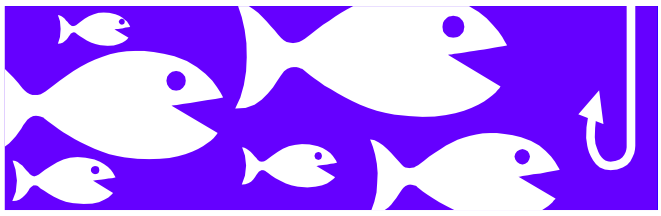
ftComputing

CFishFace for VC++

2. Edition, based on umFish40.DLL v4.1.59.1

Ulrich Müller



Content

General

Supported Interfaces and how to use them

umFish40.DLL supports the ROBO devices and the Intelligent Interface. The manual describes using their function via umFish40.DLL v4.1.59.1 by VC++ 6.0 programs in the so called "Online Mode", that means with programs running on the PC. There are two different ways to do it :

- The direct access to the umFish40 functions via umFish40VC.H.
  This is of interest, if you want to write your own class library.
- Using the class CFishFace based on umFish40.DLL via FishFace40.H / CPP.
  This is intended to be the usual way.

The supported interface in detail are :

- First ROBO Device on USB (ifType 0)
- Intelligent Interface (ifType 10)
- Intelligent Interface with Slave (ifType 20)
- ROBO Interface in Intelligent Interface Mode (ifType 50)
- ROBO Interface on USB (ifType 60)
- ROBO Interface on COM (ifType 70)
- ROBO Interface via RF Datalink (ifType 80)
- ROBO I/O Extension Module on USB (ifType 90)
- ROBO RF Datalink on USB (ifType 110)

The ROBO Interface (ifType 60, 70 and 80) can be supplemented with up to 3 ROBO I/O Extensions.

It is possible to run some Interfaces (USB / COM mixed too) simultaneously. The application references the single Interface by an handle (umFish40.DLL) or an instance of the class CFishFace.

umFish40.DLL is based on the FtLib supported by www.fischertechnik.de. In addition to their functions umFish40.DLL supports impulse counters which register all changes of the state of the I-Inputs. And the combination M-Output and I-Input operated by an impulse wheel are called RobMotors. For more details see also the Notes at the end.

The sources and examples are contained in www.ftcomputing.de/zip/ccfish40.zip.

In addition to it you need USB-Driver and – possibly – actual Firmware. Yout get it on www.fischertechnik.de Computing | Downloads. The USB-Driver install themselve. The Firmware must be updated by ROBO Pro | Interface Test.

In every cases you need umFish40.DLL with umFish40.LIB. For some more details to umFish40.DLL look to http://www.ftcomputing.de/zip/umfish40.zip and www.ftcomputing.de/ccfishe.htm .

HelloFish – Console Project with umFish40.DLL

Setting up the VC++ 6.0 Project

Easiest way : copy the whole ccFish40.ZIP in a new directory

Or : build yourself a new project

- New Workspace : Console
- Copy umFish40.DLL to \Debug or \Release or alternatively to \WinNT\System32
- Insert umFish40VC.h, umFish40.lib, umFish40VC.cpp
- Compile (F7)

To connect the Interface

The interface is suggested to be a single ROBO Interfaceon USB.

The HelloFish project expects 3 lamps on M1 to M3.

The Source

```
#include <windows.h>
#include <iostream.h>
#include "..\umFish40VC.h"

void main()
{
   char END;
   cout << "---- HelloFish started ----" << endl;

// --- Connection to the interface --------------------------------
   int iHandle = rbOpenInterfaceUSB(0, 0);
   if(iHandle == rbFehler) {
     cout << "Interface Problem, exit" << endl;
     cout << endl << "--- FINIS : Return Key ---" << endl;
     cin.get(END);
     return;
   }
   cout << "Interface : " << rbGetActDeviceName(iHandle)
        << ", Type : " << rbGetActDeviceType(iHandle)
        << ", SerialNr : "
        << rbGetActDeviceSerialNr(iHandle) << endl;
   cout << "Firmware  : " << rbGetActDeviceFirmware(iHandle) << endl;

   cout << "--- Loop for the three lamps on M1 to M3" << endl;
   for(int j=1; j<=4; j++) {
     cout << "Round : " << j << endl;
     rbSetMotors(iHandle, 0);          // --- all Motors off
     Sleep(300);                       // --- 0.3 sec Pause
     for(int i=1; i<=3; i++) {
       rbSetMotor(iHandle, i, 1);      // --- switch on Lamp on Mi
       Sleep (500);
     }
   }

// ------------------------- End ----------------------------------
  rbCloseInterface(iHandle);
```

```
  cout << endl << "--- FINIS : Return Key ---" << endl;
  cin.get(END);
}
```

Explanations :

`#include <windows.h>` : Standard include
`#include <iostream.h>` : include for cout / cin
`#include "..\umFish40VC.h"` : umFish40.DLL declarations.

```
int iHandle = rbOpenInterfaceUSB(0, 0);
....
```

Connection to the interface. Parameters : first ROBO Interface on USB, no special SerialNo. Returns the Handle to umFish40. If it is == ftiFehler the connection failed.

```
for(int j=1; j<=4; j++) {...}
```

Repeat lamp switching 4 times

```
rbSetMotors(iHandle, 0);
Sleep(300);
for(int i=1; i<=3; i++) {
  rbSetMotor(iHandle, i, 1);
  Sleep (500);
}
```

Clear all M-Outputs (Lamps) and pause for 0.3 secs.
Switch on lamps on M1 – M2 – M3, pause after each switch for 0.5 secs.

```
rbCloseInterface(iHandle);
cout << endl << "--- FINIS : Return Key ---" << endl;
cin.get(END);
```

Ending the program :
– Cancel the connection to the interface.
– Write message
– Wait for Return Key.

Some more Examples

Some more examples for using umFish40.DLL can be found in ccfish40.zip, directory \USamples40.

The class CFishFace

Common

The source for CFishFace contains the following files :

- FishFace40.H : The header file with enums and the class definitions CFishFaceException and CFishFace and some inline functions for property like methods.
  FishFace40.CPP : the constructors and method implementations for CFishFace.

The methods which are running for a longer time (Waitxxx, Pause ...) can be canceled by pressing the ESC key.

Most methods are throwing an excepiton for "KeinOpen.methodname" (OpenInterface is missing) and "InterfaceProblem.methodname" (some problem with the interface – may be power off, no connection to the serial port).

CFishFace can be used with console applications as well as with MFC applications. In case of MFC the application form may be sometimes to be 'frozen' because of CFishFace methods are running in a very close loop. In this case applications must give control to the message queque. An other nice solution is to run the CFishFace part of the application in a separate thread (a C++Builder solution for that is described in www.ftcomputing.de/ccthreade.htm).

Using CFishFace

Here are described some small examples in the manner of tips & trics.

The Program Frame

Is a console application in the manner of that before. It is build with the VC++ 6.0 IDE as an empty console project, if it is not copied from ccFish30.ZIP path FiFa30CCP. All the samples are separate routines listed on the beginning of the program. Followed by the main routine with a call for one of them. This call is to be replaced by a call for the interesting routine to test the routine. The source is contained in FiFa30Test.CPP, project directory is FiFa30CPP :

```
#include <windows.h>
#include <iostream.h>
#include "..\FishFace40.h"

CFishFace ft;

void SampleRoutine() {
......
}

void main() {
```

```
  cout << "--- FiFaTest started ---" <<  endl;
  cout << "To end press Esc Key" << endl;

  try {
    ft.OpenInterface(ftROBO_first_USB, 0);

    SampleRoutine();

  }
  catch(CFishFaceException& fte) {
    cout << "Error " << fte.Nr() << " : " << fte.Text() << endl;
  }

  while(GetAsyncKeyState(VK_ESCAPE) == 0);
  ft.CloseInterface();
}
```

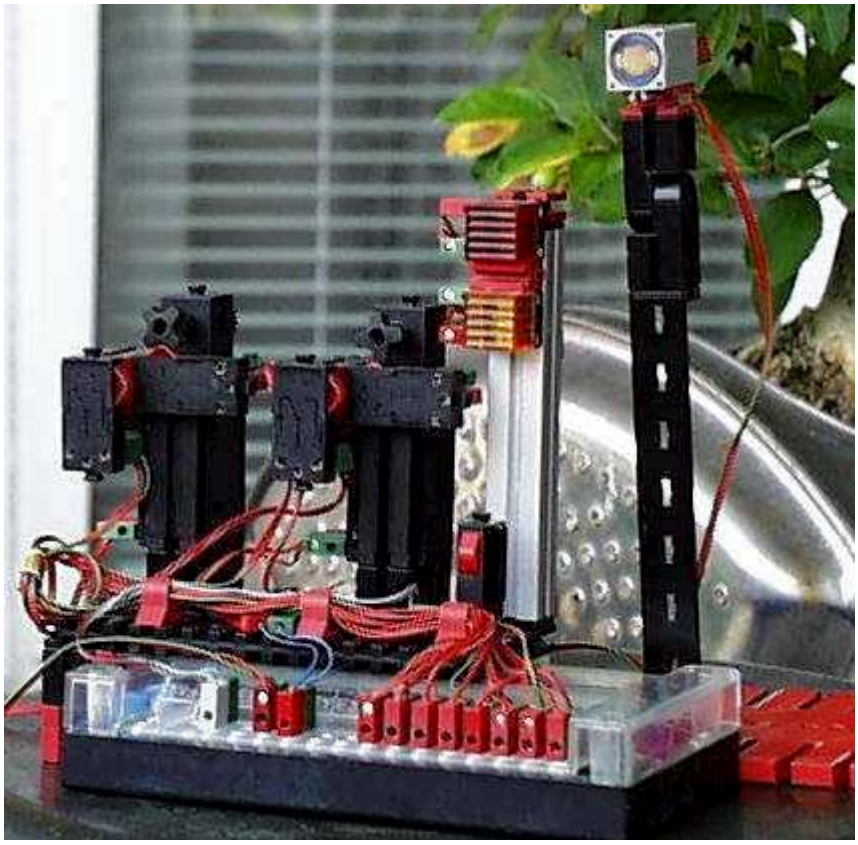
#include windows.h and iostream.h are already known VC++ includes. #include "..\FishFace40.h" includes the header file of the class CFishFace.

Main begins with a start message and the notice : to end press Esc Key. Next is a try ... catch block with ft.OpenInterface(ftROBO_first_USB, 0); for the connection to Interface. May be a ROBO Interface or ROBO I/O Extension or ROBO RF Datalink in combination with a ROBO Interface followed by the call for the sample. The catch block works with the special CFishFaceException and write an error message on the console. It works for errors thrown by OpenInterface and that from the sample routines.

The main routine ends with waiting for an Esc press and a CloseInterface connection.

Common Techniques

This techniques are based on the class CFishFace. To test them you can use a simple model like this :



O1 : red lamp (second pin connected to ground)
O2 : yellow lamp (second pin connected to ground). Addional O3 and O4
M3 : motor with end switch on I5 and impulse switch on I6
M4 : motor with end switch on I7 and impulse swicht on I8
AX : photoresistor
AY : resistor ...
I1 : switch

BlinkingLoop

Lamp on O1 (second pin to ground) is blinking on sec intervals :

```
void BlinkingLoop() {
  do {
     ft.SetLamp(outO1, dirOn);
     ft.Pause(555);
     ft.SetLamp(outO1, dirOff);
     ft.Pause(444);
  } while(!ft.Finish());
}
```

The parameters can be simple int numbers, but they should be named. This names are from the enums of CFishFace. You can use your own names to : `const int mRed = 1` instead of outO1, that is more instructive.

Mostly a program own an all over loop. In this case is that do { ... } while(!ft.Finish()); : The method Finish looks if there is an cancel request. May be the ESC key or optional an I-Input.

AlternateBlinking

Lamps on O1 and O2 are blinking alternating.

Version 1 :

```
void AlternateBlinking1() {
  do {
    ft.SetLamp(outO2, dirOff);
    ft.SetLamp(outO1, dirOn);
    ft.Pause(444);
    ft.SetLamp(outO1, dirOff);
    ft.SetLamp(outO2, dirOn);
    ft.Pause(444);
  } while(!ft.Finish());
}
```

Version 2 (more compact) :

```
void AlternateBlinking2() {
  do {
    ft.SetMotors(0x1);
    ft.Pause(333);
    ft.SetMotors(0x2);
    ft.Pause(333);
  } while(!ft.Finish());
}
```

In this case all M-Outputs are switch with one method : SetMotors. The parameter of SetMotors is a MotorState which contains the states of all O-Outputs (extension module included), each with 1 bit. That means : 00000001 O1 dirOn and 00000010 O2 dirOn. All other O-Outputs are off.

Testing an I-Input

If I1 is true --- On --- is written else Off is written :

```
void TestingAnInput() {
  do {
    if(ft.GetInput(inpI1)) cout << "--- ON ---" << endl;
    else cout << "--- OFF ---" << endl;
    ft.Pause(555);
  } while(!ft.Finish(inpI5));
}
```

Wait for an I-Input

If I1 is true ---- STARTED ---- is written :

```
void WaitForInput() {
  cout << "For start press I1" << endl;
  ft.WaitForInput(inpI1);
  cout << "--- STARTED ---" << endl;
}
```

Display all I-Inputs

Continous display off all I-Inputs :

```
void DisplayAllInputs() {
  do {
    for(int i = 0x80, E = ft.GetInputs(); i > 0; i >>= 1) {
      cout << ((E & i) > 0) ? "1" : "0";
    }
    cout << endl;
    ft.Pause(1234);
  } while(!ft.Finish());
}
```

It's a nice very C-like, very short routine, but's a little bit sophisticated :

The for statement is nearly the whole program to transform the max. 16 E-Inputs to a printable bit stream. The index i contains a mask to select on I-Input, starting with I16, E contains all I-Inputs. for ends with i <= 0, each loop the mask is shifted one bit to right to give the next mask.
cout sequences the bit beginning with I16 and cout << endl; (behind the loop) resumes them.

Display Analog-Inputs

Continous display of EX / EY :

```
void AnalogDisplay() {
  do{
    cout << "AX : " << ft.GetAnalog(anaAX) << " AY : "
                    << ft.GetAnalog(anaAY) << endl;
    ft.Pause(1111);
  } while(!ft.Finish());
}
```

Drive a Motor for a fix Time

```
void DriveForTime() {
  ft.SetMotor(outM3, dirLeft);
  ft.Pause(3500);
  ft.SetMotor(outM3, dirOff);
  cout << "Gone for 3.5 sec" << endl;
}
```

Drive to an End Switch

Motor on M3 will run to end switch I5 and than stop :

```
void DriveToEndSwitch1() {
   ft.SetMotor(outM3, dirLeft);
   while(!ft.GetInput(inpI5));
   ft.SetMotor(outM3, dirOff);
   cout << "Switch I5 true" << endl;
}
```

The better solution is :

```
void DriveToEndSwitch2() {
   ft.SetMotor(outM3, dirLeft);
   ft.WaitForInput(inpI5);
   ft.SetMotor(outM3, dirOff);
   cout << "Switch I5 true" << endl;
}
```

Motor can be stopped by ESC key.

Drive for a fix Number of Steps

WaitForChange

Motor on M3 with impulse switch on I6 will run 12 steps :

```
void DriveForStepsC() {
   ft.SetMotor(outM3, dirLeft);
   ft.WaitForChange(inpI6, 12);
   ft.SetMotor(outM3, dirOff);
}
```

WaitForPositionDown

Motor on M3 drives from actual position 12 to destination position 0,
impulse count with I6 in direction of 0 (end switch = I5) :

```
void DriveForStepsD() {
   int ActPosition = 12;
   ft.SetMotor(outM3, dirLeft);
   ActPosition = ft.WaitForPositionDown(inpI6,
                 ActPosition, 0, inpI5);
   ft.SetMotor(outM3, dirOff);
   cout << "ActPosition : " << ActPosition << endl;
}
```

The actual position after Motor is stopped is returned in ActPosition (may be one step more or less) . The method WaitForPositionDown can be stopped, if reaching I5 before the count position 0 is reached.

WaitForPositionUP

Motor on M3 drives from actual position 12 to destination position 24, impulse counting in direction leving the end switch :

```
void DriveForStepsU() {
  int ActPosition = 12;
  ft.SetMotor(outM3, dirRight);
  ActPosition = ft.WaitForPositionUp(inpI6, ActPosition, 24);
  ft.SetMotor(outM3, dirOff);
  cout << "ActPosition : " << ActPosition << endl;
}
```

The actual after the method is noted in ActPostion.

WaitForMotors

Motor on M3 runs for 12 impulses on I6 with reduced speed to left :

```
void DriveForStepsW() {
  ft.SetMotor(outM3, dirLeft, speHalf, 12);
  ft.WaitForMotors(0, outM3);
  cout << "ActPosition += 12" << endl;
}
```

The program waits for reaching the destination. If ft.WaitForMotors is dropped, the programm does every thing else and Motor will stop too, if destination is reached.

Two motors (M3 – full speed 121 impulse to left and M4 half speed 64 impulses to right. Impulse count on I6 / I8) simultaneously with contious display of the actual position :

```
void DriveForStepsW34() {
  ft.SetMotor(outM3, dirLeft,  speFull, 121);
  ft.SetMotor(outM4, dirRight, speHalf, 64);
  do {
    cout << "Position M3 - M4 : " << ft.GetCounter(inpI6) << " - "
      << ft.GetCounter(inpI8) << endl;
  } while(ft.WaitForMotors(300, outM3, outM4) == ftiTime);
    cout << "Position M3 - M4 : " << ft.GetCounter(inpI6) << " - "
      << ft.GetCounter(inpI8) << "   --- Final ---" << endl;
}
```

ft.WaitForMotors controls the position of M3 / M4 and return every 0.3 secs until waiEnd or waiESC. The loop is to display the actual position (ft.GetCounter(..);). If finished the final position is displayed. Notice a value 0 for waiting time means endless waiting (see sample before).

Light Barriers

Wait for broken Light Barrier

Light barrier with lamp on M1 an phototransistor on I1 :

```
void WaitForLightBarrierBroken() {
  const int mLight = 1, ePhoto = 1;
  ft.SetMotor(mLight, dirOn);
  ft.Pause(555);
  ft.WaitForInput(ePhoto, false);
  cout << "LightBarrier M1 - I1 is broken" << endl;
}
```

Lamp is switched on, 0.5 waiting for 'warming up' the phototransistor, than waiting for a broken barrier.

Wait for Enter a Light Barrier

Barrier with M1 and I1, feeder motor M3 :

```
void WaitForLightBarrierEnter() {
   const int mLight = 1, mFeeder = 3, ePhoto = 1;
   ft.SetMotor(mLight, dirOn);
   ft.Pause(555);
   ft.SetMotor(mFeeder, dirLeft);
   ft.WaitForLow(ePhoto);
   ft.SetMotor(mFeeder, dirOff);
   cout << "LightBarrier M1 - I1 is entered" << endl;
}
```

Light barrier is clear when program starts, feeder (with an parcel) runs until the parcel breaks the barrier.

Wait for Leaving a Light Barrier

Barrier with M1 and I1, feeder motor M3 :

```
void WaitForLightBarrierLeave() {
   const int mLight = 1, mFeeder = 3, ePhoto = 1;
   ft.SetMotor(mLight, dirOn);
   ft.Pause(555);
   ft.SetMotor(mFeeder, dirLeft);
   ft.WaitForHigh(ePhoto);
   ft.SetMotor(mFeeder, dirOff);
   cout << "LightBarrier M1 - I1 is free" << endl;
}
```

Light barrier is broken when program starts, feeder (with a parcel) runs until the parcel is clear off the barrier.

Switching all M-Outputs at once

SetMotors can switch all M-Outputs with one stroke. Therefore the parameter MotorStatus must have the right values, 2 bit for each M-Outputs, right beginning with M1 : 00 00 00 00 mean all M-Outputs are off (with extension module additional four 00's). 01 means turn left, 10 turn right.10 01 00 00 for example M4 right, M3 left others off.

Traffic Lights

The phases are Green – Yellow – Red – RedYellow. The lamps are on O1 green, O2 yellow, O3 red. The constants needed : mGreen = 00 00 01 00 (0x4), mYellow = 00 00 00 10 (0x2) and mRed = 00 00 00 01 (0x1) :

```
void TrafficLights() {
   const int mGreen = 0x4, mYellow = 0x2, mRed = 0x01;
   while (!ft.Finish()) {
      ft.SetMotors(mGreen);
      ft.Pause(1000);
      ft.SetMotors(mYellow);
      ft.Pause(250);
      ft.SetMotors(mRed);
      ft.Pause(1000);
      ft.SetMotors(mRed + mYellow);
      ft.Pause(250);
   }}
```

List controlled Traffic Lights

Using a constant interval, it is possible to control the lamps by a list of switch values :

```
void TrafficLightsList() {
  const int mGreen = 0x4, mYellow = 0x2, mRed = 0x01;
  int Phase[] = {mGreen,  mGreen, mGreen, mGreen,
         mYellow, mRed,   mRed,   mRed,   mRed,   mRed + mYellow};
  while(!ft.Finish()) {
    for(int i = 0; i <= 9; i++) {
      ft.SetMotors(Phase[i]);
      ft.Pause(500);
    }}}
```

In this case the interval is 0.5 secs. This procedure is useful by more complex applications.

Running Lights

If you have connected 4 lamps on the interface (O1 ... O4) you can try a nice running light :

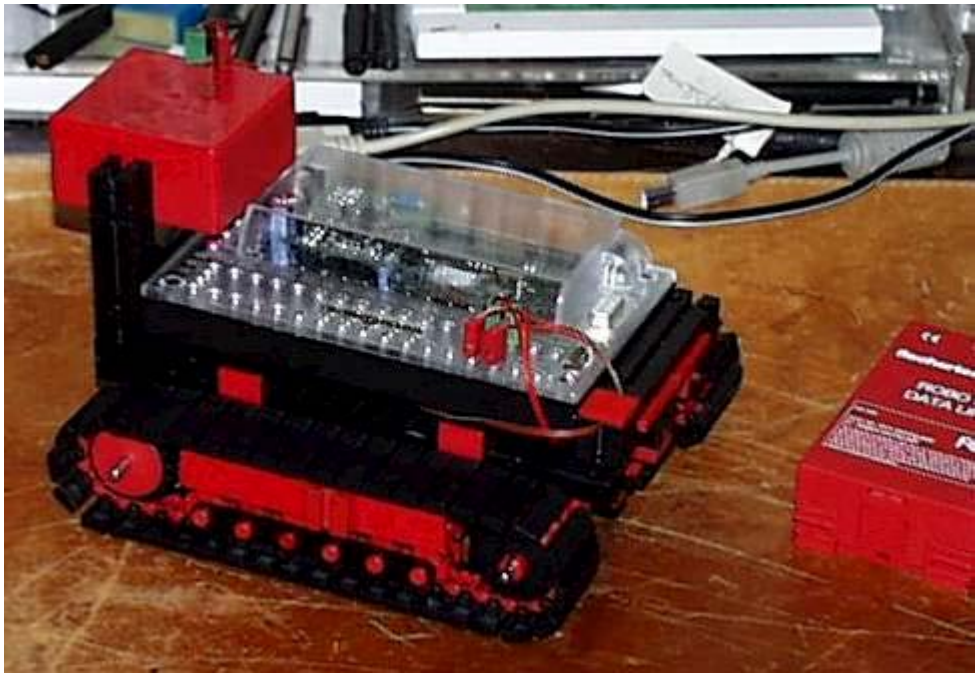
```
void RunningLights() {
  while(!ft.Finish()) {
    for(int Phase = 1; Phase < 0x10; Phase <<= 1) {
      ft.SetMotors(Phase);
      ft.Pause(555);
    }}}
```

Phase is running index and MotorStatus. Only one lamp is switched on at the same time. To do that a 01 bit combination is shift through the MotorStatus, beginning with 01.

Radio Controlled Techniques

Just one example :

Remote Control of a Bulldozer



Configuration

- PC program for the remote control of the bulldozer ROBO Interface RF 2/1 via RF Datalink RF 2/0
- ROBO RF Datalink RF 2/0 on USB for Route Through to RF 2/1 and Message Routing to RF 2/2
- ROBO Interface with radio card RF 2/1 for remote control of the bulldozer with the switches on I1 .. I4 and lamps on O1 and O2 for state display.
  Control commands : for, back, right, left.
- ROBO Interface with radio card RF 2/2 for operating the motors of the bulldozer.

PC program (VC++ 6.0, FunkRaupe4)

```
#include <windows.h>
#include <iostream.h>
#include <string>
#include "FishFace40.h"

const USHORT cVor    = 0x0A00;
const USHORT cRueck  = 0x0500;
const USHORT cLinks  = 0x0900;
const USHORT cRechts = 0x0600;
const USHORT cAus    = 0x0000;

CFishFace ft;

void main() {
  cout << "--- FunkRaupe4 gestartet ---" <<  endl;
```

```
  cout << "To end : ESC Key" << endl;

  try {
    ft.OpenInterface(ftROBO_first_USB, 0);
    MessageData outN;
    outN.HwId  = 2;
    outN.SubId = 1;
    outN.MsgId = 0x01;
    outN.Msg   = cAus;
    ft.SetLamp(outO1, dirEin);
    cout << "Ready, waiting for your orders : I1 .. I4" << endl;
    do {
      if(ft.GetInputs() != 0) {
        if(ft.GetInput(inpI1))
          outN.Msg = outN.Msg == cVor ? cAus : cVor;
        else if(ft.GetInput(inpI2))
          outN.Msg = outN.Msg == cLinks  ? cAus : cLinks;
        else if(ft.GetInput(inpI3))
          outN.Msg = outN.Msg == cRechts ? cAus : cRechts;
        else if(ft.GetInput(inpI4))
          outN.Msg = outN.Msg == cRueck  ? cAus : cRueck;
        ft.SendRFMessage(outN);
        if(outN.Msg != cAus) {
            ft.SetLamp(outO1, dirAus);
            ft.SetLamp(outO2, dirEin);
            cout << "--- Busy ---" << endl;
        }
        else {
            ft.SetLamp(outO1, dirEin);
            ft.SetLamp(outO2, dirAus);
            cout << "--- Ready ---" << endl;
        }
      }
      ft.Pause(555);
    } while(!ft.Finish());
    ft.CloseInterface();
  }
  catch(CFishFaceException& fte) {
    cout << "Error " << fte.Nr() << " : " << fte.Text() << endl;
  }
}
```

Above the complete VC++ 6.0 application is listed (without the CFishFace class file).

- Constants for the bulldozer control commands (all M-Outputs are switched at same time)
- Declare of the structure for outgoing messages outN:
  The application in the Interface RF 2/2 only uses outN.Msg, others : nice to see.
- Endless loop asking for switched to be pressed (GetInputs()).
- If construct for recognizing the single I-Input and assigned the command, belonging to.
- Sending of the modified message.
- If Msg != off, lamp busy is switched on otherwise lamp ready.
- Pause to separate the commands. I think if ft.SendRFMessage(outN, 1) is used instead of the simple ft.SenRFMessage(outN), the Pause can be dropped, not yet tested.

RF 2/2 program (Renesas C)

```
#include "TA_Firmware\TAF_00D.h"
#include "TA_Firmware\TAF_00P.h"
#include "Message\Msg_00D.h"
#include "Message\Msg_00P.h"

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;
      }
   } while(1);
   return(0);
}
```

The program ist using the firmware of the ROBO Interface. Control is done from an endless do loop which asks for a message without addional control and uses 8 bit of inMessage as a command for the M-Outputs.

Reference

umFish40 – Functions

Used Variable Types

iHandle	Handle to identify the actual umFish40 instance (1 – 8).
MotNr	Number of a M-Output (1 – 4(16))
LampNr	Number of a O-Output (1 – 8(32)). A "half" M-Output. Only for ROBO's.
InputNr	Number of an I-Input (1 – 8(32)) Intelligent Interface (1-8(16))
Inputvalue	Value of an I-Input 0 / 1
InputStatus	State of all (max. 32) I-Inputs, I1 right, one bit for each.
CounterNr	Number of the counter assigned to an I-Input (1 – 8(32))
AnalogNr	AX / AY / AXS1 / AXS2 / AXS3 : 1 – 2 (5) Intelligent Interface EX / EY : 1 – 2
AnalogValue	Value of an A-Input (0 – 1023)
VoltNr	A1 / A2 / AV / AZ : 1 – 4 Not on Intelligent Interface
Dir	Revolving direction of an M-Output : Off = 0, Left = 1, Right = 2
MotorStatus	DirValues of all Motors, M1 right bits, 2bit
Mode	Operating mode of an motor. Normal = 0, RobMode = 1
ModeStatus	**(**Operating)Mode of all motors, M1 right bits, 2bit
Speed	PWM speed level (M-Outputs) : 0 – 7
SpeedStatus	Speed level of 8 succeeding motors M1 .. M8 or M9 .. M16. M1 or M9 are the right one. 4bit
Power	PWM intensity of the O-Outputs : 0 – 7
OnOff	On / Off : 1 / 0
ICount	Value of an impulse counter
ifTyp	Interface type (see notes at the beginning)
SerialNr	Active serial number of an USB Interface. Serial number = 0 means : first (or only) Interface found on USB.
ComNr	Number of the COM-Port (1 – 4) the Interface is connected.
Errorcode	Errorcode rbFehler or a correct return value or 0
int	A 32bit Integer-value, signed.

All variables have the type int (32bit, signed).The value range in brackets is that for a fully extended ROBO Interface with 3 I/O Extensions.

Messages

Are handled within the structe MessageData :

```
typedef struct {
  BYTE HwId;        Code of sending type (in this case allways 2 : send to all others)
  BYTE SubId;       class of the message
  USHORT MsgId;     Number of the message
  USHORT Msg;       The message itself
} MessageData;
```

HwId excluded, all parts of a message can be use free, an other sense of their parts is possible, if the partner in the ROBO Interface does accept it (especially if it is programmed in ROBO Pro).

Error Handling

All functions give an return value which is in cas of error rbFehler (0xE0000001). If the function succeeded it return a special value (like an analog value) or 0, if there is none. The long list of error code which come from FtLib are not used, because of an application runs or not, mostly the the context depended interpretation will lead to its reason.

Functions

iHandle	**rbOpenInterfaceUSB**(ifTyp, SerialNr) Setting up a connection to an interface connected to USB, if ifTyp = 0 the first available is choosen (SerialNr = 0). If it is a ROBO RF Datalink, the first available interface with radio card and corresponding RF channel number is choosen, the subchannel number can be one in the range of 1 – 8.
iHandle	**rbOpenInterfaceRF**(SerialNrInterface) Setting up a connection to the USB Interface with the noted SerialNumber via a ROBO RF Datalink (first one anywhere). The interface must have power, but no USB connection (ifTypes 110 with 80).
iHandle	**rbOpenInterfaceCOM**(ifTyp, ComNr, AnalogZyklen) Setting ap a connection to an Interface via the noted COM-Port (ifTypes 10, 20, 50 und 70)
Errorcode	**rbCloseInterface**(iHandle) Ending a connection to an interface
A- and I-Inputs	
OnOff	**rbGetInput**(iHandle, InputNr) Read the state of the noted I-Input.
int	**rbGetInputs**(iHandle) State of all I-Inputs (I1 is the rightone bit)
int	**rbGetAnalog**(iHandle, AnalogNr) Read the value of the noted A-Input (AX, AY bzw. EX, EY und ggf. AXS1, AXS2, AXS3)
int	**rbGetIRKey**(iHandle, Code, KeyNr) Read the state of the noted IR-Key of the IR transmitter
int	**rbGetVoltage**(iHandle, VoltNr) Read the value in volts of the noted A-Input (A1 – A2)
M- and O-Outputs :	

Errorcode	**rbSetMotor**(iHandle, MotNr, Dir) Setting of an M-Output. Noted direction, default speed = 7
Errorcode	**rbSetMotorEx**(iHandle, MotNr, Dir, Speed) Setting of an M-Output with noted direction and speed (ROBO only)
int	**rbGetMotors**(iHandle) Read the state of all M-Outputs (M1 is the right one, 2bit)
Errorcode	**rbSetMotors**(iHandle, MotorStatus) Set the state of all M-Outputs (M1 is the right one, 2bit), NormalMode, default speed = 7
Errorcode	**rbSetMotorsEx**(iHandle, MotorStatus, SpeedStatus, SpeedStatus16) Set all M-Outputs, speed included, NormalMode
int	**rbGetModeStatus**(iHandle, MotNr) Read the ModeState of an M-Output (Normal = 0, RobMode = 1)
Errorcode	**rbSetModeStatus**(iHandle, MotNr, Mode) Set the ModeState of an M-Output (Normal = 0, RobMode = 1)
Fehlercode	**rbSetLamp**(iHandle, LampNr, OnOff) Set an O-Output to OnOff, default power = 7.
Fehlercode	**rbSetLampEx**(iHandle, LampNr, OnOff, Power) Set an O-Output to OnOff and Power (ROBO only)
Fehlercode	**rbRobMotor**(iHandle, MotNr, Dir, Speed, ICount) Start of a M-Output in RobMode with motor on MotNr and assigned I-Inputs for impulse wheel and end switch (see alse Notes for RobMode). The function runs asynchrone. Motor ends with reaching ICout by itself. The state of ICount can be controlled by rbGetCounter.
Errorcode	**rbRobMotors**(iHandle, MotorStatus, SpeedStatus, SpeedStatus16, ModeStatus) Set the complete state of all M-Outputs. Used counters must be set separate.

ImpulseCounter :

ICount	**rbGetCounter**(iHandle, CounterNr) Read the value of a counter
Errorcode	**rbSetCounter**(iHandle, CounterNr, ICount) Set an impulse counter to ICount
Errorcode	**rbClearCounter**(iHandle) Clear all impulse counters to 0.

Radio Functions :

Errorcode	**rbClearMessagesIn**(int iHandle) Clear the queque of incoming messages
Errorcode	**rbClearMessagesOut**(int iHandle) Clear the queque of outgoing messages
Errorcode	**rbGetMessage**(int iHandle, MessageData* inMessage) Read the topmost message from incoming message queque
int	**rbIsMessage**(int IHandle) Are there any incoming messages in the queque 0 = none, > 0 = number of messages or Errorcode
Fehlercode	**rbSendMessage**(int iHandle, MessageData* outMessage) Place a (broadcast) message in the out queque
Fehlercode	**rbSendMessageEx**(int iHandle, MessageData* outNachricht, int Spez) Conditioned placing of an out message in the queque Spez : 0 = always, 1 = if new inspect of the last one 2 = if not contained in the out queque

Information Functions :

int	**rbGetActDeviceType**(iHandle) Read the active device type. Only if rbOpenInterface comes true, otherwise Errorcode
int	**rbGetActDeviceSerialNr**(iHandle) Read the active serial number Only if rbOpenInterface comes true, otherwise Errorcode
int	**rbGetActDeviceFirmwareNr**(iHandle) Read the active firmware number Only if rbOpenInterface comes true, otherwise Errorcode
LPCSTR	**rbGetActFirmware**(iHandle) Read the active firmware string Only if rbOpenInterface comes true, otherwise NULL
LPCSTR	**rbGetActName**(iHandle) Read the name of the active device Only if rbOpenInterface comes true, otherwise NULL

class CFishFace

Frequently used Variables

The variable are mostly of the type int. Here is given a short description of those variable use in the CFishFace reference following. The enum values are used to describe the value range of the variables. The enum name is noted in brackets.

All parameters are value parameters.

AnalogNr	Number of an Analog-Input (Nr) AX = 1, AY = 2
AnalogValue	Return value from reading AX/AY (AXS1 .. AXS3) : 0 - 1023
AnalogZyklen	Number of cycles for skipping reading analog values internally (typical : 5)
Code	Notation which Code Key is to be used for interpreting the IR transmitter inputs
ComNr	Number of the COM-Port for an interface connection (Ports).
Counter	Value of an Counter (int)
Direction	Revolving direction of a motor (Dir) dirOff = 0, dirOff = 1, dirLeft = 1, dirRight = 2
ifTyp	Type of the connected Interface (IFTypen)
InputNr	Number of an I-Input (Nr) inpI1 = 1 ... inpI32 = 32
InputStatus	Actual state of all I-Inputs, one bit for each Input, beginning with 0 (0 = I1, 1 = I2 .... 31 = I32)
KeyNr	Number of the key wanted from the IR transmitter
LampNr	Number of a O-Output - 'half' M-Output (int) values 1 – 8(32)
ModeStatus	Actual running mode of all M-Outputs.2 bit for each Output. Beginning with 0 for M1, value 00 = normal, 01 = RobMode
MotorNr	Number of a M-Output (Nr) outM1 = 1 ... outM16 = 16
MotorStatus	Actual state of all M-Outputs. 2 bit for an Output. Beginning with 0 for M1(00 = dirOff, 01 = dirLeft, 10 = dirRight).
mSek	Time in milliseconds
NrOfChanges	Number of impulses (int)
OnOff	Switching Off/On an Output dirOff = false, ftiOn = true
Position	Position given in number of impulses from the end switch (int)
SerialNr	Active serial number of an ROBO Interface
Speed	Speed to run an M-Output (Speed) dirOff = 0, 1 – 7 (speFull)
SpeedStatus	Actual speeds of all M-Outputs. 4 bit for each M-Output. Values 0000 – 0111(speFull).
TermInputNr	Number of an I-Input to cancel a method. (Nr) inpI1 = 1 ... inpI32 = 32

Value	Common int value
VoltNr	Number of a currency input (Inp)
WaitValue	Return value of WaitForMotors (Wait)

Enums

Used for symbolic name for method parameters.

IFTypen	Names for the available Interfaces to be connected
Port	Name of the Port to be used
Dir	Revolving direction of M-Outputs ...
Inp	Input names
Out	Output names
IRCode	Code keys of the IR transmitter
IRKeys	Special keys of the IR tansmitter
Speed	Speed names
Wait	Return values method WaitForMotors

Alternativly numeric values (e.g. if storing than in tables) or own numeric constants can be used.

Structures

DeviceData

Informations to the active interface

Name	Name of the interface (LPCSTR)
Type	Type of the interface (int)
SerialNr	actual serial number of the interface (int)
Firmware	version of the firmware (LPCSTR)

MessageData

Members of a message

HwId	Broadcast type (always 2)
SubId	Classification (BYTE)
MsgId	Message number (USHORT)
Msg	Message (USHORT)

Constructor

FishFace()
No parameters

Property like Methods

DeviceData	**ActDevice** Informations about the active Interface. A successfull OpenInterface must preceed.
bool	**NotHalt** Cancel request(default = false).
int	**Outputs** Read/write all M-Outputs (MotorStatus)
char*	**Version** (get) Version of the class

get : value only can be read, but not changed.

Methods

ClearCounter

Clear (0) an input counter.

ft.**ClearCounter**(InputNr)

See also : ClearCounters, GetCounter, SetCounter

ClearCounters

Clear (0) all Counters

ft.**ClearCounters**()

See also : ClearCounter, GetCounter, SetCounter

ClearRFMessagesIn

Clear the queque with incoming messages

ft.**ClearMessagesIn**()

Exception : KeinOpen, Messages

See also : ClearRFMessagesOut, GetRFMessage, IsRFMessage, SendRFMessage, WaitForRFMessage

ClearMessagesOut

Clear the queque of outgoing messages

ft.**ClearMessagesOut**()

Exception : KeinOpen, Messages

See also : ClearRFMessagesIn, GetRFMessage, IsRFMessage, SendRFMessage, WaitForRFMessage

ClearMotors

Switch off all M-Outputs

ft.**ClearMotors**()

Exception : InterfaceProblem, KeinOpen

See also : SetMotor, SetMotors, SetLamp Outputs

CloseInterface

Close the connection to the interface

ft.**CloseInterface**()

See also : OpenInterface

Finish

Cancel request (NotHalt, Escape, I-Input(optional))

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

Exception : InterfaceProblem, KeinOpen.

See also : GetInput, GetInputs, Inputs

Example :

```
do {
   cout << "running" << endl;
  ft.Pause(2345);
} while (!ft.Finish(inpI1));
```

The do .. while loop will run as long until ended with `ft.NotHalt(true);` ESC key is pressed or I1 becomes true. The loop will run once in each case.

Alternative :

```
while (ft.Finish(inpI1) == false) {
   cout << "running" << endl;
  ft.Pause(2345);
}
cout << "--- FINIS ---" << endl;
```

This loop will be skipped if an cancel request comes with starting the loop. An while(!ft.Finish(inpI1) is possible too.

Overload for IRKeys :

Cancel request (NotHalt, Escape, IRKey)

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

Exception : InterfaceProblem, KeinOpen

Example :

```
while (!ft.Finish(ircAll, irkM3R) {
....
}
```

The while loop is execute as long as ft.NotHalt comes true, the ESC Key is pressed or Key irkM3R on IR transmitter is pressed.

GetAnalog

Reading an internal analog value (AX / AY, no access to the interface).

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

Exception : InterfaceProblem, KeinOpen

See also : GetVoltage

Example :

```
cout << " AX : " << ft.GetAnalog(anaAX) << endl;
```

GetCounter

Read the value auf counter InputNr

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

See also : SetCounter, ClearCounter, ClearCounters

Example

```
cout << "Counter für I2 : " << ft.GetCounter(inpI2) endl;
```

The actual counter value corresponding to I2 is displayed.

GetInput

Read the value of InputNr.

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

Exception : InterfaceProblem, KeinOpen.

See also : GetInputs, Inputs, Finish, WaitForInput

Example :

```
if (ft.GetInput(inpI1)) {
  ...
}
else {
  ...
}
```

If I-Input I1 (switch, phototransistor, reedrelais) is true, the first block is executed. With `!ft.GetInput(inpI1)` the else path will be executed. Possible too is `if (ft.GetInput(inpI1) == false)` { ... } or `if(!ft.GetInput(inpI1)) {...}`

GetInputs

Read all I-Inputs

IntputStatus = ft.**GetInputs**()

Exception : InterfaceProblem, KeinOpen.

See also : GetInputs, Finish, WaitForInputs

Example :

```
int E13;
  E13 = ft.GetInputs();
  if ((E13 & (0x1 + 0x4)) > 0) cout << "TRUE" << endl;
```

cout is executed if the expression is true (I1 and I3 must be true)
Alternative :

```
  if ((E13 & 0x1) > 0 || (E13 & 0x4) > 0) cout << "TRUE" << endl;
```

GetIRKey

Read the value of IRKeys of the IR transmitter. The IRCode can be considered too.

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

Exception : InterfaceProblem, KeinOpen

See also : GetInputs, GetInput, Finish

GetRFMessage

Read the topmost message from the in queque

MessageData = ft.**GetRFMessage**()

Exception : KeinOpen, Message

See also : ClearRFMessagesIn, ClearRFMessagesOut, IsRfMessage, SendRFMessage, WaitForRFMessage

GetVoltage

Read the voltage value of the noted curency input

value = ft.**GetVoltage**(VoltNr)

Exception : InterfaceProblem, KeinOpen

See also : GetAnalog

Example :

```
cout << ft.GetVoltage(volA1) << endl;
```

The actual voltage value is printed

IsRFMessage

Ar there one or more messages in the in queque.

bool = ft.**IsRFMessage**()

Exception : KeinOpen

See also : ClearRFMessagesIn, ClearRFMessagesOut, GetRFMessage, SernRFMessage, WaitForRFMessage

OpenInterface

Setting up a connection to the interface. Must be the first method used after instancing. There are overloads for ROBO Interface on USB and Interfaces on the COM-Port :

Overload USB :

ft.**OpenInterface**(ifTyp, SerialNr)

If a RF Datalink is connected to USB, the first (or only) found ROBO Interface with radio card will be assigned to it.
ifTyp : ftROBO_IF_USB, ftROBO_RF_Datalink, ftROBO_IO_Extension. ftROBO_first_USB is used, if the first (or only) found Interface on USB is to be connected. It should be done, if only one Interface on USB is used. The SerialNr than is 0.

Exception : InterfaceProblem

See also : CloseInterface

Example :

```
try {
  ft.OpenInterface(ftROBO_first_USB, 0);
......
}
```

```
catch(FishFaceException eft) {
  cout << eft.Text() << endl;
}
ft.CloseInterface();
```

Connection to the interface first found at USB. In case of Error the text '`InterfaceProblem.Open`' is displayed on console.

Overload RF Datalink :

ft.**OpenInterface**(SerialNrInterface)

Setting up a connection to an ROBO Interface (on power only, with radio card) via a ROBO RF Datalink connected to USB. Both must have the same channel number (e.g. RF 2/5 – RF 2/0). The number of the subchannel can be choosen free. Makes only a sense, if there are some more ROBO Interfaces (on power only, with radio card). In this case can be determined which of them should be operated via RF Datalink.

Example :

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

Used are a RF Datalink with RF 2/0 and a ROBO Interface with RF 2/x and the active SerialNr. 7. The Interface with SerialNr 7 is operated by the PC program via RF Datalink.

Overload COM :

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

- iFTyp : ftIntelligent_IF, ftIntelligent_IF_Slave, ftROBO_IIM, ftROBO_COM
- ComNr : Number of the COM-Port the interface is connected to.

Exception : InterfaceProblem

See also : CloseInterface

Example :

```
try {
  ft.OpenInterface(ftIntelligent_IF, portCOM1, 5);
....
}
catch ...
```

Connection to an Intelligent Interface on COM1, each 5 cycles the A-Inputs are updated.

Pause

Stop the program execution for mSek millisecs

ft.**Pause**(mSek)

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForTime

Example :

```
ft.SetMotor(outM1, dirLeft);
ft.Pause(1000);
ft.SetMotor(outM1, dirOff);
```

Motor on M-Output M1 is running for 1 sec.

SendRFMessage

Sending of a broadcast message

ft.**SenRFMessage**(MessageDate, Spez)

- MessageData : structure with the message
- Spez : specification for the sending manner. 0 = to be sended in each case, 1 = sended only, if not the same as the last message in the queque, 2 = sended only, if not contained in the queque at all.

Exception : KeinOpen, Message

See also : ClearRFMessagesIn, ClearRFMessagesOut, GetRFMessage, IsRFMessage, WaitForRFMessage

SetCounter

Set the counter for the noted E-Input.

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

See also : GetCounter, ClearCounter, ClearCounters

SetLamp

Set a 'half' M-Output. To connect a lamp or a magnet ... to a contact of a M-Output and ground.

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

Exception : InterfaceProblem, KeinOpen

See also : SetMotors, SetMotors, ClearMotors

Example :

```
const int lGreen = 1, lYellow = 2, lRed = 3;

  ft.SetLamp(lGreen, ftiOn);
  ft.Pause(2000);
  ft.SetLamp(lGreen, dirOff);
  ft.SetLamp(lYellow, ftiOn);
```

The green lamp on M1 (in front) and ground is switch on for 2 secs and than the yellow on M1 back.

SetMotor

Set one M-Output (motor).

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

Exception : InterfaceProblem, KeinOpen;

See also : SetMotors, ClearMotors, SetLamp, Outputs.

Example 1

```
ft.SetMotor(outM1, dirRight, speFull);
ft.Pause(1000);
ft.SetMotor(outM1, dirLeft, speHalf);
ft.Pause(1000);
ft.SetMotor(outM1, dirOff);
```

Motor on M1 is switched on for 1 sec, right revolving, full speed and than for 1 sec left revolving, half speed.

Example 2

```
ft.SetMotor(outM1, dirLeft, 12, 123);
```

Motor on M-Output M1 runs for 123 impulses (counted on E2) with speed 12. Motor is stopped, if E1 is true before reaching the 123 impulses. The program doesn't wait for ready. See also example WaitForMotors.

SetMotors

Set the state of all M-Outputs. SpeedStatus default = 15 (speFull), ModeStatus = 0 (normal).

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

Exception : InterfaceProblem, KeinOpen;

See also : ClearMotors, SetMotors, SetLamp, Outputs

Example

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

The M-Output M1 is set to dirLeft and that of M4 to dirRight. All other M-Outputs are dirOff. After 1 sec running, all M-Outputs are stopped.

WaitForChange

Wait for NrOfChanges impulses on InputNr or TermInputNr = true

The counter of InputNr is used for counting impulses.

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

Exception : InterfaceProblem, KeinOpen; Can be canceled.

See also : WaitForPositionDown, WaitForPositionUp, WaitForInput, WaitForLow, WaitForHigh.

Example

```
ft.SetMotor(outM1, dirLeft);
ft.WaitForChange(inpI2, 123, inpI1);
ft.SetMotor(outM1, dirOff);
```

M-Output M1 is started with dirLeft and runs for 123 impulses on E2 or E1 = true, M1 is then turned off.

WaitForHigh

Wait for a false/true changing for InputNr

ft.**WaitForHigh**(InputNr)

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForLow, WaitForChange, WaitForInput.

Example

```
ft.SetMotor(outM1, ftiOn);
ft.SetMotor(outM2, dirLeft);
ft.WaitForHigh(inpI1);
ft.SetMotor(outM2, dirOff);
```

A light barrier with lamp on M1 and phototransistor on E-Input E1 is switched on. A Feeder with Motor on M2 is started, than waiting until a parcel on the feeder has leaved the light barrier (light barrier will be true (closed)). After this feeder motor is switched off. The light barrier must be false when starting this sequence.

WaitForInput

Wait for InputNr becomes OnOff. (default = true).

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

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForChange, WaitForLow, WaitForHigh.

Example :

```
ft.SetMotor(outM1, dirLeft);
ft.WaitForInput(inpI1);
ft.SetMotor(outM1, dirOff);
```

Motor on M-Output M1 is started, than waiting for E-Input E1 becomes true. Motor is than switched off : Runnig to an end position.

WaitForLow

Wait for a true/false changing for InputNr

ft.**WaitForLow**(InputNr)

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForChange, WaitForInput, WaitForHigh.

Example :

```
ft.SetMotor(outM1, ftiOn);
ft.SetMotor(outM2, dirLeft);
ft.WaitForLow(inpI1);
ft.SetMotor(outM2, dirOff);
```

A light barrier with lamp on M1 and phototransistor on E-Input E1 is switched on. A Feeder with Motor on M2 is started, than waiting until a parcel on the feeder will enter the light barrier (light barrier will be false (opened)). After this feeder motor is switched off. The light barrier must be true when starting this sequence.

WaitForRFMessage

Waiting for an incoming message

Wait = ft.**WaitForRFMessage**(Time, MessageData)

Exception : KeinOpen, Message; Can be canceled.

- Time (millisecs) : to be waited for a message.
  Time = 0 : Waiting until a message comes in.
- Wait (return code) :
  waiEnd : a message was received
  waiTime : waiting time has ended
  waiESC : ESC key has been pressed
  waiNotHalt : canceled by NotHalt condition
- Message : structure with the incoming Message

See also : ClearRFMessagesIn, ClearRFMessagesOut, GetRFMesssage, IsRFMessage, SendRFMessage.

Example :

```
do {
  .. do something ..
} while(ft.WaitForRFMessage(300, inMessage) == waiTime);
... process the incoming message if waiEnd elsewhere cancel
```

At the end of do loop wait for 300 millisecs for an incoming message, if there is none, the loop repeats. But the loop can be canceled by ESC key or NotHalt, not processed in that example.

WaitForMotors

Wait for a MotorReady event or for timeout of Time.

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

Time (int) : time in millisecs. Time = 0 endless waiting : all motors of the list are off.

MotorNr(Nr) : List of M-Outputs in any order to be waiting for stop.

WaitWert(Wait) : reason why the method has ended
ftiEnde : all requested M-Outputs are dirOff
ftiTime : the requested waiting time is off
ftiNotHalt : NotHalt = true, all requested motors are stopped.
ftiESC : ESC key was pressed, all requested motors are stopped.

Exception : InterfaceProblem, KeinOpen; Can be canceled.

See also : SetMotor

Example :

```
ft.SetMotor(outM4, dirLeft, Speed.Half, 50);
ft.SetMotor(outM3, dirRight, Speed.Full, 40);
do {
   cout << ft.GetCounter(inpI6) << " - " <<
               ft.GetCounter(inpI8) << endl;
} while (ft.WaitForMotors(300, 4, 3) == ftiTime);
cout << ft.GetCounter(inpI6) << " - " <<
             ft.GetCounter(inpI8) << endl;
```

Motor on M-Output M4 starts with half speed, left for 50 impulses, that on M3 with full speed, right for 40 impulses. The do while loop waits for reaching the position (ft.WaitForMotors). Every 0.3 secs the actual postion is display within the loop (300 ... ftiTiime). If position is reached (<> ftiTime), the job is done, motors have stopped already. The final position is displayed.
Notice : The loop can be broken by NotHalt or ESC key, that is not controlled.

WaitForPositionDown

Wait for reaching the (destination)Position, by decrement the (actual)Counter:

ActPosition = ft.**WaitForPositionDown**(InputNr, Counter, Position, Optional TermInputNr)

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForPositionUp, WaitForChange

Example :

```
int Zaehler = 12;
  ft.SetMotor(outM1, dirLeft);
  ft.WaitForPositionDown(inpI2, ref Zaehler, 0, inpI1);
  ft.SetMotor(outM1, dirOff);
  cout << "Counter : " << Zaehler << endl;
```

The actual position is 12 (Zaehler), motor on M1 is started left. WaitForPositonDown is waiting for reaching position 0, motor is stopped than. Same is done, if E1 becomes true.

WaitForPositionUp

Wait for reaching the (destination)Position, by increment the (actual)Counter.

ActPosition = ft.**WaitForPositionUp**(InputNr, Counter, Position, Optional TermInputNr)

Exception : InterfaceProblem, KeinOpen; Can be canceled

See also : WaitForPositionDown, WaitForChange

Example :

```
int Zaehler = 0;
  ft.SetMotor(outM1, dirRight);
  ft.WaitForPositionUp(inpI2, Zaehler, 24);
  ft.SetMotor(outM1, dirOff);
  cout << "Counter : " << Zaehler << endl;
```

The actual position is 0 (Zaehler), motor on M1 is started right. WaitForPositionUp ist waiting for reaching position 24, motor is stopped than.

WaitForTime

Stop the program execution for mSek millisecs

ft.**WaitForTime**(mSek)

Same as Pause

Exception : InterfaceProblem, KeinOpen; Can be canceled.

See also : Pause

Example

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

Loop do ... while switches first M-Output (lamp) M1 on and all others off (00 01), wait for 555 millisecs, M2 (lamp) switched on (all others off, 01 00) and waiting for 555 millisecs. Result is a alternating blinker. Loop ends with pressing the ESC key.

Notes

Notes to the Counters

An essential element of determining the position are the counters. There is a counter for each E-input (attention : E1 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 are part of the control block and can read from it. In cs style there are special functions. The counter are used internally be some functions (e.g. SetMotor with Counter parameter and most of the Wait methods. umFish30.DLL uses the counters only with um/csRobMotor(s)).

Notes to the Speed Control

The speed control is based on a cyclic switch on/off of the M-outputs (PWM). For that reason internally there is a list of switch commands. The speed is determined by the parameter Speed (SetMotor) and SpeedStatus (SetMotors). The speed control is located in a separate thread of umFish30.DLL which controls the motors in this manner until they are switched off by SetMotor(s).

Notes to the Rob Functions

The Rob functions 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 that counter is switched off. On the last 6 impulses they will operate with half speed to have a more exact positioning. Sometime it may happen one more impulse is counted. It can determined by read the counter. Values > 0 signal a plus position. They actual position should be corrected.

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

Motor	End Switch	Impulse Switch
1	1	2
2	3	4
3	5	6
4	7	8
5	9	10
6	11	12
7	13	14
8	15	16

The motors turn "left". That means they run to the end switch if operated in direction dirLeft. Motors are switched off, if they are reaching the end switch before the counter is zero.

The motors can be operated with umRobMotor/csRobMotor/SetMotor (a single motor). The parameter ICount/Counter noted the way to go in items of impulses (a true/false or false/true signal on the appropriate impulse switch). The impulse switches are decreased during polling. They can be accessed via ftiDCB.Counter or the function csGetCounter. Note Counter set by the application will be changed in this turn.

The motors can be operated all together with one function : umRobMotors / csSetMotors / SetMotors. Therefore the parameters must be prepared in the following manner :

MotorStatus : each motor 2bit, starting with M1 : bit 0 and 1.
00 : off, 01 left, 10 right.
SpeedStatus : each motor 4bit, starting with M1 : bit 0-3,
0000 off, 1000 half speed, 11111 full.
ModeStatus : each motor 4 bit, starting with M1 : bit 0-3,
0000 Normal-Mode, 0001 Rob-Mode, others free for further use.
(may be stepper motors).

Example : csRobMotors(ft, 0x9, 0xF6, 0x11);
0x means Hexa, binary : 1001 | 11110110 | 10001 -> M2 = right, speed 15, Rob-Mode, M1 = lelft, speed 6 RobMode. Other motors are off.

Before operating the motors, the counters are to be set for each involved motor.

Direction = 0 or the appropriate bit value in MotorStatus overrides the speed parameter, motor is stopped.

The motors are running simultaneously (up to 8 motors). They can be switched one after the other by umRobMotor/csRobMotor. They will started with the next polling cycle and run asynchronous (that means independent of the actions of the application) until they have reached the mentioned positon. Than they are switched off during the normal polling.

To observe, the motors reaching their position and to synchronized the application a WaitForMotors can be used. The FishFace classes own such a method. umFish30.DLL offers none.

Notes to Radio Controlled Operations

Elements of the radio controlled operations are the ROBO RF Datalink an one ore more ROBO Interfaces with radio card.

There are three different types of radio operating :

1. Route Through : One Interface with radio card is connected via RF Datalink to the PC. The application runs on the PC without knowing the kind of connection. Useful if a model equipped with ROBO Interface and radio card can move free in a room. Control an user interface is done by a PC program.
   Fully supported by CFishFace.
2. Autonomous : Some ROBO Interfaces equipped with radio card communicate via radio control. The RF Datalink is in the role of a message router. The applications run on the controller of the interfaces.
   Not supported by CFishFace.
3. Route Through and Message Router : The first Interface is controlled by an PC program, all the others run autonomous. But they can be contacted via radio control by the first. This kind of radio control is supported by CFishFace on the PC side. Programming of the other interfaces is done by Renesas C or ROBO Pro.

(3) is supported from CFishFace (and umFish40.DLL) by some special things :
- SendRFMessage, GetRFMessage, IsRFMessage, WaitForRFMessage, ClearRFMessagesIn, ClearRFMessagesOut.

- structure MessageData for the incoming and outgoing data.