 = if not contained in the queue.

Information Functions:

- `rbGetActDeviceType(iHandle)`
  Read of the device type of the active Interface
  Only if rbOpenInterface has been successfull -> rbFehler
- `rbGetActDeviceSerialNr(iHandle)`
  Read of the serial number of the active Interface.
  Only if rbOpenInterface has been successfull -> rbFehler
- `rbGetActDeviceFirmwareNr(iHandle)`
  Read of the firmware number of the active Interface.
  Erfolgreiches rbOpenInterface erforderlich, sonst RbFehler
- `LPCSTR rbGetActDeviceFirmware(iHandle)`
  Read of the firmware string of the active Interface.
  Only if rbOpenInterface has been successfull -> rbFehler
- `LPCSTR rbGetActDeviceName(iHandle)`
  Read of the name of the active Interface.
  Only if rbOpenInterface has been successfull -> rbFehler
Notes to the umFish40.DLL Source

umFish40.DLL is a system.DLL, written in VC++ 6.0, which uses for the real Interface access the functions of FtLib_Static_LIBCMT_Release.lib (supported by fischertechnik). It offers a number of base functions for the access to the Interfaces of the ROBO series and the Intelligent Interface. In addition to the functions offered by FtLib umFish40.DLL supports an impulse counter for each I-Input, opening and closing of an input are counted separately. Based on this there are RobMotors : A fix combination of motor, end switch an impuls switch -> M1, I1, I2 ... M4, I7, I8.

Goal of the umFish40.DLL developement is building a base for using it from most of the common programming languages. Therefore no C++ typical data types are used. Only int (32bit, signed) variables are used. Never the less some programming languages nee special constructs : Java -> a wrapper.DLL (JNI), script languages -> a ActiveX.DLL. The return values of the type LPCSTR are an exection, they only can be used with VC++.

The source contains the following main files :

- umFish40.DEF : Declarations of the DLL entries
- umFish40.H : Declarations of external funtions, interna
- umFish40.CPP : The functions.

FtLib Functions

The FtLib controls the direct access to the Interfaces. Therefore it contains a series of function to control the connection to the Interfaces (OpenFtUsbDevice / OpenFtCommDevice ... StartFtTransferArea ... ) and some information functions (GetFtDeviceType / GetFirmwareStrg ...) and in additon functions for the download of assembled programs to the ROBO Interface ( not used in umFish40.DLL).

The main part of communication with the Interface is done using a communication area – the TransferArea – this area is actualized every 10 ms. It contains the values of the Inputs, Outputs .... In this frequence a CallBack Entry is called. With data in a structure : NOTIFICATION_EVENTS. It is used by umFish40.DLL to supply additional functions.

umFish40.DLL Functions

Main task of umFish40.DLL is to prepare the data of the TransferArea and to convert them to functions. e.g. 'unsigned char E_Main' is converted to the function rbGetInput(InputNr), which notes the state of a single I Input. The Inputs and Outputs are counted – in opposite to FtLib – continously (I Inputs 1 – 32, M Outputs 1 – 16):

The complicate building of an Interface connection is reduced to a single function (rbOpenInterfaceUSB / rbOpenInterfaceCOM). The closing of a connection is reduced to rbCloseInterface.

The additional functions of umFish40.DLL are situated in a CallBack routine. They are extracted from the FtLib structure NOTIFICATION_EVENTS an transformed to functions.

The information functions of FtLib are offered as umFish40.DLL functions as far as it seems to be usefull.

The function Code Download is not supported.
umFish40.DLL Details

umFish40.DLL supports up to 8 Interface connection for simultaneous operating. The instance data therefore are placed in the array ROBOInstanz. The single OpenInterface returns as an handle an index to that array to identify the instance.

A typical access to the TransferArea looks like this:

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

Switch Mn to left

The construction:

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

is used to check for a correct OpenInterface.

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

fis an check for an existing connection to an Interface.

The Inputs and Outputs are counted beginning with 1, internally beginning with 0. Therefore e.g. an MotNr-- is to be found at the beginning of a function.

Masking of Input and Output areas is alternatively done by table and by shifting.
Using umFish40.DLL

Common

umFish40.DLL is delivered as compiled DLL and as VC++ project.

Additionally there are some "H-Files" (declarations, classes) for use in different programming languages. They are supplemented by a typical HelloROBO program (M1 blinking 5 times after I1 comes to true). First ROBO Interface at USB.

umFish40.DLL first is a base for the development of own libraries. Direct using may be complicated because of no DoEvents (interrupt to process Windows messages) or ESC key (cancel) is contained. Some programming languages (Script, Logo, Java ..) can't process system.DLLs. For many languages exits FishFace class libraries based on umFish40.DLL which have more sophisticated functions too.

The Hello projects contain a sample for sending an arbitrary message for demonstration purposes. Nevertheless the Hello can be executed on ROBO Interfaces without radio functions or on ROBO I/O Extensions.
InterfacePanel

Is delivered as EXE file (umFishDP40.EXE). Can be used to control newly assembled fischertechnik model for ist function and for setting the model to a defined position.
New VC++ 6.0 project

1. Menu Files New ...
2. Add the program source
3. Add umFish40.H and umFish40.lib
4. umFish40.DLL (Release Version) must be in an accesible path: Directory Debug of the project or WinNT\System32
5. Project parameter: Win32Debug | C/C++ Kategorie CodeGeneration | LaufzeitBibliothek: Multithreaded.DLL (sorry german version)

```cpp
#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 << "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);
}
```
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 project. The elements of the class umFish40CS are static, in this case no instance is needed.
Imports cs = System.Console
Imports um = HelloVBNETHelloB.rob0.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) & " mit Firmware : " & um.rbGetActDeviceFirmwareNr(iHandle).ToString("X") & "IN ACTION : Start 1 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 project. The elements of the class umFish40VBNET are static, in this case no instance is needed.
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.

Tested with Delphi4, I think it will do with Delphi 2 – Delphi 7.
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"
        Exit Sub
    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
import ftcomputing.robo.*;
public class TestJF {
    public 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;
        iHandle = ft.jrOpenInterfaceUSB(0, 0);
        System.out.println("Action : I1 druecken");
        while((ft.jrGetInput(iHandle, 1)==0) && (JavaFish.escape() == 0))
            do {
                try {
                    System.out.println("Runde : " + Runde++);
                    ft.jrSetMotors(iHandle, 0x01);
                    Thread.sleep(333);
                    ft.jrSetMotors(iHandle, 0x03);
                    Thread.sleep(333);
                    ft.jrSetMotors(iHandle, 0x07);
                    Thread.sleep(333);
                    ft.jrSetMotors(iHandle, 0x08);
                    Thread.sleep(666);
                } catch(InterruptedException e){}
            while((ft.jrGetInput(iHandle, 1) == 0) && (JavaFish.escape() == 0));
        ft.jrSetMotors(iHandle, 0);
        ft.jrCloseInterface(iHandle);
    }
}
Notes to the Counters

An essential element of determining the position are the counters. There is a counter for each I Input (attention: 11 in some languages is 0 in others is 1). The counters will notify (and count) each change of the state of an input (e.g. opening or closing a switch).

The counter can be read and set with special functions. The counter are used internally be some functions (e.g. rbRobMotor).

Notes to the Rob Functions

The Rob function are running in a special operating mode, the RobMode. In this mode the involved counters are decreased. Reaching the value 0, the motor belonging to the counter is switched off.

Operating of a motor in RobMode uses a fix concept of wiring the motors. Each motor is associated with an en switch and an impulse switch:

<table>
<thead>
<tr>
<th>Motor</th>
<th>Endtaster</th>
<th>Impulstaster</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>4</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>5</td>
<td>9</td>
<td>10</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
<td>12</td>
</tr>
<tr>
<td>7</td>
<td>13</td>
<td>14</td>
</tr>
<tr>
<td>8</td>
<td>15</td>
<td>16</td>
</tr>
</tbody>
</table>

And up to 16 if connected to an ROBO Interface with 3 Extensions.

The motors are "left turning". That means, they run in direction of the end switch if turning ftLinks / ftLeft.

A single motors can be operated with rbRobMotor. The parameter ICount stands for the number of impulses to be run. ICount is decreased to 0.

All RobMotors can be operated with one function at the same time: rbRobMotors. The values for all motors come in the parameter:

MotorStatus: each motor 2bit, M1: bit 0 and 1
            00: off, 01 left, 10 right.
SpeedStatus: each motor 4bit, M1: bit 0-3,
            0000 off, 0100 half power, ... 0111 full.
ModeStatus: each motor 2 bit, M1: bit 0-1,
            00 NormalMode, 01 RobMode.

Example: rbRobMotors(ft, 0x9, 0x74, 0x0, 0x05);
0x means Hexa, binary: MotorStatus 1001 SpeedStatus 01110100 ModeStatus 0101 -> M2 = right, Speed 7 with Rob-Mode, M1 = left, Speed 4 with Rob-Mode. Other motors are stopped. Before using rbRobMotors the impulse counter for each RobMotor are to be set.

The motors operate simultaneously (up to 16 motors). They can be switched one after the other by rbRobMotor or all together with rbSetMotors. The stop if their special counter is come to 0. The MotorStatus bits are set to 0. Controlling MotorStatus 0 can be used for synchronizing purposes. 00 of all RobMotor in MotorStatus means all motors are stopped, destination is reached.
Notes to Radio Control

Components of the radio control are the ROBO RF Datalink and ROBO Interfaces with RF card.

There are three different kinds of radio controlled operation:

1. **Route Through**: An Interface with RF card is connected via RF Datalink to the PC. The application will run without knowing the kind of connection. Advantage: A model with Interface and RF card can operate freely in the landscape. Controlling an user interface is situated on the PC.
   This is the kind of radio control supported by umFish40.DLL.

2. **Autonom**: Some Interface with RF card communicate via radio control. In this case the RF Datalink has a role as a *Messages Routers*. The applications are running on the Interfaces.
   Not supported by umFish40.DLL.

3. **Route Through and Message Router**: The first Interface is operated by the PC application, the other run autonomously. The can be reached from the application of the first Interface via radio control.
   This kind of radio control is supported by umFish40.DLL on the side of the PC.
   Programming of the other Interfaces must be done with Renesas C or ROBO Pro.

Therefore umFish40.DLL supports some special functions:

- `rbSendMessage` for sending a buffered broadcast message.
- `rbGetMessage` and `rbIsMessage` to receive incoming messages from the input queue. In addition `rbClearMessagesIn`, `rbClearMessagesOut` for clearing the message queues.
- The structure `MessageData` for transporting the message data. `HwId` contains the kind of sending (RF Broadcast via radio, Code 2). Other fields of the structure can be used free.

The Hello Demos contain a `rbSendMessage` for a very simple attempt. But therefore some more preparation must be done:

- **ROBO RF Datalink with RF2/0 at USB**
  The Datalink must only be present.

- **ROBO Intelligent Interface with radio card (with RF2/1) connected only to power and switch at I1 and a lamp on M1. The Interface is controlled by the PC. A message is to be send and the lamp is blinking.**

- **ROBO Intelligent Interface with radio card (with RF2/2) connected only to power and a motor on M1. Loaded with an application (Renesas C / ROBO Pro) waiting in an endless loop for a message. Receiving a message – without looking to the contents – the motor on M1 is started for one second, left direction.**

At Start of the demo the motor will run (RF 2/2), afterwards blinking is done.