Eclipse 3.4, Java 6, umFish40.DLL & javaFish40.DLL

ftComputing für Java

Java für Vergnügungssüchtige, VISTA sei Dank
C/C++ zu spießig, C#/VB.NET zu monopolistisch

Ulrich Müller

Inhaltsverzeichnis

Das System

Allgemeines

Mit dem Package ftcomputing.robo.jar steht eine Möglichkeit zur Verfügung, die aktuellen fischertechnik Interfaces recht komfortabel zu programmieren. Ausgeliefert wird ein Original Workspace - Workspace34 - von Eclipse 3.4 (ohne .metadata), der neben dem oben genannten Package noch die erforderliche umFish40.DLL und die Wrapper.DLL javaFish40.DLL sowie diese Dokumentation enthält. Hinzu kommen noch eine Reihe von Beispielprojekten.

ftcomputing.robo.jar

Enthält als zentrale Klassen die Basis-Klasse JavaFish und die darauf aufsetzende Klasse FishFace. Hinzu kommen enum-Klassen (enum Natur ging nicht, da beliebige num. Werte als Ergebnis der Aufzählung benötigt wurden) für die verwendeten Parameter.

Die Klasse **JavaFish** kapselt zusammen mit der JNI-Wrapper.DLL javaFish40.DLL die Funktionen der zentralen umFish40.DLL und reicht sie weitgehend 1:1 an die Anwendung weiter. JavaFish ist primär Basis für FishFace oder eigene Kreationen.

Achtung : JNI erwartet hier zwingend den Klassennamen JavaFish. Bei eigenen Erweiterungen also lieber eine neue Klasse ableiten anstatt schnell mal ein bischen zu ändern.

Die Klasse **FishFace** enthält neben den verbesserten Methoden von JavaFish (Unterbrechbarkeit, Abbrechbarkeit) noch eine Reihe weiterer Methoden (meist Wait…), die im Programm ein stilvolles Warten auf den Abschluß der durch z.B. setMotor angestoßenen Operationen ermöglichen.

Die Beispiele sind sehr einfach gehalten, sie setzen nur ein Interface mit ein paar angebauten Teilen (Lampen, Motoren, Taster, Sensoren) voraus, aber kein bestimmtes Modell.

Install

JDK 1.6 : Install mit Setup-Programm, soweit nicht bereits geschehen. Ein älteres JDK wird’s sicher auch tun.

Eclipse 3.4 : Entpacken ZIP, Desktopsymbol einrichten. Hier wird auch eine (Versionsnummer kleiner gehen).

umFish40.DLL und **javaFish40.DLL** nach Windows\System32 kopieren

Eclipse : Workspace anlegen

Im Workspace Verzeichnis **ftComputing** anlegen, dort ftcomputing.robo.jar (mit JavaFish) ablegen. Beispielprogramme kopieren.

Eclipse : Projekt anlegen

Eclipse in Java Perspektive mit Standard Einstellungen.
Arbeiten im Package Explorer : Kontext-Menu über RechtsClick

1. **New** | JavaProject
   Project Name : NeuesProjekt | Finish
2. **NeuesProjekt** | New | Package | Name : neuesProjekt
3. **Package** neuesProjekt aufklappen | New | Class
   Name : MainFish, x Checkbox : public static void main …
4. **NeuesProjekt** | Build Path | Add External Archives
   Verzeichnis ftComputing im Workspace suchen und ftcomputing.robo.jar wählen
5. In class MainFish (im Editor) hinter package neuesProjekt
   import ftcomputing.robo.*; einfügen

Weiter im Editor mit Programmdetails für reine Console- bzw. einfache Swing-Programme

Eclipse : jar erstellen und nutzen

Eclipse in Java Perspektive mit Standard Einstellungen.
Arbeiten im Package Explorer : Kontext-Menu über RechtsClick

Erstellen einer Bibliothek, die in einem Projekt genutzt werden kann :

1. **Projekt** im Package Explorer markieren und aufklappen
2. RechtsClick auf Projekt | Export | **JAR** File
   Next : Gewünschtes Package (oder ganzes Projekt) markieren
   Bei einem Package sind die .files nicht erforderlich
   Zielpfad eingeben | Finish

Erstellen einer selbständig ablaufenden Anwendung (geht so erst mit Eclpse 3.4) :

1. RechtsClick auf Projekt | Export | Runnable JAR File
2. Next | Launch Configuration | Projekt wählen
3. Export destination (Name und Pfad des JAR Files) festlegen | Finish

Achtung : System.out.println Ausgabe werden nicht angezeigt, also eher Swing-Projekte einsetzen.

JavaFish

JavaFish Methoden

Die Klasse JavaFish ist Bestandteil des Package ftcomputing.robo.jar. Für die Parameter können entweder num. Werte, symbolische Konstanten oder die Member der enum-Klassen Dir, Inp, Out, Direction … eingesetzt werden.

Die Funktionen ist public native, eine Instanzierung ist erforderlich.

Mit iHandle wird ein Zeiger in den umFish40-Datenbereich bezeichnet. So sind mehrere Interfaces separat ansprechbar. Die Rückgabewerte der Funktionen enthalten neben einem "Nutzwert" immer auch einen möglichen jrError-Code über den der Verlauf der Operation abgefragt werden kann. mit iState sind Wahrheitswerte gemeint : 0 = false, jrError = Fehler, alles andere gleich true.

int iHandle jrOpenInterfaceUSB(int ifTyp, int SerialNr);

int iHandle **jrOpenInterfaceUSBdis**(int ifTyp, int SerialNr, int DistanceMode);

int iHandle jrOpenInterfaceCOM(int ifTyp, int ComNr, int AnalogZyklen);

Herstellen einer Verbindung zum Interface. ifTyp siehe enum Klasse IFTypen
Rückgabewert ist ein (Instanz-Handle, das von den weiteren Methoden genutzt wird)

int iError jrCloseInterface(int iHandle);

Beenden der Verbindung zum Interface

int iState jrGetInput(int iHandle, int InputNr);

Status des angegebenen Digitaleinganges (!= 0 ist true)

int iStates jrGetInputs(int iHandle);

Status aller angeschlossenen Digitaleingänge (auch der ggf. angeschlossenen Extensions)

int iValue jrGetAnalog(int iHandle, int AnalogNr);

Auslesen des angegebenen Analogeinganges

int iState jrGetIRKey(int iHandle, int Code, int KeyNr);

Status des angegebenen Keys auf dem IR Sender

int iValue jrGetVoltage(int iHandle, int VoltNr);

Auslesen des angegebenen Spannungseinganges

int iError jrSetMotor(int iHandle, int MotNr, int Dir);
int iError jrSetMotorEx(int iHandle, int MotNr, int Dir, int Speed);

Setzen eines M-Ausganges (default : Full-Speed)

int iError jrGetMotors(int iHandle);

Auslesen aller M-Ausgänge

int iError jrSetMotors(int iHandle, int MotorStatus);
int iError jrSetMotorsEx(int iHandle, int MotorStatus,
int SpeedStatus, int SpeedStatus16);

Setzen aller M-Ausgänge einschl. Geschwindigkeit (default : full)

int iError jrGetModeStatus(int iHandle, int MotNr);

Auslesen des Status aller M-Ausgänge (normal / RobModus)

int iError jrSetModeStatus(int iHandle, int MotNr, int Mode);

Setzen des Status aller M-Ausgänge

int iError jrSetLamp(int iHandle, int LampNr, int OnOff);
int iError jrSetLampEx(int iHandle, int LampNr, int OnOff, int Power);

Setzen eines O-Ausganges (default : Full-Power)

int iError jrRobMotor(int iHandle, int MotNr, int Dir, int Speed, int ICount);

Fahren eines RobMotors (Motor mit Impulsrad auf der Abtriebswelle) um eine vorgegebene Anzahl von Impulse. Das Fahren geschieht asynchron

int iError jrRobMotors(int iHandle, int MotorStatus,
int SpeedStatus, int SpeedStatus16, int ModeStatus);

Betreiben aller an die M-Ausgänge angeschlossenen Geräte einschl. des RobStatus. Evtl. benötigte Counter müssen separat gesetzt werden.

int iValue jrGetCounter(int iHandle, int CounterNr);

Auslesen des angegebenen Zählers

int iError jrSetCounter(int iHandle, int CounterNr, int ICount);

Setzen des angegebenen Zählers

int iError jrClearCounters(int iHandle);

Löschen aller Zähler

int iValue jrGetDistanceValue(int iHandle, int SensorNr);

Auslesen der vom UltraSonic-Sensor gemessenen Entfernung in cm. jrOpenInterfaceUSB mit Distance.UltraSonic muß vorhergegangen sein.

int iValue jrGetActDeviceType(int iHandle);
int iValue jrGetActDeviceSerialNr(int iHandle);
int oValue jrGetActDeviceFirmwareNr(int iHandle);

Auslesen der Werte des aktuellen Interfaces. Die Firmware Version steht byteweise in der FirmwareNr. Siehe auch FishFace.

final int *jrError* = 0xE0000001;

Allgemeiner Fehlercode bei Rückkehr aus einer Methode. Wird ggf. anstelle eines gültigen Wertes geliefert.

int iState escape();

Win32-Funktion ESC-Taste betätigt.

int iValue getTickCount();

Win32-Funktion Auslesen TickCounts in MilliSekunden seit Mitternacht.

void sleep(int MilliSec);

Win32-Funktion Anhalten des aktuellen Threads um die angegebene Zeit in MilliSekunden

Console Programme mit JavaFish

Generell gilt hier, daß die Klasse JavaFish eher Basis für eigene, darauf aufsetzende Klassen ist, als eine, die im normalen Betrieb eingesetzt werden soll. Dafür ist die Klasse FishFace vorgesehen.

```
// --- CFish : Minimale JavaFish Console-Anwendung ---------
//             Enthält JavaFish-Source

package cFish;
import ftcomputing.robo.*;

public class MainFish {
  public static void main(String[] args) {
    JavaFish ft = new JavaFish();
    int iHandle = ft.jrOpenInterfaceUSB(0, 0);
    if(iHandle == JavaFish.jrError) {
      System.out.println("OpenInterfaceProblem : ENDE");
      return;
    }
    System.out.println("Action : I1 drücken");
    while(ft.jrGetInput(iHandle,1)==0) Thread.yield();
    System.out.println("Ende   : ESC-Taste oder I1");
    int Runde = 0;
    do {
       try {
         System.out.println("Runde  : " + ++Runde);
         ft.jrSetLamp(iHandle, 1, 1);
         Thread.sleep(333);
         ft.jrSetLamp(iHandle, 2, 1);
         Thread.sleep(333);
         ft.jrSetLamp(iHandle, 3, 1);
         Thread.sleep(333);
         ft.jrSetLamp(iHandle, 4, 1);
         Thread.sleep(666);
         ft.jrSetMotors(iHandle, 0);
         Thread.sleep(333);
       } catch(InterruptedException e){}
     } while((ft.jrGetInput(iHandle, 1) == 0) &&
             (JavaFish.escape() == 0));
     ft.jrCloseInterface(iHandle);
     System.out.println("--- FINITO ---");
   }
}
```

`package cFish;`
Package Name sollte vergeben werden, sonst gilt default

`import ftcomputing.robo.*;`
Verweis auf die Klasse JavaFish mit den Zugriffsfunktionen zum Interface

`JavaFish ft = new JavaFish();`
Instanzieren.

`ft.jrOpenInterfaceUSB(0, 0);`
Herstellen einer Verbindung zum ersten ROBO Interface an USB, Abfrage auf Fehler

Das eigentliche (Nutz)Programm

`ft.jrCloseInterface(iHandle);`
Schließen der Verbindung zum Interface.

Console mit Methode

Instanzierung um Methoden der Klasse nutzen zu können. Bei größeren Programmen sinnvoll, alternativ : Erstellen weiterer Klassen in einer eigenen Source.

```
// --- DFish : JavaFish Console-Anwendung ----------------------
//     Nutzung ftcomputing.robo.jar, Instanzierung, Methoden
package dFish;
import ftcomputing.robo.JavaFish;

public class MainFish {
    JavaFish ft = new JavaFish();
    int iHandle;

    public static void main(String[] args) {
     MainFish mf = new MainFish();
     System.out.println("DFish  : gestartet");
       mf.Action();
       System.out.println("DFish  : beendet");
   }
    private void Action(){
       iHandle = ft.jrOpenInterfaceUSB(0, 0);
       System.out.println("Action : I1 drücken");
       while(ft.jrGetInput(iHandle,1)==0) Thread.yield();
       System.out.println("Ende   : ESC-Taste oder I1");
       int Runde = 0;
       do {
           System.out.println("Runde  : " + ++Runde);
           Blinken(333);
       } while((ft.jrGetInput(iHandle, 1) == 0) &&
               (JavaFish.escape() == 0));
       ft.jrCloseInterface(iHandle);
    }
    private void Blinken(int Dauer) {
     try {
         ft.jrSetLamp(iHandle, 1, 1);
         Thread.sleep(Dauer);
……
       ft.jrSetMotors(iHandle, 0);
       Thread.sleep(Dauer);
     } catch(InterruptedException e){}
    }

}
```

Globale Variable `ft, iHandle`

`MainFish mf = new MainFish();`
Instanzieren der Klasse

`mf.Action();`
Aufruf der Haupt-Methode.

`private void Action() {…}`
Ablaufsteuerung

`private void Blinken(int Dauer) {…}`
Steuern des Blinkens

Swing Programme mit JavaFish

Java Swing : GUI zu Fuß.

```
// --- GFish : Einfache Swing-Anwendung ----------------
// unter Nutzung von ftcomputing.robo.jar (Referenzed Libraries)
package gFish;
import java.awt.*;
import javax.swing.*;
import ftcomputing.robo.*;

public class MainFish {
   JFrame frmMain    = new JFrame("Hello Ampel");
   JLabel lblStatus  =
                 new JLabel("Kontrolle Interface", JLabel.CENTER);
   JLabel lblHinweis = new JLabel(" ", JLabel.CENTER);
   JavaFish ft       = new JavaFish();
   int iHandle;

   public MainFish() {
      frmMain.setLayout(new GridLayout(0, 1));
      frmMain.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      frmMain.setSize(300, 200);
      lblStatus.setForeground(Color.BLUE);
      frmMain.add(lblStatus);
      frmMain.add(lblHinweis);
      frmMain.setVisible(true);
   }

   public static void main(String[] args) {
       MainFish mf = new MainFish();
       mf.Action();
   }

   private void Action() {
       iHandle = ft.jrOpenInterfaceUSB(0, 0);
       if(iHandle != JavaFish.jrError) {
         lblStatus.setText("Action : I1 drücken");
         while(ft.jrGetInput(iHandle,1)==0) Thread.yield();
         lblHinweis.setText("Ende : ESC-Taste oder I1");
         int Runde = 0;
         do {
           lblStatus.setText("Runde : " + ++Runde);
           Blinken(333);
```

```
        } while((ft.jrGetInput(iHandle, 1) == 0) &&
                (JavaFish.escape() == 0));
        ft.jrSetMotors(iHandle, 0);
        ft.jrCloseInterface(iHandle);
      }
      else lblStatus.setText("OpenInterface.Fehler");
      lblHinweis.setText("FINITO, Ende : X");
  }

  private void Blinken(int Dauer) {
    try {
          ft.jrSetLamp(iHandle, 1, 1);
          Thread.sleep(Dauer);
…
    } catch(InterruptedException e){}
  }
}
```

Die Form ist vom Typ JFrame, deren Controls werden in einem (hier einzeiligen) Gitter der Reihe nach abgelegt (Zeilenangabe 0 : beliebig).

Zusätzlich werden die JLabel lblStatus und lblHinweis angelegt in die Ausgaben über den Programmverlauf gemacht werden.

FishFace

Allgemeines

Parameter für die Methoden sind int-Werte der **enum-Klassen** Dir, Distance, IFTypen, Inp, IrCode, IRKeys, Mot, Out, Speed, Wait.

Mit Ausnahme von Version, Open/CloseInterface verlangen alle Methoden eine intakte Verbindung zum Interface. Andernfalls wird die allgemeine **Exception** : FishFaceException ausgelöst.

Einige (langlaufende) Methoden sind **unterbrechbar** (Thread.yield) um ein Update der Oberfläche zu ermöglichen und **abbrechbar** ESC-Taste, NotHalt um in Crash-Situationen das Programm schnell beenden zu können.

Methoden

static String **Version**()

Auslesen der FishFace Version

DeviceData **getActDevice**()

Auslesen der Firmware Version des aktuellen Interfaces

int **getAnalog**(AnalogNr)

Auslesen eines Analogwertes von Dir.AX, AY (AXS1, AXS2, AXS3)

int **getCounter**(InputNr)

Auslesen des Zählers für den angegebenen Digital-Eingang Inp.I1…

setCounter(InputNr, ICount)

Setzen des Zählers für den angegebenen Digital-Eingang (Inp.I1---)

int **getDistance**(SensorNr)

Auslesen der vom UltraSonic-Sensor gemessenen Entfernung in cm. OpenInterface mit Distance.UltraSonic muß vorhergegangen sein. Inp.D1 / Inp.D2

int **getDistanceMode**()

Feststellen des DistanceModes (enum Distance)

boolean **getInput**(int InputNr)

Feststellen des Status des angegebenen Digitaleinganges Inp.I1…

int **getInputs**()

Auslesen des Status aller Digital-Eingänge

boolean **getIRKey**(IRCode, IRKey)

Feststellen des Status des angegebenen Keys des IR-Senders.

setLamp(LampNr, OnOff)

Setzen eines O-Ausganges (Out.O1… Dir.On / Dir.Off)

setMotor(MotorNr, Direction)
setMotor(MotorNr, Direction, Speed)
setMotor(MotorNr, Direction, Speed, ICount)

Setzen eines M-Ausganges (Mot.M1…, Dir.Left.., Speed.Full). Bei angeschlossenem ImpulsCounter auch Setzen der zu drehenden Impulse.

setMotors(MotorStatus)

Setzen des Status (Dir.Off, Dir.Left, Dir.Right) aller M-Ausgänge

boolean **getNotHalt**()

setNotHalt(OnOff)

Abfrage NotHalt Status

int **getOutputs**()

Auslesen des Status aller M-Ausgänge

int **getVoltage**(VoltNr)

Auslesen des Spannungswertes vom Eingang Inp.A1 / Inp.A2 / Inp.AV / Inp.AZ

clearCounter(InputNr)

Löschen des zu Inp.I1 … gehörenden Counters

clearCounters()

Löschen aller Counter

clearMotors()

Löschen aller M-Ausgänge (damit werden auch alle O-Ausgänge gelöscht)

closeInterface()

Schließen der Verbindung zum Interface

boolean **finish**()
boolean **finish**(InputNr)

Abfrage auf Ende-Wunsch (durch I-Eingang true, ESC-Taste, NotHalt)

openInterface(ifTyp, SerialNr)
ifTyp : 0 = erster USB (SerialNr = 0), 60 = ROBO Interface, 90 = I/O-Extension,
110 = RF-Datalink, 200 ROBO ConnectBox
openInterface(ifTyp, SerialNr, DistanceMode)

Herstellen einer Verbindung zum Interface, unterstützt werden z.Zt. nur die USB Interfaces.
Erster USB heißt : erstes ROBO-Gerät, dass an USB gefunden wird.
SerialNr ist hier immer 0, sonst muß sie angegeben werden.

pause(mSek)

Anhalten des aktuellen Threads für die angegebenen MilliSekunden. Das Anhalten wird durch ESC-Taste bzw. NotHalt abgebrochen. Die Methode ist unterbrechbar.

waitForChange(InputNr, NrOfChanges)
waitForChange(InputNr, NrOfChanges, TermInputNr)

Warten auf die angegebene Anzahl von Zustandsveränderungen am Digital-Eingang. Abbrechbar, unterbrechbar.

waitForHigh(InputNr)

Warten auf einen Wechsel von Low auf High am angegebenen Digital-Eingang.
Abbrechbar, unterbrechbar.

waitForInput(InputNr, OnOff)

Warten auf Zustand true bzw. false am angegebene Digital-Eingang.
Unterbrechbar, abbrechbar.

waitForLow(InputNr)

Warten auf einen Wechsel von High auf Low am angegebenen Digital-Eingang.
Abbrechbar, unterbrechbar.

Wait **waitForMotors**(mSek, MotorNrs…)

Warten auf ein Motor-Ready-Ereignis der angegebenen M-Ausgänge Mot.M1... Die M-Ausgänge müssen als RobMotoren mit setMotor(Mot, Dir, Speed, ICount) einzeln gestartet werden. Das Ready-Ereignis tritt ein, wenn alle zugehörenden Counter auf 0 stehen.
Mit mSek wird angegeben, wielange auf das Ready-Ereignis gewartet werden soll.
0 Endlos, sonst die angegebene Anzahl von MilliSekunden. Die Methode kann in einer Schleife ohne erneuten Motorstart wiederholt werden. Der Return-Wert gibt an wie die Methode beendet wurde : Ready-Ereignis, TimeOut, ESC-Taste, NotHalt.

FishFace Beispiel AFace (Console Programm)

```
import ftcomputing.robo.*;

public class MainFace {
   FishFace ft = new FishFace();

   public static void main(String[] args) {
      MainFace mf = new MainFace();
      System.out.println("AFace gestartet");
      System.out.println("FishFace-Version : " + FishFace.Version());
      try {
       mf.Action(333);
      }
      catch(FishFaceException eft) {System.out.println(eft);}
      finally {System.out.println("Finito");}
   }
   private void Action(int Dauer) {
     ft.openInterface(0, 0, Distance.UltraSonic);
     FishFace.DeviceData dd = ft.getActDevice();
     System.out.println("Device     : " + ft.getActDevice().Name);
     System.out.println("DeviceType : " + dd.Type);
     System.out.println("SerialNr   : " + dd.SerialNr);
     System.out.println("Firmware   : " + dd.Firmware);
     Explore();
     int Runde = 0;
     do {
       System.out.println("Runde     : " + ++Runde);
       System.out.println("Distance : " + ft.getDistance(Inp.D1));
         ft.setLamp(1, Dir.On);
.........
         ft.pause(Dauer * 2);
         ft.setMotors(0);
         ft.pause(Dauer);
       } while(FishFace.escape() == 0);
      ft.closeInterface();
      System.out.println("--- FINITO ---");
   }

   private void Explore(){
      System.out.println("Explorer vor");
      ft.setMotor(Mot.M3, Dir.Left, Speed.Full, 12);
      ft.setMotor(Mot.M4, Dir.Right, Speed.Half, 34);
      ft.waitForMotors(0, Mot.M3, Mot.M4);
      System.out.println("Helligkeit : " + ft.getAnalog(Inp.AX));
   }
```

Das eigentliche Nutzprogramm Action wird durch try / catch gekapselt, eventuelle Fehler werden auf der Console angezeigt.

In Action wird mit openInterface zunächst eine Verbindung zu einem ROBO-Inteface mit UltraSonic-Sensor hergestellt (der Sensor selber muß nicht unbedingt vorhanden sein, er macht aber den meisten Spaß) Danach werden die Interface-Daten angezeigt.

In der Routine Explore werden dann die RobMotoren M3 und M4 mit Impulstastern an I5, I7 ein wenig gescheucht. Auf die 12 Impulse von M3 und die 34 von M4 wird endlos gewartet.

Ist das überstanden, wird in Action heftig geblinkt : Lampen an O1 - O4. Zusätzlich wird mit einer Photozelle noch die Raumhelligkeit angezeigt.Das geschieht in einer Endlosschleife, die nur durch die ESC-Taste wieder abgebrochen werden kann.

Eine adäquate **Swing-Lösung** liegt mit dem Projekt **GFace** vor.