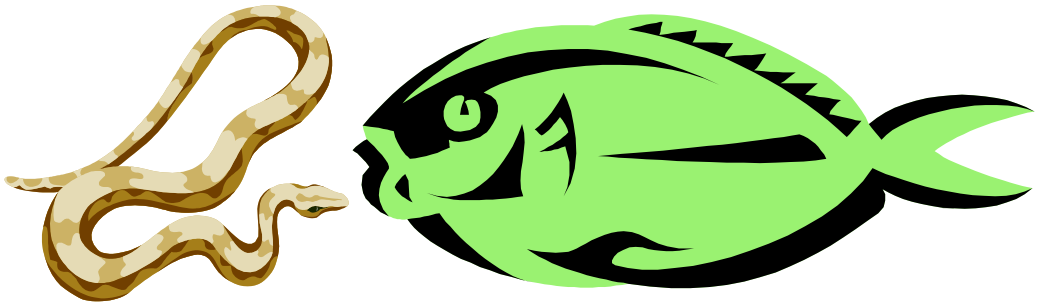
ftComputing

FishFa40 with Python

umFish40.DLL and Python 3.1.1

Ulrich Müller



Inhaltsverzeichnis

Ulrich Müller, D-33100 Paderborn, Lange Wenne 18. Fon 05251/56873
eMail : um@ftcomputing.de
HomePage : www.ftComputing.de

Dokument : FishFa40Py311e.Doc, Druckdatum : 03.04.2011
Titelbild : Einfügen | Grafik | AusDatei | Office | Fish5.WMF

Overview

Common

The class FishFace based on umFish40.DLL offers the possibility for programming ROBO / Intelligent Interfaces using Python. FishFace anables the "Online-Programming" - ROBO Interface connected via USB or RF Datalink to the PC.

FishFace contains methods for switching of the M-Outputs (O-Outputs as well) and for getting the actual values of the Interface inputs. Therefore the Interface is polled each 10 msec. In addition the state changes (on / off) of the inputs are counted. They are used in addition for switching the M-Outputs. Therefore is used a fix assignment of M-Output and I-Input (end / impulse switch driven by a motor). The M/O-Outputs can be operated with different "speed" (power). Therefore they are switched in short intervals (PWM).

The A-Inputs (analogous inputs) offer raw values in the range 0 - 1023.

The Interfaces can be operated by control of the Python module FishFa40.py based on umFish40.DLL. Used is a connection with windll.umFish40… a very simple solution. FishFa40.py is available as source, therefore it can be changed for own needs. IDLE is recommended as an most common editor. Notice : FishFace methods are not interruptable (but can be canceld). For interruptable issues thread programming must be done (or Tk.update() must be inserted). Beside the main class FishFace FishFa40.py owns classes for programming of Robots (FishRobot) with RobMotors (fix combination of motor, end - and impulse switch) and a class for programming step motors (FishStep).

This document describes the use of FishFa40.py with the classes FishFace, FishFaceException, FishRobot and FishStep.

Installation of Python 3.1.1 and FishFa40

Python- 3.1.1
contains th Phyton system with documentation. The IDE IDLE is included.
www.python.org/download

PythonFish311.ZIP
contains FishFa40.py, examples and umFish40.DLL. umFish40.DLL should be copied in a central directory (e.g. \Windows\System32). FishFa40.py / pyc should be copied to the directory \Programs\Pyton31\Lib.
www.ftcomputing.de/zip/pythonfish311.zip

Hint

Supported are the following ROBO Interfaces : a single ROBO Interface (connected to USB or COM). ROBO Interface with up to 3 Extensions, ROBO Interface connected via RF Datalink (on USB), the I/O Extension connected direct to USB and the ROBO Connect Box. In addition the Intelligent Interface (up with one extension).

FishFa40.py contains the central class FishFace and the inherited classes FishRobot and FishStep.

Reference

Imports and Instances

FishFa40.PY

Using FishFa40.py the following import is to be used :

```
from FishFa40 import *
```

FishFa40.py must be positioned in an accessible path (see Installation).

Instancing is be done with :

```
ft = FishFace()
```

or when using FishRobot :

```
ft = FishRobot(motors list)
```

The motors list contains the motor number (no of the M-Output) and the max. number of impulses can be driven (see also : Notes to the Rob functions). e.g. (2, 120), (3, 96), (1, 180).

or when using FishStep.:

```
ft = FishStep(motors list)
```

The motors list contains the motor number (no of th M-Output) and the max number of stepper cycles can be done (12 cycles are one motor revolving). See also : Notes to the Stepper functions. Exsample : ft = FishStep((1, 680), (3, 500)).

Instead of ft each other correct name can be used.

Error Handling

It is recommended to handle possible FishFace errors with an try ... except :

```
try:
  ft.OpenInterface()
except FishFaceException:
  print ("------- Interface Problem ------")
  sys.exit(0)
```

sys requieres an additional `import sys`.

FishFa40.PY

Common

bool	0 = false, 1 = true
int	32bit integer signed (Python default)

Used variables names

All the variables are integer (32bit value)

ActPosition	actual position in impulses counted from 0 (positive values)
AnalogNr	number of the analog input (0 – 1)
Counter	impulse counter (the true/false change is counted automatically).
Direction	revolving direction of a motor (Ein/On, Aus/Off, Links/Left, Rechts/Right)
InputNr	number of an I-Input (1 – 8(32))
LampNr	number of an O-Output (1-8(32))
ModeStatus	state of the operating mode of all M-Outputs four bits for each Output beginning with bits 0-3 for M1 (0000 normal, 0001 RobMode)
MotorNr, MotNr	number of a M-Output (1-4(16))
MotorNrs	list of motor numbers to be controlled
MotorStatus	state of all M-Outputs.2 bits for each. beginning with 0-1 for M1 (00 = Off, 01 = Left, 10 = Right).
mSek	timespan in millisecs
NrOfChange	number of impulses (changes of an I-Input state).
OnOff	logical value (1 = true, 0 = false)
PortName	name of the port the Intelligent Interface is connected to. "COM1", ... "COM4"
PosHook	name of a CallBack function for handling of the act. position
Speed	speed to operate an M-Output (0, off – 7, full)
SpeedStatus	speed state of all M-Outputs. 4bit for each. start with bits 0-3 for M1. values 0000-0111.
TargetPosition	target position counted in impulses from 0.
TermInputNr	number of an I-Input to terminate an operation out of normal end.
Wert	some 32bit integer value

Symbolic Constants

Fehler	common FishFace Error
Ein	switch On an M-Output
Aus	switch Off an M-Output
Links	switch On an M-Output left turning
Rechts	switch On an M-Output right turning
Ende	WaitForMotors ends correctly
Time	WaitForMotors the timespan has finished
ESC	WaitForMotors aborted bei ESC key

FishFace Methods

- **ClearCounter**(InputNr)
  clear(0) of the named counter

- **ClearCounters**()
  clear(0) of all counters

- **ClearMotors**()
  switch off of all M-Outputs

- **CloseInterface**()
  close a connection to the Interface

bool **Finish**(InputNr)
control if there is an end request (ESC key, I-Input (optional))
Example :

```
while not ft.Finish():
  ft.SetMotor(1, ft.Ein)
  ft.Pause(555)
  ft.SetMotor(1, ft.Aus)
  ft.Pause(333)
```

The while loop (a blinking lamp) runs until ESC key is pressed

int **GetAnalog**(AnalogNr)
read the addressed analog value

int **GetCounter**(InputNr)
read the value of the addressed counter.
Example

```
print "Tower Position : ", ft.GetCounter(2)
```

bool **GetInput**(InputNr)
read the value of the addressed I-Input
Example

```
if ft.GetInput(1):
  .....
else:
  ....
```

if I-Input is true (!= 0), the "then" lines are operated

int **GetInputs**()
read values of all I-Inputs

Example

```
e = ft.GetInputs()
if e & 0x1 or e & 0x40: print "TRUE"
```

if I1 or I7 are true, "TRUE" is printed

int **GetOutputs**()
read the value of all M-Outputs

- **OpenInterface**(TypNr, Nr)
  making of an connection to an Interface. OpenInterface must be executed as first method.
  TypNr :

```
ftROBO_first_USB       =   0
ftIntelligent_IF       =  10
ftIntelligent_IF_Slave =  20
ftROBO_IF_IIM          =  50
ftROBO_IF_USB          =  60
ftROBO_IF_COM          =  70
ftROBO_IF_Over_RF      =  80
ftROBO_Extension       =  90
ftROBO_RF_Datalink     = 110
   ROBO Connect Box    = 200
```

  Nr : If connected by USB the serial number of the Interface or 0, if the first Interface should be taken. If connected via COM : COM number.

```
ft.OpenInterface()
```

  Open the first (or only) ROBO Interface at USB

```
ft.OpenInterface(ft.ftROBO_IF_USB, 1)
```

  Open ROBO Interface with serial number 1 at USB

```
ft.OpenInterface(ft.ftIntelligent_IF, 1)
```

  Open Intelligent Interface atn COM1

- **Pause**(mSek)
  stop the program execution for millisecs
  Example

```
ft.SetMotor(1, ft.Links)
ft.Pause(1000)
ft.SetMotor(1, ft.Aus)
```

  motor on M-Output M1 is switched On for 1 second.

- **SetCounter**(InputNr, Wert)
  set the addressed counter to the noted value.

- **SetLamp**(LampNr, OnOff)
  set the addressed O-Output to the noted value
  Example:

```
ft.SetLamp(1, ft.Ein)
ft.Pause(2000)
ft.SetLamp(1, ft.Aus)
ft.SetLamp(2, ft.Ein)
```

  lamp on O1 (outside pin and ground) is switched on for 2 seconds, after this the lamp on O2 …

- **SetOutputs**(MotorStatus)
  set a new state of all M-Outputs

int **GetVoltage**(VoltNr)
  read the actual currency value on the addressed Input (A1, A2, AV, AZ (1 – 4))

- **SetMotor**(MotorNr, Direction, Speed, Counter)
  set of a M-Output, optional with speed value (default = 7, full) and number auf impulses to be run, the motor stops, if reached the count.
  Example 1 :

```
ft.SetMotor(1, ft.Rechts, 7)
ft.Pause(1000)
ft.SetMotor(1, ft.Links, 4)
ft.Pause(1000)
ft.SetMotor(1, ft.Aus)
```

  motor at M1 runs right for 1000 milliseconds, full speed and than 1000 milliseconds half speed.
  Example 2 :

```
ft.SetMotor(1, ft.Links, 5, 123)
```

  motor at M1 is switched on for 123 impulses at I2 or I1 = true, speed value 5. Motor is switched off if the impulses are reached or I1 becomes true. The program does not stop in meanwhile.

- **SetMotors**(MotorStatus, SpeedStatus, ModeStatus)
  set state of all M-Outputs, optional with speed value and ModeStatus (default = 0). With RobMode (1) the counters belonging to the motors are to be set before SetMotors(SetCounter(m)), The motors stop, if count is reached.
  Example :

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

  motorer at M1 is switched left and the motor at M4 to right, M2 and M3 are off. After 1 second all M-Outputs are switched off.

- **WaitForChange**(InputNr, NrOfChanges, TermInputNr)
  Wwait for for NrOfChanges changes at InputNr or TermInputNr (optional) becomes true
  Example :

```
ft.SetMotor(1, ft.Links)
ft.WaitForChange(2, 123, 1)
ft.SetMotor(1, ft.Aus)
```

  motor at M1 is switched on left, than wait for 123 Impulses at or I1 = true. Than motor is switched off.

- **WaitForHigh**(InputNr)
  waiting for an false/true state at InputNr
  Example :

```
ft.SetMotor(1, ft.Ein)
ft.SetMotor(2, ft.Links)
ft.WaitForHigh(1)
ft.SetMotor(2, ft.Aus)
```

  A light barrier with lamp at M1 and phototransistor at I1 is switched on. An transportation belt with motor M2 is started and than waiting for a piece on the belt which interrupt the light barrier to come free (light barrier is than true). In this case motor is switched off.

- **WaitForInput**(InputNr, OnOff)
  waiting for addressed Input comes to OnOff (true or false). OnOff is optional (default=1)
  Example :

```
ft.SetMotor(1, ft.Links)
ft.WaitForInput(1)
ft.SetMotor(1, ft.Aus)
```

  motor at M1 is started, waiting for I1 goes to true. In this case M1 is switched off. Can be used for positioning to an end position.

- **WaitForLow**(InputNr)
  waiting for addressed Input for a true/false change
  Example :

```
ft.SetMotor(1, ft.Ein)
ft.SetMotor(2, ft.Links)
ft.WaitForLow(1)
ft.SetMotor(2, ft.Aus)
```

  A light barrier with lamp at M1 and phototransistor at I1 is switched on. A transportation belt with motor M2 is started. Waiting for light barrier comes false (must be true with start). Than motor M2 is set to off.

wait **WaitForMotors**(mSek, MotorNrs)
waiting for MotorReady Event or the end of a timespan (msec) started by SetMotor(…Count) or ModeStatus (0001....) and is true with Counter = 0 for all listed motors.
mSek = 0 : unlimited wait.
MotorNrs = list of the motors
wait = Time : ended by time
wait = Ende : all motors stopped
wait = ESC : canceled by ESC key
Example :

```
ft.SetMotor(4, ft.Links, 4, 50)
ft.SetMotor(3, ft.Rechts, 7, 40)
while ft.WaitForMotors(500, [4, 3]) == ft.Time :
  print ft.GetCounter(6), " - ", ft.GetCounter(8)
```

motor M4 left, half speed, 50 impulses and motor M3 right, full speed, 40 impulses are started. The while loop is waiting for success. Every 500 millisecs the actual position is printed.

FishRobot-Methods

FishRobot Methods use 'RobMotors'. See 'Notes to the Rob Functions'.

- **MotCntl**
  list of the supported motors and max position (maxPos) for each motor (beginning with 0, the end switch position) and the actual position (actPos)
  Example :

```
[[3,222,0],[4,88,0]]
```

  Robot with two motors at M3 and M4 with maxPos = 222 and 88, actPos = 0

- **MoveHome**()
  Move to home position (at the end switches) with all listet motors.
  Example

```
ft = FishRobot([[3,222],[4,88]])
ft.MoveHome()
```

  The motors listed with the instancing (M3, M4) move left turning to the end switches I6 and I8 (see also Notes to the Rob Functions).

- **MoveTo**(PosHook, PosList)
  simultanous move to the positions listed in PosList. The CallBack function PosHook is called in even intervals.
  Example

```
ft = FishRobot([[3,222],[4,88]])
ft.MoveHome()
ft.MoveTo(ft.PosPrint, [23,34])
```

  The robot owns two motors at M3 and M4. First home position is taken (position [0, 0]), after that position [23, 34] is taken. Position values are interpreted in line with the instancing (M3 to 23 and M4 to 34). If there is no CallBack to be used it must be noted as None.
  Example :

```
ft.MoveTo(PosPrint, [23,34])
```

  move to postions 23 and 34 while printing the actual position in intervals by the PosPrint function.

```
def PosPrint(PosList):
  for p in PosList: print p
```

Tips & Trics

Program Frame

The program clips to be found in chapter Techniques need a small program frame to run within it. This frame is needed for all examples of 'Techniques'.

Import / Instancing

The program clips use the module FishFa40.py which must be placed in a Python path. e.g. C:\Programs\Python\Lib.

```
from FishFa40 import *
ft = FishFace()
```

Open/Close

Before the Interface can be used there must be a connection to it :

```
ft.OpenInterface()
```

In this case first ROBO Interface on USB is accessed.

Before ending the program it must be released :

```
ft.CloseInterface
```

To control possible open errors OpenInterface must be placed in a try / except construct :

```
# ----- Programmname.py : short description ---------------

from FishFa40 import *
import sys
ft = FishFace()

try:
  ft.OpenInterface()
except FishFaceError:
  print ("--- Interface Problem ---")
  sys.exit(0)

# --- program operations ---


.....


ft.CloseInterface()
```

Techniques

Blinking Loop

Lamp at M1 blinking with 1000 msec frequency :

```
mYellow = 1

while not ft.Finish():
  ft.SetMotor(mYellow, ft.Ein)
  ft.Pause(555)
  ft.SetMotor(mYellow, ft.Aus)
  ft.Pause(333)
```

Used Parameters should be symbolic (mYellow instead of 1). Some Names are predefined : ft.Ein (On), ft.Aus, ft.Links, ft.Rechts und ft.Ende, ft.Time, ft.ESC (WaitForMotors).

Mostly the program contains a large loop for repeating all command. In this case is it `while not ft.Finish()`: the method Finish controls if there is any cancel request (return value 1). To end the loop press the ESC key.

Alternate Blinking

Lamps at M1 and M2 are blinking alternatively.

solution 1 :

```
while not ft.Finish():
  ft.SetMotor(2, ft.Aus)
  ft.SetMotor(1, ft.Ein)
  ft.Pause(444)
  ft.SetMotor(1, ft.Aus)
  ft.SetMotor(2, ft.Ein)
  ft.Pause(444)
```

solution 2 more compact :

```
while not ft.Finish():
  ft.SetMotors(0x1)
  ft.Pause(444)
  ft.SetMotors(0x4)
  ft.Pause(444)
```

In this cause all outputs are switched with one statement. Always 2bit for an output : 00000001 for M1 On and 00000100 for M2 On. All other outputs are set to Off.

State of an I-Input

If I1 is true print "---EIN---" else "---AUS---" :

```
if ft.GetInput(1):
  print ("--- EIN ---")
else:
  print ("--- Aus ---")
```

Waiting for I-Input comes true

If I1 is closed, print "--- Es geht los ---" :

```
print ("--- For program start press I1 ---")
ft.WaitForInput(1)
print ("--- Es geht los ---")
```

Display the state of an I-Input

state of I1 :

```
e1 = ft.GetInput(1)
if e1: print ("I1 : ", e1)
```

permanent display of all I-Inputs :

```
while not ft.Finish():
  print ("Status der I-Eingaenge : ", hex(ft.GetInputs()))
  ft.Pause(1234)
```

Analogous display

permanent display of AX and AY :

```
while not ft.Finish():
  print ("Values AX : ", ft.GetAnalog(1), " AY : ", \
        ft.GetAnalog(2))
  ft.Pause(1111)
```

Run motor for a timespan

motor at M3 runs 3,5 secs to left :

```
ft.SetMotor(3, ft.Links)
ft.Pause(3500)
ft.SetMotor(3, ft.Aus)
```

Run to the end switch

motor at M3 run until end switch I5 and than is turned off :

```
ft.SetMotor(3, ft.Links)
while not ft.GetInput(5): pass
ft.SetMotor(3, ft.Aus)
```

The example is a little to complicated and can't be canceled with ESC key. Better solution :

```
ft.SetMotor(3, ft.Links)
ft.WaitForInput(5)
ft.SetMotor(3, ft.Aus)
```

Run for a specified number of impulses

WaitForChange

motor at M3 with impulse switch I6

```
ft.SetMotor(3, ft.Links)
ft.WaitForChange(6, 12)
ft.SetMotor(3, ft.Aus)
```

WaitForMotors

motor at M3 runs 12 impulses on I6 with half speed, left turn.

```
ft.SetMotor(3, ft.Links, 4, 12)
ft.WaitForMotors(0, [3,])
```

Es wird gewartet, bis das Ziel erreicht wurde. Es geht auch ohne WaitForMotors, wenn das Programm anderweitig beschäftigt ist (Die Motoren schalten bei Erreichen der Zielposition selbsttätig ab). Siehe auch Anmerkungen : Rob-Funktionen.

Two motors simultanously with posistion display

Two motors(M3, M4 with impulse counter I6 / I8) are running (see also Rob Functions). The actual position is displayed :

```
ft.SetMotor(3, ft.Links, 7, 121)
ft.SetMotor(4, ft.Rechts, 4, 64)
while ft.WaitForMotors(300, (3,4)) == ft.Time:
  print ("Position M3 - M4 : ", ft.GetCounter(6), " - ", \
         ft.GetCounter(8))
print ("Position M3 - M4 : ", ft.GetCounter(6), " - ", \
       ft.GetCounter(8), " --- Final ---")
```

Motor M3 runs full speed for 121 impulses to left.
Motor M4 runs half speed for 64 impulses to right
WaitForMotors is waiting for both are ready. Each 300 millisecs the actual position is diplayed. If finish, the final position is displayed.

Lamps

Lamps mostly are operated in the same manner as motors (connected to the two pins of an M-Output.e.g. `ft.SetMotor(1, ft.Ein))`, because of they only can be switched on or off, an connection to ground and one O-Output (half M-Output) is possible to. In this case up to 8 lamps can be connected :

```
ft.SetLamp(1, ft.Ein)
ft.SetLamp(4, ft.Ein)
ft.Pause(1000)
ft.SetLamp(1, ft.Aus)
ft.SetLamp(4, ft.Aus)
```

the lamps at O1 and O4 are switched on for 1 sec.

Light Barriers

Waiting for Interception

Lamp at M1, Phototransistor at I1.

```
mLicht, ePhoto, false = 1,1,0

ft.SetMotor(mLicht, ft.Ein)
ft.Pause(555)
ft.WaitForInput(ePhoto, false)
```

Lamp is switched on, 500 millisecs wait for warming up, then waiting for phototransistor = false

Waiting for entering a light barrier

Lamp at M1, conveyor belt at M3, phototransistor at I1 :

```
mBand, ePhoto = 3,1

ft.SetMotor(mBand, ft.Links)
ft.WaitForLow(ePhoto)
ft.SetMotor(mBand, ft.Aus)
```

The motor mBand runs until a piece on the belt will interrupt the light barrier (the lamp for the light barrier is switched on before).

Waiting for leaving a light barrier

Lamp a1 M1, conveyor belt at M3, phototransistor at I1 :

```
mBand, ePhoto = 3,1

ft.SetMotor(mBand, ft.Links)
ft.WaitForHigh(ePhoto)
ft.SetMotor(mBand, ft.Aus)
```

The motor mBand runs until a piece on the belt which interrupts the barrier leaves the light barrier.

Swichting all M-Outputs

SetMotors is used to switch all M-Outputs at the same time. Therefore the the bits of the parameter MotorStatus must be set in right manner. MotorStatus contains 2bits for each M-Output : 00 00 00 00 (with Extension Modules additional bits). 00 means off, 01 left, 10 right.
00 01 00 00 M3 left (or on) and 01 00 00 00 M4 left.

Simple traffic lights

A simple traffic light cycle may be this : Green - Yellow - Red - RedYellow
The Lamps M1 : Green, M2 : Yellow, M3 : Red and the constants for that
mGreen 00 00 00 01, mYellow = 00 00 01 00, mRed = 00 01 00 00 (decimal 1, 4, 16)

```
mGreen, mYellow, mRed = 1,4,16
while not 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)
```

Traffic lights controlled by a list

Wenn man einen festen Ampeltakt vorgibt, kann man den Ablauf auch listengesteuert machen :
If there is a fix cycle the lights can be controlled by a list

```
mGreen, mYellow, mRed = 1,4,16
Phase = [mGreen, mGreen, mGreen, mGreen, mYellow,
         mRed,   mRed,   mRed,   mRed,   mRed+mYellow]

while not ft.Finish():
  for p in Phase:
    ft.SetMotors(p)
    ft.Pause(250)
```

Hier wird mit einer festen Taktung von 250 MilliSekunden gearbeitet. Das Verfahren lohnt bei komplexeren Steuerungen.
The cycle lasts 250 millisecs. This procedure is useful, if there are more complex controls.

Running lights

If there are connected 8 lamps to the M-Outputs of the Interface ist very easy programming running lights :

```
while not ft.Finish():
  Phase = 3
  while Phase < 256:
    ft.SetMotors(Phase)
    ft.Pause(555)
    Phase = Phase<<1
```

To run a robot

Using the class FishRobot instead off FishFace there are some nice Robot method in addition :

```
from FishFa40 import *
ft = FishRobot([[3,222],[4,88]])

ft.OpenInterface()

ft.Home()

ft.MoveTo(ft.PosPrint, [23,34])
ft.MoveTo(None, [23,34])
ft.MoveTo(ft.PosPrint, [23,23])

ft.CloseInterface()
```

On instancing the robot configuration is described : motors M3 and M4 with max. 222 and 88 impulses.
After OpenInterface run to the home position (turn left to end switches) position = 0.
Than position 23 / 34 is moved to. The actual position is printed via ft.PosPrint.
Again to 23 / 34 controlling there is no new movement.
Than move to 23 / 23 : Only motor M4 runs.

Notes

Notes to the Counters

An essential element of determining the postion are the counters. There is a counter for each I-Input. The counters will notify (and count) each change of the state of an input (e.g. opening or closing a switch).

Notes to the Rob Functions

The Rob functions are running in a special operation mode, the RoboMode. In this mod the involved counters are decreased. Reaching the value 0, the motor belonging to that counter is switched off.

Operating of a motor in RobMode uses a fix concept of wiring the motors: Each motor is associated with an end switch an 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
...	...	...
16	31	32

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

The motors can be operated with SetMotor (a single Motor) or all together witch SetMotors. The parameter counter notes the number of impulses to be run. Methods GetCounter / SetCounter can access directly the involved counter.