



fischertechnik TXT  
community firmware

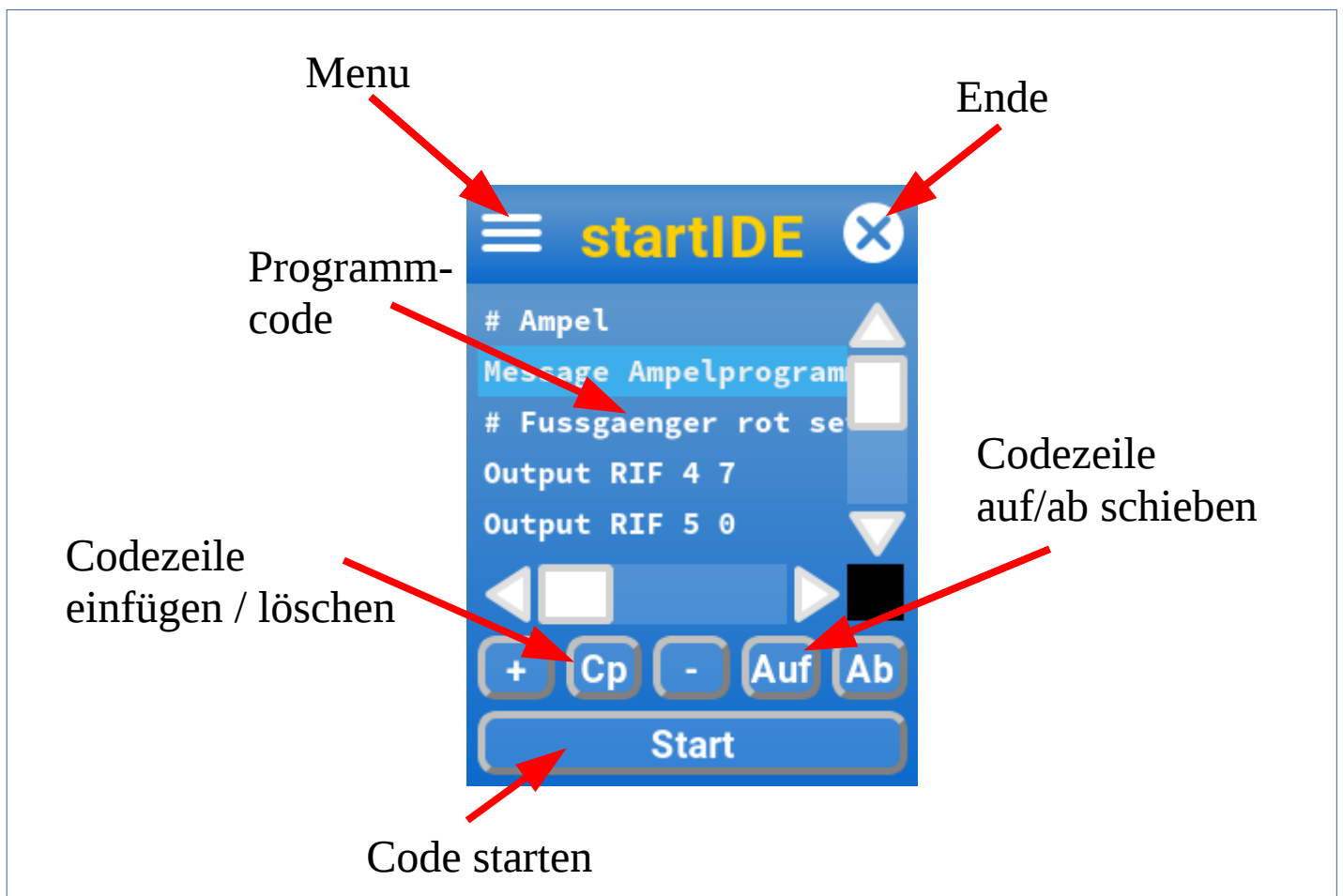


# startIDE

v1.0 prelim

## Schnelleinstieg

Handbuch 2017/12/30



startIDE ist eine Programmier-App für die community firmware des TXT Controllers (auch für den TX-Pi und ftdduino), mit der sich eine Vielzahl einfacher Modelle programmieren lässt.

startIDE © 2017 Peter Habermehl, [peter.habermehl@gmx.de](mailto:peter.habermehl@gmx.de)

Basierend auf der community firmware für den fischertechnik ® TXT Controller: <http://cfw.ftcommunity.de> mit herzlichem Dank an alle Beteiligten.

Besonderer Dank geht an **Till Harbaum** für den TX-Pi <https://github.com/harbaum/tx-pi> und seine Motivationsmaßnahmen. Weiterhin seien erwähnt **Richard Kunze**, **Raphael Jacob** und **Esther Mietzsch** vornehmlich für die Arbeit an der community firmware und **Torsten Stuehn** für ftrobopy <https://github.com/ftrobopy/ftrobopy> zur TXT-Programmierung sowie **Erik Andresen** für die libroboint <https://github.com/nxdefiant/libroboint> zum Zugriff auf die Robo Interfaces und last but not least **Rolf Meingast** als „der Tester“.

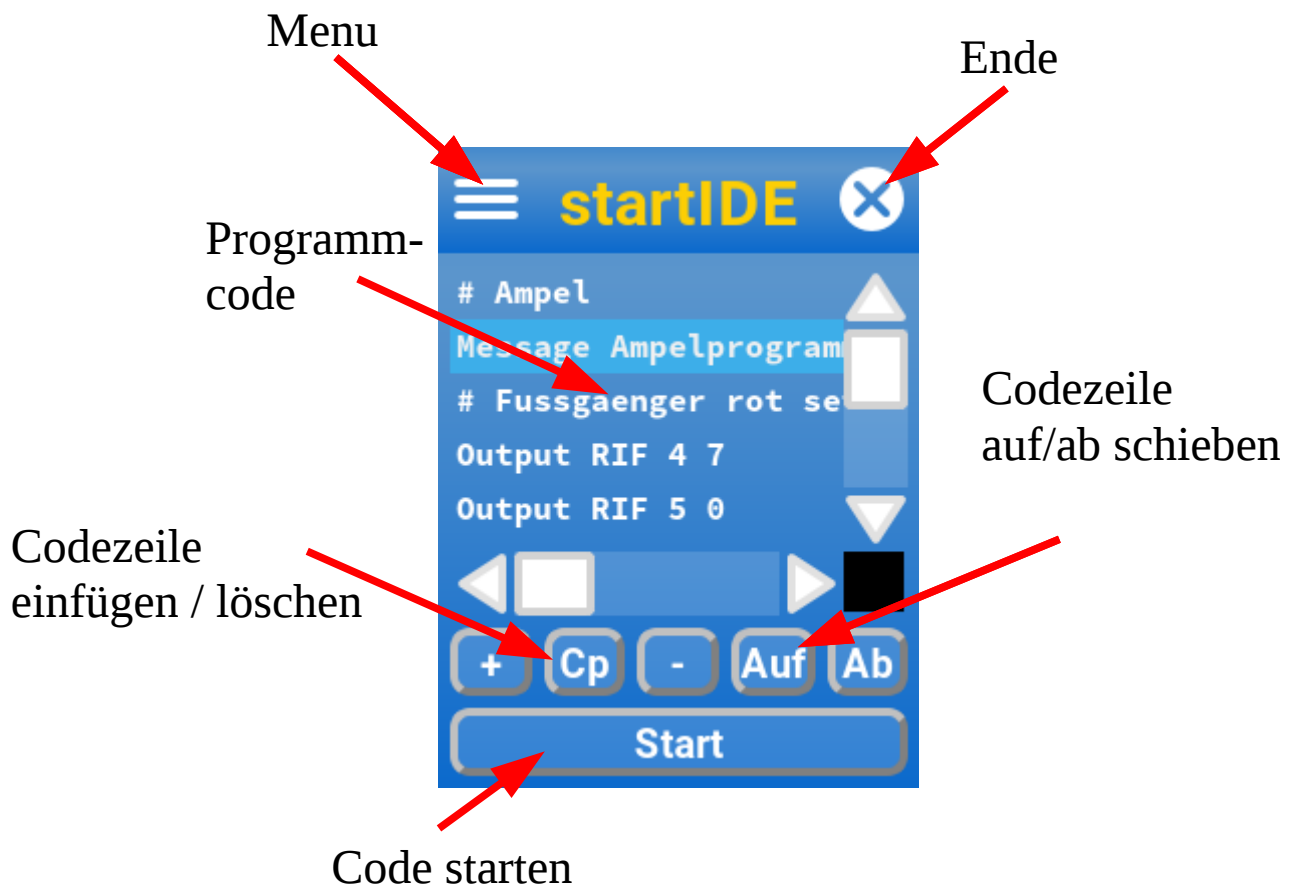
# Inhaltsverzeichnis

1. Der Hauptbildschirm.....	6
2. Das Menu.....	7
3. Funktionsübersicht.....	9
3.1. Eingänge.....	10
3.1.1. WaitForInputDig.....	11
3.1.2. IfInputDig.....	12
3.1.3. WaitForInput.....	13
3.1.4. IfInput.....	14
3.1.5. QueryInput.....	15
3.2. Ausgänge.....	16
3.2.1. Output.....	16
3.2.2. Motor.....	17
3.2.3. MotorPulsewheel.....	18
3.2.4. MotorEncoder.....	19
3.2.5. MotorEncoderSync.....	20
3.3. Variable.....	22
3.3.1. Init.....	23
3.3.2. From.....	24
3.3.2.1. FromIn.....	24
3.3.2.2. FromKeypad.....	25
3.3.2.3. FromDial.....	26
3.3.2.4. FromButtons.....	27
3.3.3. QueryVar.....	28
3.3.4. IfVar.....	28
3.3.5. Calc.....	28
3.4. Steuerung.....	30
3.4.1. # comment.....	30

3.4.2. Tag.....	30
3.4.3. Jump.....	31
3.4.4. LoopTo.....	32
3.4.5. Time.....	34
3.4.5.1. Delay.....	34
3.4.5.2. TimerQuery.....	35
3.4.5.3. TimerClear.....	35
3.4.5.4. IfTimer.....	35
3.4.5.5. Interrupt.....	35
3.4.5.6. QueryNow.....	37
3.4.6. Stop.....	37
3.5. Module.....	38
3.5.1. Call / CallExt.....	39
3.5.2. Return.....	39
3.5.3. Module.....	39
3.5.4. MEnd.....	40
3.6. Interaction.....	41
3.6.1. Print.....	41
3.6.2. Clear.....	42
3.6.3. Message.....	42
3.6.4. Logfile.....	43
4. Beispiele.....	45
4.1. Mitgelieferte startIDE-Projekte.....	45
4.1.1. c_Ampel.....	45
4.1.2. c_Blink.....	46
4.1.3. c_Haendetrockner.....	46
4.1.4. c_Lauflicht und c_Signalf Feuer.....	46
4.2. Programmieraufgaben.....	47
4.1.1. Hallo Welt 1.....	47

4.1.2. Hallo Welt 2.....	47
4.1.3. Hallo Welt 3.....	47
4.1.4. Gottes Rache.....	48
4.1.5. Es regnet immer noch.....	51
5. Tips und Tricks.....	52
5.1. Programmierhinweise.....	52
5.2. Kopieren von Programmteilen.....	53
5.3. Debugging / Fehlersuche.....	53
5.4. startIDE-Projekte auf PC übertragen.....	56
Anhang.....	60
Befehlsreferenz.....	61
Ablaufplan „Turm von Hanoi“.....	62

## 1. Der Hauptbildschirm



Die wesentlichen Elemente des Hauptbildschirms sind in der Abbildung gekennzeichnet.

Oben links befindet sich der „**Menu**“-Knopf, dessen Funktionen in Kap. 2 erläutert werden.

Mit dem Kreuz-Knopf („**Ende**“) wird startIDE beendet. Der aktuelle Programmcode geht nicht verloren und steht beim nächsten Aufruf von startIDE wieder zur Verfügung.

Mit dem **Plus**- und **Minus**-Knopf wird eine neue Programmzeile eingefügt bzw. die aktuelle Zeile gelöscht. Zum Löschen ist ein schnelles zweimaliges Antippen des Minus-Knopfs erforderlich.

Der **Cp**- Knopf dupliziert die aktuelle Befehlszeile (Copy).

Mit dem **Up**- bzw. **Down**-Knopf wird die aktuelle Zeile nach oben bzw. unten verschoben.

Der „**Start**“-Knopf schließlich startet die Ausführung des Programmes.

## 2. Das Menu

Das Menu bietet die Optionen



Unter „**Project**“ ist es möglich,

- mit „**New**“ ein neues Projekt anzulegen. Dabei wird der Programmspeicher gelöscht.
- mit „**Load**“ ein vorher gespeichertes Projekt wieder zu laden
- mit „**Save**“ das aktuelle Projekt abzuspeichern
- mit „**Delete**“ ein abgespeichertes Projekt dauerhaft zu löschen

Der Speicherort für die Projekte ist dabei ein Unterverzeichnis des startIDE-Verzeichnisses auf der SD-Karte. In **Kap. 5.4.** wird erläutert, wie man von außen (PC) auf diese Daten zugreifen kann.

Mit den unter „**Modules**“ angebotenen Optionen können

- mit „**Import**“ Programmmodule von SD-Karte zum aktuellen Code dazugeladen werden
- mit „**Export**“ Programmmodule aus dem aktuellen Programm auf SD-Karte exportiert werden
- mit „**Delete**“ Module von SD-Karte dauerhaft gelöscht werden

Zur Erklärung, was ein Modul im Sinne von startIDE ist, siehe **Kap. 3.**

Der Menüpunkt „**Interfaces**“ öffnet eine Benachrichtigung, die anzeigt, welche Hardware-Interfaces (TXT und/oder Robo Interface Familie sowie ftdduino) aktuell gefunden wurden.

Wurde ein externes Interface nach dem Start von startIDE angeschlossen, so muß dieser Menüpunkt aufgerufen werden, um das Interface in startIDE nutzen zu können.

**startIDE kann neben der TXT-Hardware das Robo Interface, die Robo I/O Extension, das Robo LT Interface und das Robo RF Funkmodul ansprechen. Es wird jedoch nur das unter „Interfaces“ angezeigte Gerät verwendet, auch wenn mehrere Interfaces gleichzeitig angeschlossen sind.**

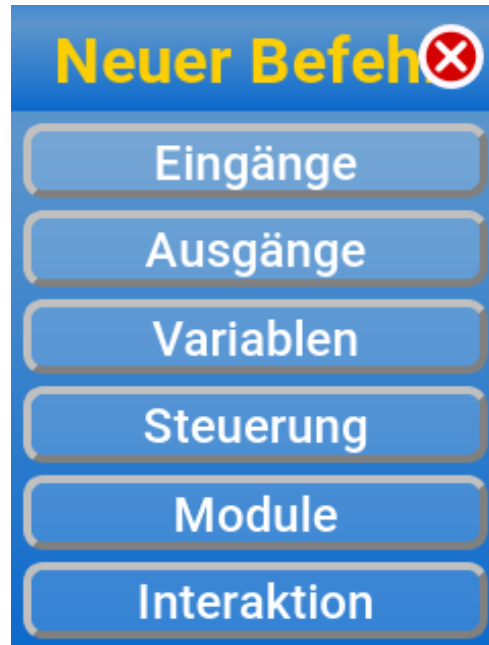
Auf dem TXT ist es jedoch möglich, in einem Programm gleichzeitig die TXT-Hardware **und** ein weiteres, per USB an den TXT angeschlossenes Interface zu nutzen.

**Weiterhin kann auch der ftdduino per USB angeschlossen und als I/O Hardware verwendet werden; dies ist ideal für die Kombination TX-Pi/ftduino.**



### 3. Funktionsübersicht

Über den „+“-Knopf auf dem Hauptbildschirm können Funktionen aus den folgenden Gruppen in den Programmcode eingefügt werden:



Mit dem Kreuz-Knopf oben rechts kann die Auswahl abgebrochen werden, ohne eine neue Code-Zeile einzufügen.

Soweit die Funktionen Parameter benötigen, werden diese jeweils in entsprechenden Bildschirmdialogen konfiguriert.

Dabei können in vielen Fällen, in denen Zahlen als Parameter erwartet werden, auch Variablen verwendet werden, siehe Abs. 3.3.

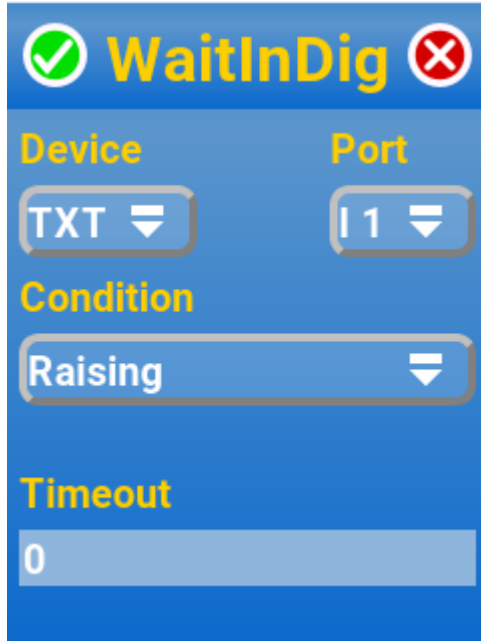
### 3.1. Eingänge



Mit den hier verfügbaren Funktionen können die Hardware-Eingänge abgefragt und der Programmablauf abhängig vom Eingangszustand gesteuert werden.

### 3.1.1. WaitForInputDig

Der Befehl „WaitForInputDig“ wartet auf eine Änderung des Signals an einem Digitaleingang.



Es gibt folgende Optionen:

**Device:** TXT oder RIF  
Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:** Die Nummer des Anschlusses

**Condition:** Soll auf eine steigende („Raising“) oder fallende („Falling“) Signalflanke gewartet werden?

Steigende Flanke bedeutet, daß der Kontakt geschlossen wird, fallend dementsprechend ein Öffnen des Kontaktes.

**Timeout:** maximale Wartezeit in ms, bis auch ohne Signaländerung fortgefahren wird. 0 steht dabei für „unendlich lange“ Wartezeit.

Im Code-Abschnitt des Hauptbildschirms ist die Syntax

```
WaitInDig <Device> <Port> <Condition> <Timeout>
```

also z.B.

```
WaitInDig RIF 1 Raising 500
```

Wartet max. 500ms darauf, daß der Kontakt an RoboInterface I1 geschlossen wird. Nach Schließen des Kontakts oder 500ms wird mit dem Programmablauf fortgefahren.

### 3.1.2. IfInDig

Der Befehl „IfInDig“ überprüft den Zustand eines Digitaleingangs und springt mit der Programmausführung abhängig vom Ergebnis ggf. zu einer Sprungmarke.

Vor dem Einfügen des IfInDig-Befehls muß mindestens eine Sprungmarke definiert sein.

Zur Definition von Sprungmarken siehe **Abs. 3.4.2., Befehl „Tag“**

**Device:** TXT oder RIF  
Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:** Die Nummer des Anschlusses

**Condition:** „True“ (Wahr, Eingang ist aktiv / Kontakt geschlossen) oder „False“ (Falsch, Eingang ist deaktiv / Kontakt offen)

**Jump Tag:** Die Sprungmarke, die angesprungen werden soll

Im Codeabschnitt des Hauptbildschirms ist die Syntas

```
IfInDig <Device> <Port> <Condition> <Jump Tag>
```

also z.B.

```
IfInDig TXT 2 False ende
```

Springt zur Marke „ende“, wenn I2 am TXT Falsch (also Kontakt geöffnet) ist. Andernfalls wird mit der direkt folgenden Programmzeile fortgefahren.

### 3.1.3. WaitForInput

Analog dem WaitForInputDig-Befehl wartet WaitForInput auf das Eintreten eines Zustands an einem beliebigen (Analog-)Eingang.



**Device:** TXT, FTD oder RIF  
**Port:** Je nach gewählter Eingangs-Art  
**Eing.-Art:** Typ des Einganges (s.u.)  
**Operator:** Vergleichsoperator  
**Wert:** Vergleichswert  
**Timeout:** maximale Wartezeit in ms, bis auch Erreichen des Sollzustands fortgefahren wird. 0 steht dabei für „unendlich lange“ Wartezeit.

Mögliche Eingangstypen sind:

- switch S (Digitaleingang)
- resistance R (Widerstand)
- voltage V (Spannung)
- distance D (Ultraschall-Abstandssensor)

Vergleichsoperatoren sind:

- Eingangswert kleiner als Vergleichswert: „<“
- Eingangswert gleich Vergleichswert: „==“
- Eingangswert nicht gleich Vergleichswert: „!=“
- Eingangswert größer als Vergleichswert: „>“

Im Code-Abschnitt des Hauptbildschirms ist die Syntax

```
WaitIn <Device> <Port> <Type> <Operator>  
      <Vergleichswert> <Timeout>
```

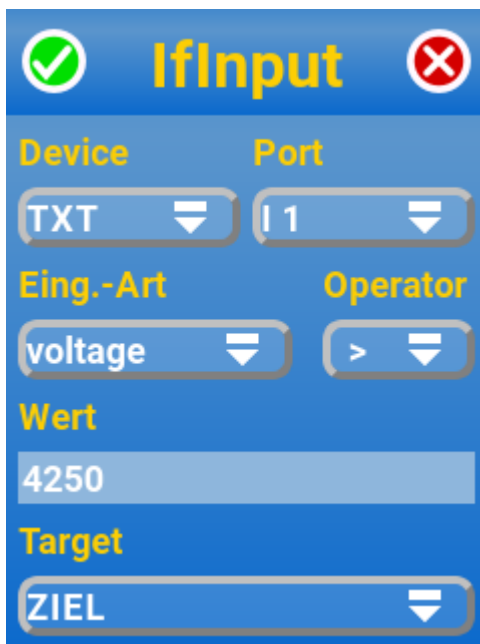
also z.B.

WaitIn RIF 1 D < 50 2500

Wartet max. 2500ms darauf, ob ein am Eingang D1 des Robo Interface angeschlossener Entfernungssensor eine Objektannäherung auf weniger als 50cm erkennt.

### 3.1.4. IfInput

Dieser Befehl arbeitet analog dem IfInputDig-Befehl. Mit ihm wird der Zustand eines analogen Einganges abgefragt und bei Erfüllung eines vorgegebenen Zustands verzweigt die Programmausführung zu einer angegebenen Sprungmarke.



**Device:** TXT, FTD oder RIF

**Port:** Je nach gewählter  
Eingangs-Art

**Eing.-Art:** Typ des Einganges (s.u.)

**Operator:** Vergleichsoperator

**Wert:** Vergleichswert

**Target:** Sprungmarke (Tag), bei der  
bei Erfüllung des Sollzustands  
fortgefahren wird

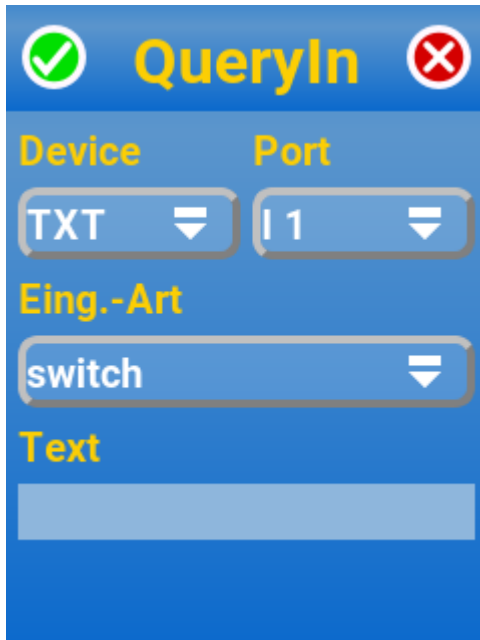
Eingangsart, Operator und Wert entsprechen der Funktion „WaitForInput“.

Im Code-Abschnitt des Hauptbildschirms ist die Syntax

```
WaitIn <Device> <Port> <Type> <Operator>  
      <Vergleichswert> <Sprungziel>
```

### 3.1.5. QueryInput

QueryInput fragt den Zustand eines Eingangs ab und gibt den eingelesenen Wert auf dem Logscreen aus.



**Device:** TXT, FTD oder RIF  
**Port:** Je nach gewählter Eingangs-Art  
**Eing.-Art:** Typ des Einganges (s.u.)  
**Operator:** Vergleichsoperator  
**Text:** Text, der der Ausgabe vorangestellt wird

Eingangsart, Operator und Wert entsprechen der Funktion „WaitForInput“.

Im Code-Abschnitt des Hauptbildschirms ist die Syntax

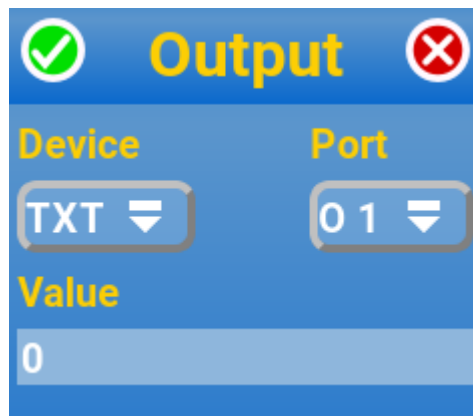
```
QueryIn <Device> <Port> <Type> <Text>
```

## 3.2. Ausgänge



### 3.2.1. Output

Der Befehl „Output“ dient zum Schalten eines einzelnen Ausganges.



**Device:** TXT oder RIF

Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:**

Die Nummer des Anschlusses

**Value:** Einstellender Wert, zwischen 0 ( = Aus ) und 7 am RIF bzw. 0 und 512 am TXT.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Output <Device> <Port> <Value>
```



also z.B.

```
Output TXT 1 255
```

Setzt den Ausgang O1 am TXT auf 255, also ungefähr halbe Leistung ( 255/512).

### 3.2.2. Motor

Der „Motor“ Befehl dient zum Ansteuern eines Motorausgangs.

**Device:** TXT oder RIF

Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:**

Die Nummer des Anschlusses

**Value:** Einstellender Wert, zwischen 0 ( = Aus ) und 7 am RIF bzw. 0 und 512 am TXT.

**Direction:** Drehrichtung „right“, „left“ oder „stop“. Dabei ist zu beachten, daß die tatsächliche Drehrichtung von der Polung des Motors abhängt.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Motor <Device> <Port> <Direction> <Value>
```

also z.B.

```
Motor TXT 1 r 512
```

Setzt den Ausgang M1 am TXT auf 512, also volle Leistung und Drehrichtung „rechts“.

### 3.2.3. MotorPulsewheel

Zur Ansteuerung eines Motors, der über einen gekoppelten Impulsgeber (üblicherweise Impulszahnrad, das einen Taster betätigt) und einen Endschalter überwacht wird. Damit läßt sich z.B. eine genaue Zielposition ansteuern.



**Device:** TXT oder RIF

Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:** Die Nummer des Anschlusses

**Value:** Einstellender Wert (Spannung), zwischen 0 ( = Aus ) und 7 am RIF bzw. 0 und 512 am TXT.

**Direction:** Drehrichtung „right“, „left“ oder „stop“. Dabei ist zu beachten, daß die tatsächliche Drehrichtung von der Polung des Motors abhängt.

**End Sw.:** Eingang, an dem der Endschalter angeschlossen ist. Der Endschalter wird nur bei Drehrichtung „links“ überwacht. D.h. per Konvention bedeutet eine Drehung rechtsherum eine Bewegung vom Endschalter weg.

**Pulse Inp.:** Eingang, an dem der Impulsgeber (Schalter) angeschlossen ist.

**Pulses:** Anzahl Impulse, für die der Motor laufen soll.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
MotorP <Device> <Port> <End Sw.>  
      <Pulse Sw.> <Direction> <Value> <Pulses>  
      - 18 -
```

also z.B.


```
MotorP TXT 1 1 2 1 400 144
```

um einen am Ausgang M1 des TXT-Controllers angeschlossenen Motor, dessen Endschalter am Eingang I1 und dessen Pulsgeber am Eingang I2 verbunden sind, linksherum mit Geschwindigkeit (Ausgangsspannung) 400 für 144 Impulse drehen zu lassen.

Diese Funktion eignet sich besonders dazu, ältere Industriemodelle ohne Encoder-Motor anzusteuern.

### 3.2.4. MotorEncoder

Zur Ansteuerung eines Encoder-Motors am TXT. Dabei müssen Motorausgangs- und Zählereingangsnummer übereinstimmen. Ein an M2 angeschlossener Motor muß sein Encoder-Signal über den Eingang C2 empfangen.



**Device:** TXT

Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:** Die Nummer des Anschlusses

**Value:** Einzustellender Wert (Spannung), zwischen 0 ( = Aus ) und 512

**Direction:** Drehrichtung „right“, „left“ oder „stop“. Dabei ist zu beachten, daß die tatsächliche Drehrichtung von der Polung des Motors abhängt.

**End Sw.:** Eingang, an dem der Endschalter angeschlossen ist. Der Endschalter wird nur bei Drehrichtung „links“ überwacht. D.h. per Konvention bedeutet eine Drehung rechtsherum eine Bewegung vom Endschalter weg.

**Pulses:** Anzahl Impulse, für die der Motor laufen soll.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
MotorE <Device> <Port> <End Sw.> <Direction> <Value>  
      <Pulses>
```

also z.B.

```
MotorE TXT 2 2 1 400 150
```

um einen am Ausgang M2 des TXT-Controllers angeschlossenen Motor, dessen Endschalter am Eingang I2 (und dessen Encoderanschluß per Konvention am Eingang C2) angeschlossen ist, linksherum mit Geschwindigkeit (Ausgangsspannung) 400 für 150 Impulse drehen zu lassen.

Hinweis zu den Encoder-Motoren:

Die alten ft-Encoder-Motore aus TX-Zeiten liefern 75 Pulse pro Umdrehung, die neueren Motoren aus dem TXT Discovery Set 66 1/3.

### 3.2.5. MotorEncoderSync

Zur synchronen Ansteuerung zweier Encoder-Motoren am TXT. Dabei müssen Motorausgangs- und Zählereingangsnummern übereinstimmen. Ein an M2 angeschlossener Motor muß sein Encoder-Signal über den Eingang C2 empfangen.

Eine Überwachung von Endschaltern ist nicht vorgesehen, die Motoren können eine bestimmte Impulszahl oder unendlich lange (Pulses=0) laufen.

Wenn eine Pulszahl angegeben ist, läuft der Befehl bis zum Erreichen derselben und stoppt die Motoren dann.

**Wird die Pulszahl Null vorgegeben, laufen beide Motore solange synchron, bis sie über einen erneuten MotorEncoderSync-Befehl mit Drehrichtung „Stop“ wieder angehalten werden.**

Damit kann z.B. ein Spurfolge- oder Hinderniserkennungs-Roboter programmiert werden, der so lange geradeaus fährt, bis er ein Hindernis erkennt.

**Device:** TXT

Es wird erst beim Programmstart überprüft, ob das gewählte Interface tatsächlich vorhanden ist.

**Port:** Die Nummer des Anschlusses

**Value:** Einstellender Wert (Spannung), zwischen 0 ( = Aus ) und 512

**Direction:** Drehrichtung „right“, „left“ oder „stop“. Dabei ist zu beachten, daß die tatsächliche Drehrichtung von der Polung des Motors abhängt.

**Sync to:** Motor, mit dem dieser Motor synchronisiert werden soll.

**Pulses:** Anzahl Impulse, für die der Motor laufen soll.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
MotorES <Device> <Port> <Sync to> <Direction> <Value>
      <Pulses>
```

also z.B.

```
MotorES TXT 3 4 1 400 1500
```

Damit laufen die Motoren M3 und M4 synchron linksherum mit Geschwindigkeit 400 für 1500 Pulse (=5 Umdrehungen bei altem Encodermotor mit 75 Pulsen pro Umdrehung)

### 3.3. Variable

startIDE kennt Integer(Ganzzahl)-Variablen und kann einige arithmetische Funktionen auf Variablen bzw. konstante Operatoren anwenden.



Bevor auf eine Variable zugegriffen werden kann, muss sie im Programmcode initialisiert werden, siehe „Init“-Funktion in Abs. 3.3.1.

Um eine Variable als Parameter zu setzen, ist das entsprechende Eingabefeld im Konfigurationsdialog für mehr als 0,5 Sekunden zu drücken. Daraufhin öffnet sich eine Auswahlliste, in der alle bereits deklarierten Variablen angezeigt werden, so dass die gewünschte Variable ausgewählt werden kann.



Im hier dargestellten Konfigurationsdialog des Output-Befehls ist die Variable „zahl“ als Parameter für den Ausgang gewählt.

Wird das entsprechende Eingabefeld für weniger als 0,5 sek. betätigt, so kann über die Bildschirmtastatur eine Zahlenkonstante als Parameter eingegeben werden.

Wichtig: werden Variablen als Parameter verwendet, so findet keine Bereichsüberprüfung statt. Eventuell ist es sinnvoll, den Wertebereich mit der calc min / calc max – Funktion (Abs. 3.3.5) zu

begrenzen.

Möchte man z.B. den Wertebereich einer Variable zur Ansteuerung eines Ausgangs auf  $0 \leq \text{var} \leq 512$  begrenzen, so wären dazu die Befehle

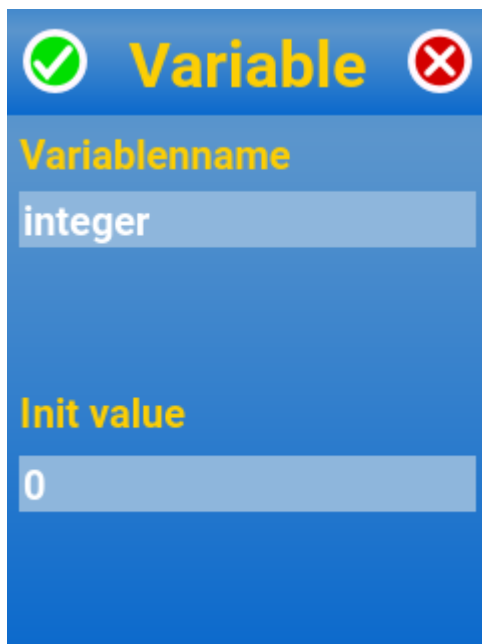
```
Calc variable variable min 512  
Calc variable variable max 0
```

zu verwenden. Der erste Calc-Befehl setzt den Wert von „variable“ auf das Minimum aus aktuellem Wert von „variable“ und 512, so daß „variable“ damit nach oben hin auf maximal 512 begrenzt wird.

Analog dazu setzt der zweite Calc-Befehl den Wert von „variable“ anschließend auf das Maximum aus aktuellem Wert und Null, so daß „variable“ nun im Bereich zwischen Null und 512 liegt.

### 3.3.1. Init

Mit dem Init-Befehl wird der Name der Variablen festgelegt und ihr ein Wert zugewiesen.



Der Initialisierungswert kann eine Zahlenkonstante oder eine bereits initialisierte Variable sein.

Während des Programmablaufs kann eine wiederholte Initialisierung vorgenommen werden, um die Variable auf einen gewünschten Wert zu setzen.

Im Codeabschnitt des Hauptfensters ist die Syntax

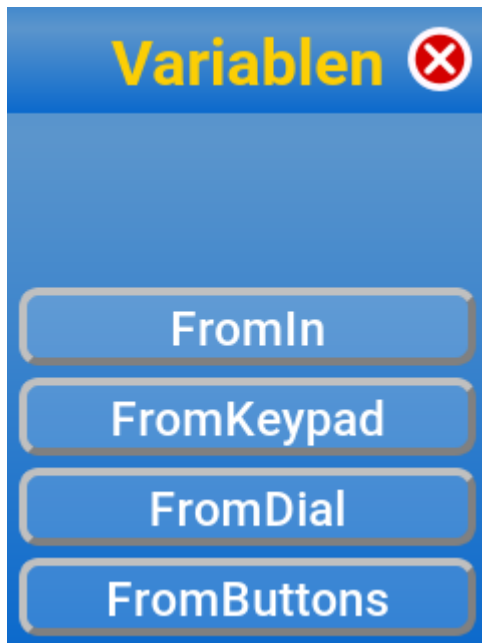
```
Init <Name> <Wert>
```

also z.B.

Init zahl 5

um die Variable „zahl“ mit dem Wert 5 zu initialisieren.

### 3.3.2. From...



In dieser Gruppe sind mehrere Funktionen zum Setzen eines Variablenwertes zusammengefaßt.

#### 3.3.2.1. FromIn



FromIn setzt den Wert einer Variablen auf den von einem angegebenen Eingang eingelesenen Wert.

Mögliche Eingangstypen sind:

- switch S (Digitaleingang)
- resistance R (Widerstand)
- voltage V (Spannung)
- distance D (Ultraschall-Abstandssensor)



Im Codeabschnitt des Hauptfensters ist die Syntax

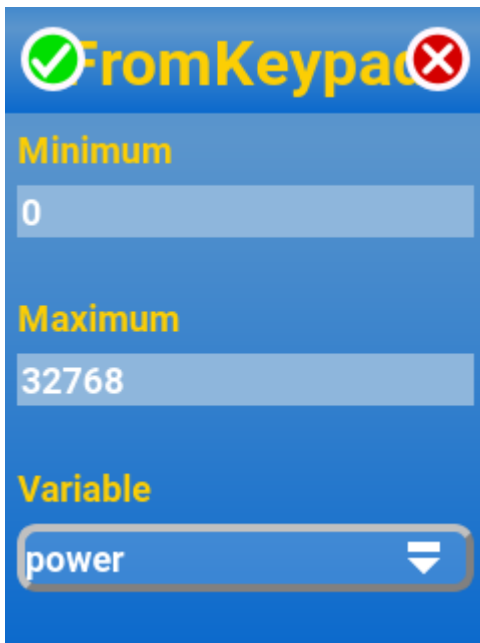
```
FromIn <device> <port> <type> <variable>
```

also z.B.

```
FromIn FTD 6 V power
```

Damit würde die am Port 6 eines ftdduino anliegende Spannung in mV in der Variable power zurückgegeben.

### 3.3.2.2. FromKeypad



FromKeypad öffnet während des Programmablaufes die Bildschirmtastatur zur Eingabe eines Wertes. Dieser wird auf den Bereich zwischen „Minimum“ und „Maximum“ begrenzt und in die angegebene Variable übertragen.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
FromKeypad <variable> <min> <max>
```

also z.B.

```
FromKeypad power 0 512
```

um eine Zahl zwischen 0 und 510 in die Variable power einzugeben.

### 3.3.2.3. FromDial



FromDial öffnet während des Programmablaufes ein Eingabefenster mit einem Drehknopf, an dem ein Wert zwischen „Minimum“ und „Maximum“ eingestellt werden kann, der anschließend in der angegebenen Variable zur Verfügung steht.

Die angegebene Textnachricht wird über dem Steller angezeigt.

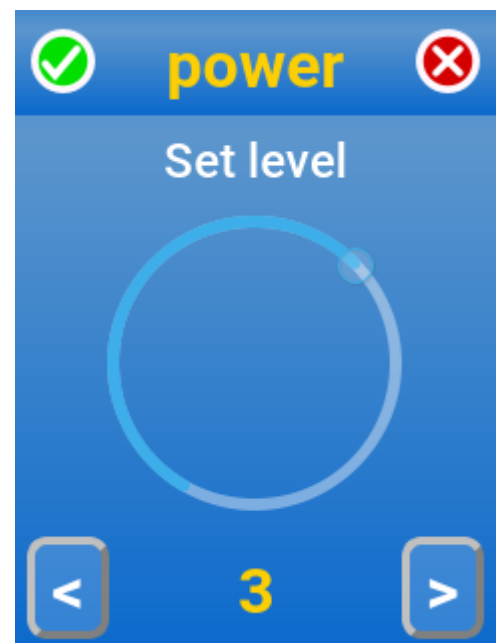
Im Codeabschnitt des Hauptfensters ist die Syntax

```
FromDial <variable> <min> <max> <text>
```

also z.B.

```
FromDial power -10 10 Set level
```

um eine Zahl zwischen -10 und 10 in die Variable power einzugeben, wobei der Text „Set level“ im Eingabefenster angezeigt wird:



### 3.3.2.4. FromButtons



FromButtons öffnet während des Programmablaufes ein Eingabefenster bis zu sieben Buttons.

Die Nummer des vom Benutzer gewählten Knopfes wird in der Variablen zurückgegeben.

Der Name der Variable wird als Titel des Eingabefensters verwendet.

Mit dieser Funktion lassen sich somit interaktive Menusteuerungen verwirklichen, so zum Beispiel die Steuerung eines Hochregallagers:

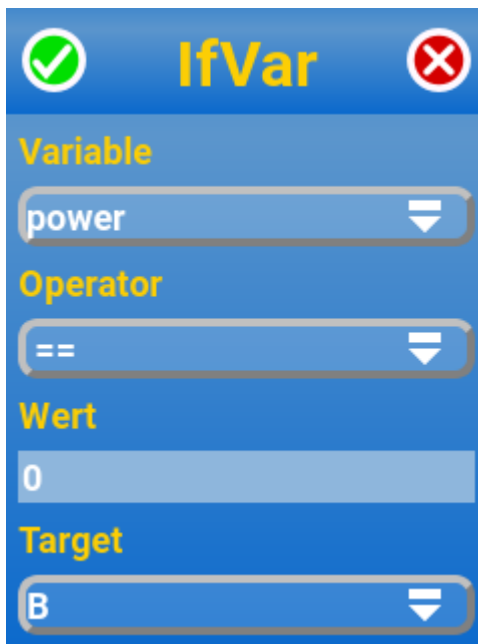
```
# Hochregal
Init Hochregal
Tag start
FromButtons Hochregal Einlagern Auslagern Inventur Ende
IfVar Hochregal == 1 einlagern
IfVar Hochregal == 2 auslagern
IfVar Hochregal == 3 inventur
Stop
# Unterrountinen
Tag einlagern
...
Jump start
Tag auslagern
...
Jump start
Tag inventur
...
Jump start
```



### 3.3.3. QueryVar


Mit QueryVar werden der Variablenname und der Wert, getrennt durch ":" ausgegeben.

### 3.3.4. IfVar



IfVar führt einen Vergleich der angegebenen Variable mit dem gewählten Vergleichsoperator und dem vorgegebenen Wert durch. Fällt der Vergleich positiv aus, so wird die Programmausführung an der unter „Target“ angegebenen Sprungmarke (Tag) fortgesetzt.

### 3.3.5. Calc



Die Calc-Funktion ist die wohl komplexeste Funktion von startIDE. Sie führt die mathematische Verknüpfung zweier Operanden mittels eines Operators aus und übergibt das Ergebnis in die angegebene Zielvariable.

Die beiden Operanden können Konstanten oder Variablen sein.

Als Operatoren stehen zur Verfügung:

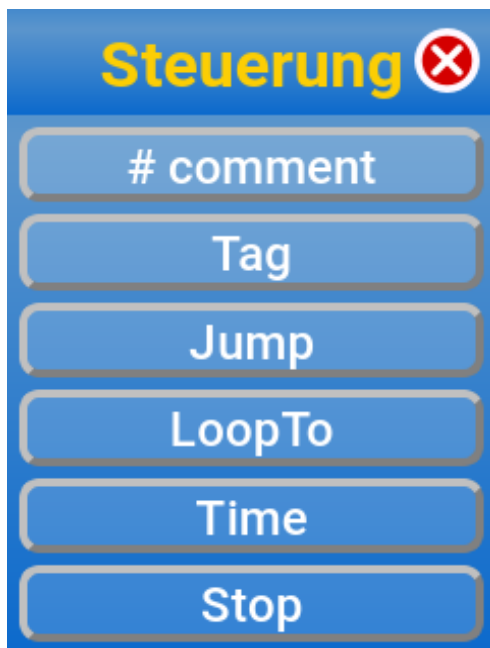
- die **Grundrechenarten** „+, - , \*, /“
  - Calc var var + 1 entspricht var = var + 1
- der **Modulo-Operator** „mod“ (liefert den Rest einer Ganzzahldivision)
  - Calc var 9 mod 5 entspricht var = 9 mod 5
- der **Exponentialoperator** „exp“
  - Calc var var exp 2 entspricht var = var ^ 2
- der **Wurzeloperator** „root“ liefert die <operand 1>. Wurzel aus <operand 2>
  - Calc var 2 root 9 entspricht var =  $\sqrt[2]{9}$
- der **min-** und **max-Operator** liefern jeweils das Minimum oder Maximum der beiden Operanden zurück
- **sin-** und **cos-Operator** liefern einen skalierten ganzzaligen Sinus- oder Cosinus-Wert des in Grad angegebenen Winkels zurück
  - Calc var faktor sin winkel entspricht var=faktor \* sin(winkel)

Weiterhin gibt es noch **boolesche** und **Vergleichsoperatoren**:

- **&&** und **||** sind boolesche und- bzw. oder-Verknüpfung der beiden Operanden. Dabei wird „0“ als Falsch und alle anderen Werte als Wahr interpretiert. Rückgabewerte sind „0“ oder „1“
- 
- „<, <=, ==, !=, >= , >“ vergleichen Operand 1 mit Operand 2 und liefern 0 (Falsch) oder 1 (Wahr) zurück.

## 3.4. Steuerung

Hier sind die Funktionen



zur Programmablaufsteuerung zu finden.

### 3.4.1. # comment

Hiermit wird ein Kommentartext in den Programmcode eingefügt.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
# <Kommentar>
```

also z.B.

```
# Hier startet das Hauptprogramm
```

### 3.4.2. Tag

Der Tag-Befehl definiert eine Sprungmarke innerhalb des Programmcodes, die mit „IfInputDigital“, „Jump“ oder „LoopTo“ angesprungen werden kann, d.h. bei Erreichen der Programmzeile „Jump <Sprungmarke >“ wird die Programmausführung bei der Zeile „Tag <Sprungmarke>“ fortgeführt.

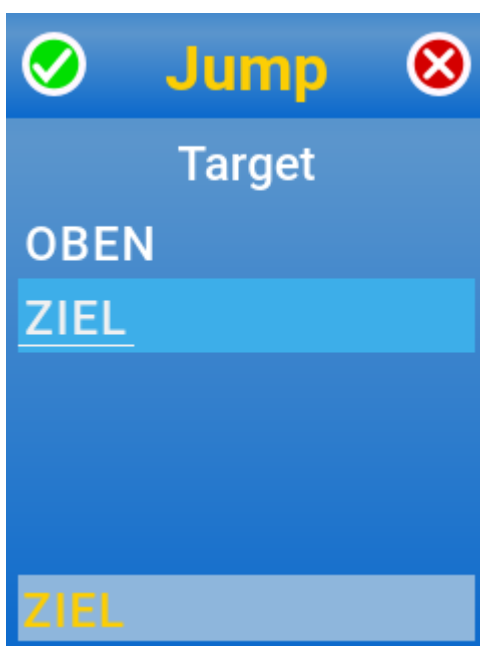
Im Codeabschnitt des Hauptfensters ist die Syntax

```
Tag <Sprungmarke>
```

also z.B.

```
Tag START
```

### 3.4.3. Jump



Jump ist ein Sprungbefehl. Bei Erreichen der Programmzeile „Jump <Jump Tag>“ wird die Programmausführung bei der Zeile „Tag <Jump Tag>“ fortgeführt. Damit ist die Programmierung von (Endlos-)Schleifen möglich.

Es muß mindestens ein „Tag“ im Programmcode definiert sein, damit der Jump-Befehl eingefügt werden kann.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Jump <Jump Tag>
```

also z.B.

```
Jump START
```

Eine Endlosschleife wäre:

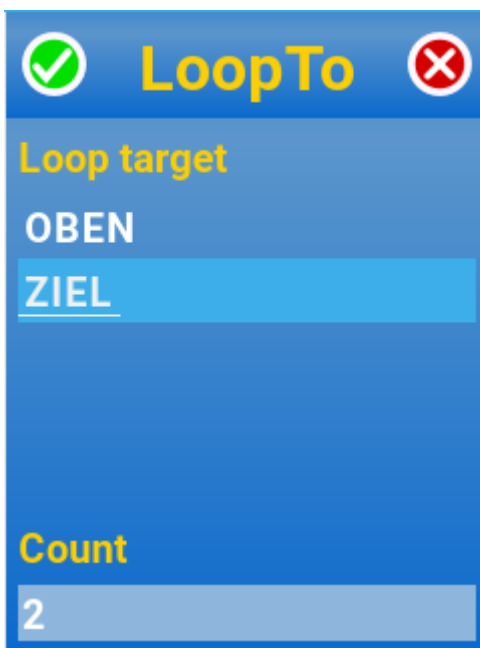
```
Tag oben  
Print Hallo  
Jump oben
```

### 3.4.4. LoopTo

LoopTo ist eine Zählschleife, das heißt, bei Erreichen des LoopTo Befehls wird die Programmausführung für eine bestimmte Anzahl von Durchläufen beim angegebenen Sprungziel fortgeführt.

Ist die vorgegebene Anzahl von Durchläufen erreicht, wird statt des Sprunges mit dem auf LoopTo folgenden Befehl fortgefahren.

Es muß mindestens ein „Tag“ im Programmcode definiert sein, damit der LoopTo-Befehl eingefügt werden kann.



**Loop Target:** Sprungmarke, zu der gesprungen werden soll.

**Count:** Anzahl der Wiederholungen, mindestens 1

Im Codeabschnitt des Hauptfensters ist die Syntax

```
LoopTo <Jump Tag> <count>
```

also z.B.

```
LoopTo START 5
```

Hier wird bei den ersten fünf Programmdurchläufen zum Tag „START“ gesprungen.



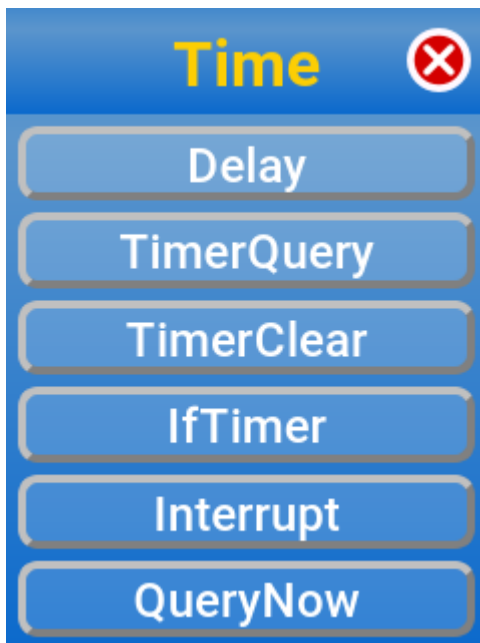
Eine Schleife sähe demnach so aus:

```
Tag START
...
<weitere Befehle>
...
LoopTo START 5
```

**Es ist zu beachten, daß LoopTo nicht prüft, ob die Sprungmarke vor oder nach dem LoopTo-Befehl liegt.**

Beides ist möglich und kann ggf. sinnvoll sein.

### 3.4.5. Time



Hier sind Funktionen zu finden, die eine zeitliche Steuerung des Programmablaufes ermöglichen.

#### 3.4.5.1. Delay

Delay verzögert den Programmablauf für die angegebene Zeit in Millisekunden.

Damit kann z.B. die Verzögerung zwischen Ein- und Ausschalten eines Ausganges definiert werden.

Wenn eine **negative Zeit** eingegeben wird, so wird für eine **zufällige Zeitspanne** zwischen 0 und der eingegebenen Zeit gewartet.

Im Code wird dann der Betrag der eingegebenen Zahl mit einem nachgestellten „R“, dem random flag, angezeigt.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Delay <Zeit in ms> <random flag>
```

also z.B.

```
Delay 1500
```

Das Beispiel führt zu einer Pause von 1,5 sek im Programmablauf.

Delay 5000 R

führt zu einer zufälligen Pause zwischen 0 und 5 sek.

### 3.4.5.2. TimerQuery

TimerQuery liefert die aktuelle Zeit seit Start des startIDE-Programmes in Millisekunden im Format „Timer: <Zeit>“ zurück.

### 3.4.5.3. TimerClear

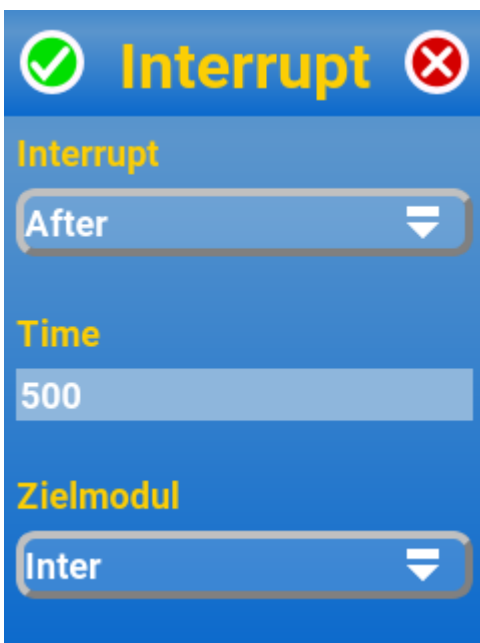
TimerClear setzt den Zeitzähler auf Null zurück, so daß mit TimerQuery die verstrichene Zeit seit Zurücksetzung abgefragt werden kann.

### 3.4.5.4. IfTimer

Mit IfTimer läßt sich abfragen, ob seit Start des Programmes bzw. letztem TimerClear-Befehl eine bestimmte Zeit in Millisekunden (noch nicht) verstrichen ist.

Ist die Abfrage positiv, so wird die Programmausführung an der angegebenen Sprungmarke (Tag) fortgesetzt.

### 3.4.5.5. Interrupt



startIDE kann einen Interrupt, d.h. eine Unterbrechung der Programmausführung nach Ablauf einer bestimmten Zeit, ausführen.

Dabei wird das angegebene Zielmodul aufgerufen und ausgeführt.

Der Interrupt kann einmalig („After“) nach Ablauf der angegebenen Zeit (in Millisekunden) oder wiederholend („Every“) jeweils nach Ablauf der Zeitspanne ausgeführt werden.

Mit „off“ wird der Interrupt deaktiviert.

Mit dieser Funktion ist z.B. die Abfrage eines Ausschalters in regelmäßigen Zeitabständen möglich:

```
# Spannungslogger
Interrupt Every 100 stoptaster
TimerClear
Log 1
Tag schleife
Clear
TimerQuery
QueryIn TXT 1 V I1(mV):
Delay 25
Jump schleife
#
Module stoptaster
IfInDig TXT 2 False weitermachen
Log 0
Stop
Tag weitermachen
MEnd
```

In diesem Beispiel wird die Spannung an Eingang I1 des TXT fortlaufend abgefragt und auf dem Bildschirm und in einer Logdatei protokolliert. Das Programm endet, wenn ein an Eingang I2 des TXT angeschlossener Taster betätigt wird. Zu diesem Zweck wird über den Interrupt alle 100ms das Modul „stoptaster“ aufgerufen, das überprüft, ob der Taster „nicht betätigt“ ist und in diesem Fall das Programm weiterlaufen läßt.

Es ist zu beachten, dass der Interrupt nur jeweils nach Ausführung einer startIDE-Programmzeile überprüft wird.

Das heißt, dass Befehle, die längere Zeit zur Ausführung brauchen oder den Programmablauf sogar anhalten (z.B. Delay, WaitIn oder Motorbefehle, die den Motor für eine bestimmte Zeit laufen lassen), zu einer verspäteten Ausführung des Interrupts führen können.

Gegebenenfalls sind diese Befehle dann zu vermeiden, indem z.B. statt eines

```
Delay 1000
```

die Befehlsfolge

```
TimerClear  
Tag warte  
IfTimer < 1000 warte
```

verwendet wird, die nichtblockierend arbeitet. Ähnliches gilt für WaitIn, das analog des obigen Beispiels durch IfIn ersetzt werden kann.

#### **3.4.5.6. QueryNow**

QueryNow gibt das aktuelle Datum und die Uhrzeit in der Form

```
Now: YYYY-MM-DD_HH:MM:SS
```

auf dem Log-Screen aus. Dies ist insbesondere für Langzeit-Logging in Verbindung mit der Log-Datei interessant, wenn man z.B. über mehrere Tage in größeren Zeitabständen Meßwerte sichern will.

#### **3.4.6. Stop**

Das Stop-Kommando beendet die Programmausführung. In seiner Wirkung entspricht es dem Erreichen des Programmendes. Alle Ausgänge werden abgeschaltet, das Ausgabelog bleibt geöffnet.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Stop
```

### 3.5. Module

Module sind in sich geschlossene Programmblöcke.

Beispiel:

```
Module LAMPEN_AN  
Output RIF 1 7  
Output RIF 2 7  
MEnd
```

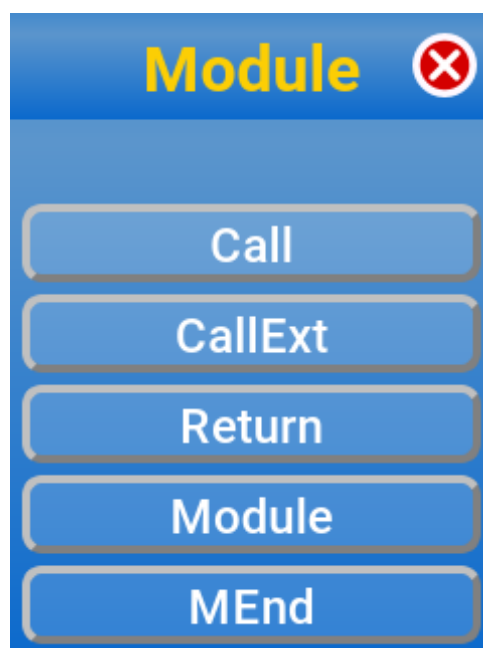
Dieses Modul mit dem Namen „LAMPEN\_AN“ würde bei seiner Ausführung die Ausgänge 1 und 2 an einem Robo Interface auf Stufe 7 einschalten.

Um ein Modul auszuführen, muß es mit dem „Call <Modulname>“ Befehl aufgerufen werden.

Nach Beendigung des Moduls wird die Programmausführung in der auf den Modulaufruf („Call“) folgenden Programmzeile fortgesetzt.

**Stößt startIDE bei Programmausführung auf das „Module“-Schlüsselwort, endet die Programmausführung. Module werden nicht ausgeführt, wenn sie nicht explizit aufgerufen werden. Daher müssen Module immer am Ende des Programmcodes eingefügt werden.**

Es gibt folgende Modul-Funktionen:



### 3.5.1. Call / CallExt

Der Call-Befehl dient zum ggf. mehrmaligen Aufrufen eines im Programmcode definierten Moduls.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Call <Modulname> <Anzahl>
```

also z.B.

```
Call LAMPEN_AN 2
```

Hier würde ein Modul mit dem Namen „LAMPEN\_AN“ zwei Mal aufgerufen.

CallExt arbeitet analog, allerdings wird hierbei ein vorher exportiertes Modul, das nicht Bestandteil des Programmcodes ist, aufgerufen.

Existiert ein gleichnamiges Modul im Programmcode, so hat dieses Vorrang.

### 3.5.2. Return

Der Return-Befehl beendet die Ausführung eines Moduls VOR Erreichen des durch Mend definierten Modulendes.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Return
```

### 3.5.3. Module

Hiermit wird unter Angabe des gewünschten Namens ein Modul-Anfang definiert.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Module <Modulname>
```

also z.B.

```
Module LAMPEN_AN
```

### 3.5.4. MEnd

MEnd schließt einen Modulblock ab. Bei Erreichen von MEnd wird die Ausführung des Moduls beendet und zum aufrufenden Call-Befehl zurückgekehrt.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
MEnd
```

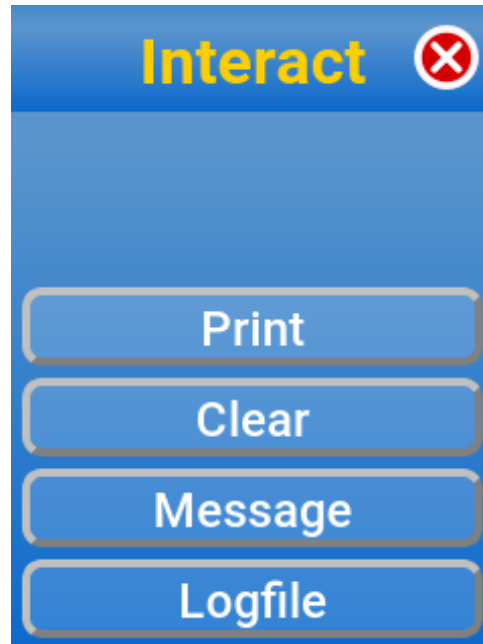
Ein Beispiel für die Verwendung von Modulen:

```
# Start
Call LAMPEN_AN
Delay 1000
Call LAMPEN_AUS
Delay 1000
# Programmende
# Lampen an
Module LAMPEN_AN
Output RIF 1 7
Output RIF 2 7
MEnd
# Lampen aus
Module LAMPEN_AUS
Output RIF 1 0
Output RIF 2 0
MEnd
```



## 3.6. Interaction

Unter „Interaction“ finden sich einige Befehle, die eine grundlegende Interaktion mit dem Nutzer ermöglichen.



### 3.6.1. Print

Der Print-Befehl gibt eine Nachricht auf dem Log-Screen aus.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Print <Text>
```

also z.B.

```
Print Hallo Welt
```

Alles, was auf Print folgt, wird als auszugebender Text interpretiert.

### 3.6.2. Clear

Der Clear-Befehl löscht den Inhalt des Log-Screens.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Clear
```

### 3.6.3. Message

Der Message-Befehl öffnet ein Benachrichtigungsfenster, das mit einem Knopfdruck bestätigt werden muß.

Damit kann man wichtige Meldungen ausgeben oder auf Freigabe durch den Benutzer warten.

Der Message-Befehl benötigt neben dem anzuzeigenden Nachrichten-Text auch einen Text für den Bestätigungs-Knopf.

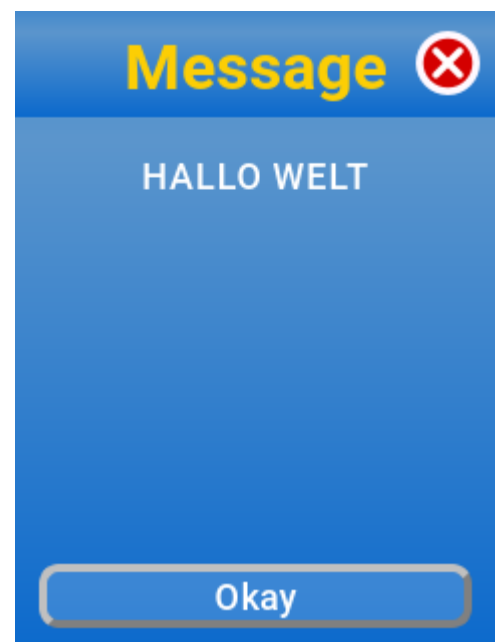
Im Codeabschnitt des Hauptfensters ist die Syntax

```
Message <Text>'<Buttontext>
```

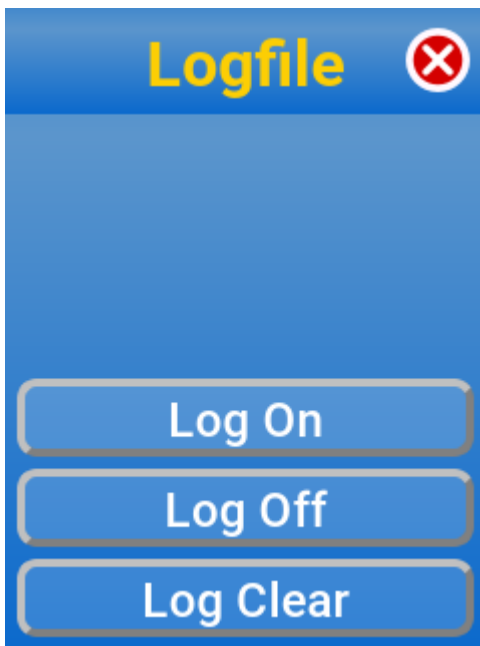
also z.B.

```
Message HALLO WELT'Okay
```

Nachricht und Knopf-Text sind durch ein Hochkomma (einfaches Anführungszeichen oben) getrennt.



### 3.6.4. Logfile



Mit Log On wird das Protokollieren der Log-Screen-Ausgaben in eine Datei gestartet.

Log Off beendet das Protokollieren in die Datei.

Log Clear löscht ALLE auf dem TXT vorhandenen Logfiles.

Die Logfiles werden nach dem Schema  
log<yyyymmdd-hhmmss>.txt

also z.B. log20171228-014045.txt benannt.

Über das Webinterface von startIDE kann auf die Logfiles zugegriffen werden.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Log < 0 | 1 | C >
```

also z.B.

```
Log 1
```

zum Einschalten des Protokollierens in die Logdatei.

Wenn in einer Schleife gleichförmige Datensätze im Format  
<text> <wert> in die Logdatei geschrieben werden, kann diese über das Webinterface im csv-Format heruntergeladen werden, um z.B. anschließend in einer Tabellenkalkulation die Daten weiter bearbeiten zu können.

Ein Spannungslogger, der im Sekundenabstand die Spannung an Eingang 1 des TXT mißt, sähe so aus:

```
# Spannungslogger
Log 1
TimerClear
Tag top
TimerQuery
QueryIn TXT 1 V U_1(mV):
Delay 995
Jump top
```

Das Programm muss manuell gestoppt werden, dabei wird auch das Logging beendet.

## 4. Beispiele

### 4.1. Mitgelieferte startIDE-Projekte

Der App sind bereits einige Beispiele beigelegt, die über das Menu „Project → Load“ geladen werden können.

#### 4.1.1. c\_Ampel

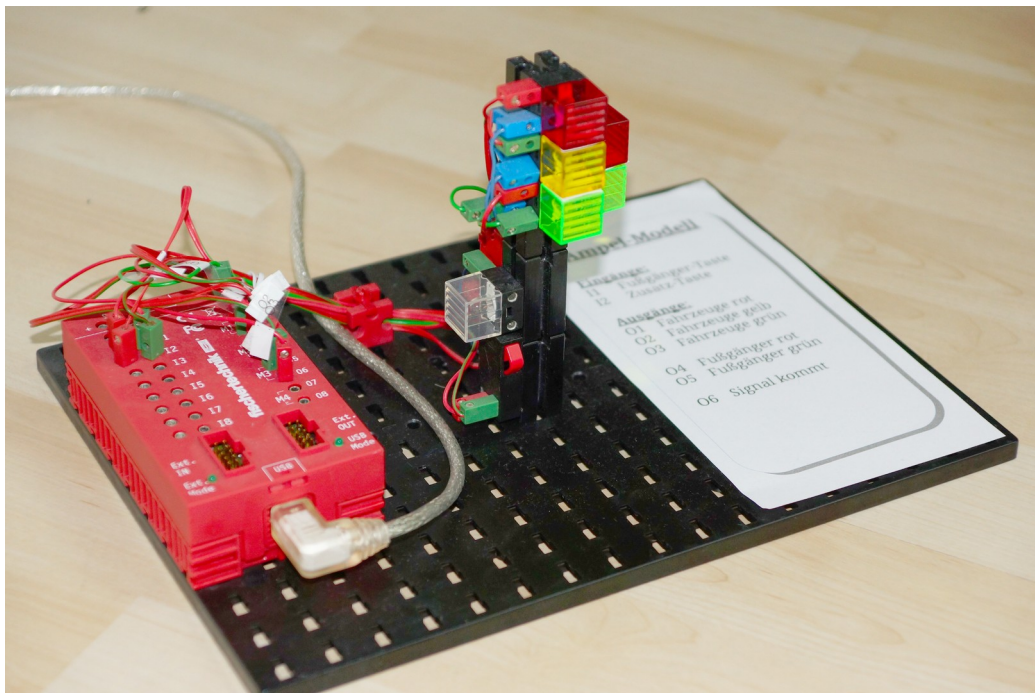
Dies ist eine einfache Fußgängerampel-Steuerung. Das Programm ist für die Robo Interface Familie (RIF) ausgelegt.

Es soll eine Fahrzeugampel (rot-gelb-grün) und eine Fußgängerampel (rot-grün) angesteuert werden. Die Ampel hat eine Fußgänger-Bedarfstaste und eine zweite Taste zum Beenden des Programmes.

An Eingang I1 ist die Fußgänger-Taste angeschlossen, an Eingang I2 die Zusatztaste.

An Ausgang O1-O3 sind die drei Lampen der Fahrzeugampel in der Reihenfolge rot-gelb-grün angeschlossen.

An Ausgang O4 und O5 sind rote und grüne Lampe der Fußgängerampel angeschlossen, und an O6 die „Signal kommt“-Leuchte.



### 4.1.2. c\_Blink

Das Programm c\_Blink läßt eine an Ausgang O1 eines RIF angeschlossene Lampe im 2Hz-Takt aufblitzen.

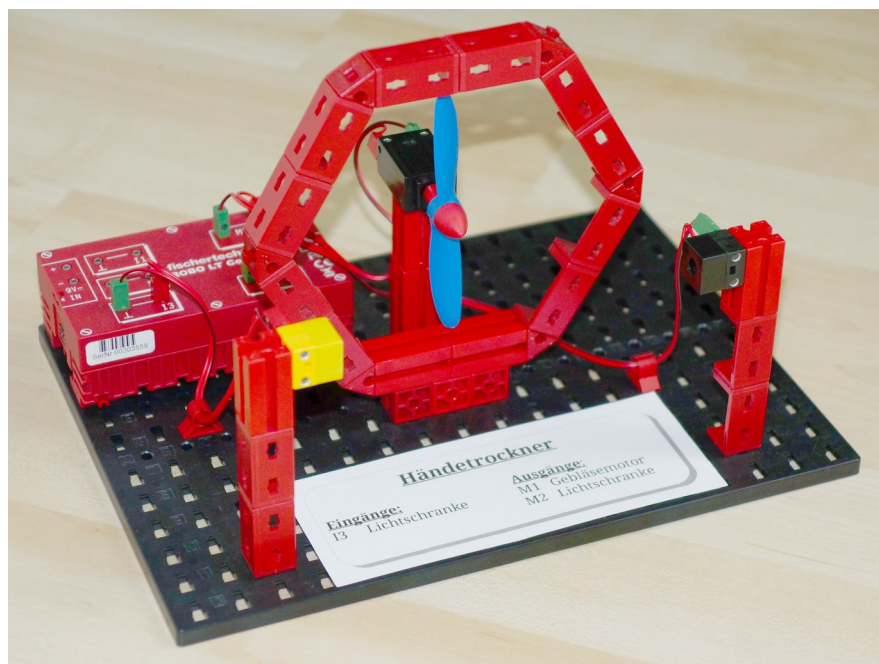
### 4.1.3. c\_Haendetrockner

Steuerungsprogramm für einen Händetrockner, z.B. aus dem Robo LT Beginner Lab.

Der Händetrockner besteht aus einem Motor mit Luftschaube als Gebläse und einer Lichtschranke, bei deren Unterbrechung das Gebläse anlaufen soll.

Der Fototransistor der Lichtschranke ist an Eingang I3 angeschlossen, die dazugehörige Lampe an Ausgang M2.

Der Gebläsemotor ist an Ausgang M1 angeschlossen.



### 4.1.4. c\_Lauflicht und c\_Signalfeuer

c\_Lauflicht benutzt den unter 4.1.1. beschriebenen Aufbau, um nacheinander alle Lampen anzusteuern.

c\_Signalfeuer ist das unter Abschn. 5.1. dargestellte Beispiel.

## 4.2. Programmieraufgaben

### Hallo Welt

In jeder Programmiersprache gibt es „Hallo Welt“-Programme, die zu didaktischen Zwecken den Minimalumfang eines Programmes in der jeweiligen Sprache darstellen sollen. Außerdem ist es eine nette Tradition, eine neue Programmierumgebung zu begrüßen.

#### 4.1.1. Hallo Welt 1

**Aufgabe:** Der Text „Hallo Welt“ soll auf dem Logscreen ausgegeben werden.

**Lösung:**

```
# Hallo Welt 1  
Print Hallo Welt
```

#### 4.1.2. Hallo Welt 2

**Aufgabe:** Wir wollen wissen, ob wir allein im Universum sind. Also senden wir die Nachricht: „Hallo Welt, ist da jemand?“, die mit „Ja!“ bestätigt werden soll.

**Lösung:**

```
# Hallo Welt 2  
Message Hallo Welt, ist da jemand?'Ja!
```

#### 4.1.3. Hallo Welt 3

**Aufgabe:** Ein klassisches, unter Informatikern zu beobachtendes Phänomen ist, daß kleine Erfolgserlebnisse zu einem überhöhten Selbstbild führen.

Daher wollen wir nach der mit „Sofort!“ zu bestätigenden Nachricht „Es werde Licht!“ auch noch eine an Ausgang M1 des TXT (analog Robo IF) angeschlossene Lampe für 5 Sekunden einschalten.

## Lösung:

```
# Hallo Welt 3
Message Es werde Licht!'Sofort!
Motor TXT 1 1 512
Delay 5000
```

bzw. für Robo Interfaces

```
# Hallo Welt 3
Message Es werde Licht!'Sofort!
Motor RIF 1 1 7
Delay 5000
```

### 4.1.4. Gottes Rache

Die Strafe für unser anmaßendes Schöpfungsgehabe – es werde Licht – folgt auf der Stelle. Gott / Zarquon / jenes höhere Wesen, das wir verehren läßt sintflutartigen Regen fallen...

**Aufgabe:** Wir beschließen, mit der an Ausgang M1 angeschlossenen Lampe SOS zu signalisieren. 3x kurz, 3x lang, 3x kurz.  
Ein kurzes Blinken soll aus 0,25 sec an / 0,25 sec aus, ein Langes aus 0,5 sec an / 0,25 sec aus bestehen.

## Lösung:

```
# SOS
Motor TXT 1 1 512
Delay 250
Motor TXT 1 s 0
Delay 250
Motor TXT 1 1 512
Delay 250
Motor TXT 1 s 0
Delay 250
Motor TXT 1 1 512
```



```

Delay 250
Motor TXT 1 s 0
Delay 250
# jetzt lang
Motor TXT 1 l 512
Delay 500
Motor TXT 1 s 0
Delay 250
Motor TXT 1 l 512
Delay 500
Motor TXT 1 s 0
Delay 250
Motor TXT 1 l 512
Delay 500
Motor TXT 1 s 0
Delay 250
# und wieder kurz
Motor TXT 1 l 512
Delay 250
Motor TXT 1 s 0
Delay 250
Motor TXT 1 l 512
Delay 250
Motor TXT 1 s 0
Delay 250
Motor TXT 1 l 512
Delay 250
Motor TXT 1 s 0

```

### **Lösung mit Schleifen:**

```

# SOS 2
Tag KURZ1
Motor TXT 1 l 512
Delay 250
Motor TXT 1 s 0
Delay 250
LoopTo KURZ1 3
Tag LANG
Motor TXT 1 l 512
Delay 500
Motor TXT 1 s 0
Delay 250
LoopTo LANG 3

```

```
Tag KURZ2
Motor TXT 1 1 512
Delay 250
Motor TXT 1 s 0
Delay 250
LoopTo KURZ2 3
```

### **Lösung mit Modulen:**

```
# SOS 3
Call KURZ 3
Call LANG 3
Call KURZ 3
# Module
Module KURZ
Motor TXT 1 1 512
Delay 250
Motor TXT 1 s 0
Delay 250
MEnd
Module LANG
Motor TXT 1 1 512
Delay 500
Motor TXT 1 s 0
Delay 250
MEnd
```

#### 4.1.5. Es regnet immer noch...

...und wir beschließen dauerhaft SOS zu funken...

**Aufgabe:** Das SOS-Signal soll in Endlosschleife mit einer Pause von 1 sek gesendet werden.

#### Lösung mit Modulen:

```
# SOS 4
Tag TOP
Call KURZ 3
Call LANG 3
Call KURZ 3
Delay 1000
Jump TOP
# Module
Module KURZ
Motor TXT 1 l 512
Delay 250
Motor TXT 1 s 0
Delay 250
MEnd
Module LANG
Motor TXT 1 l 512
Delay 500
Motor TXT 1 s 0
Delay 250
MEnd
```

## 5. Tips und Tricks

### 5.1. Programmierhinweise

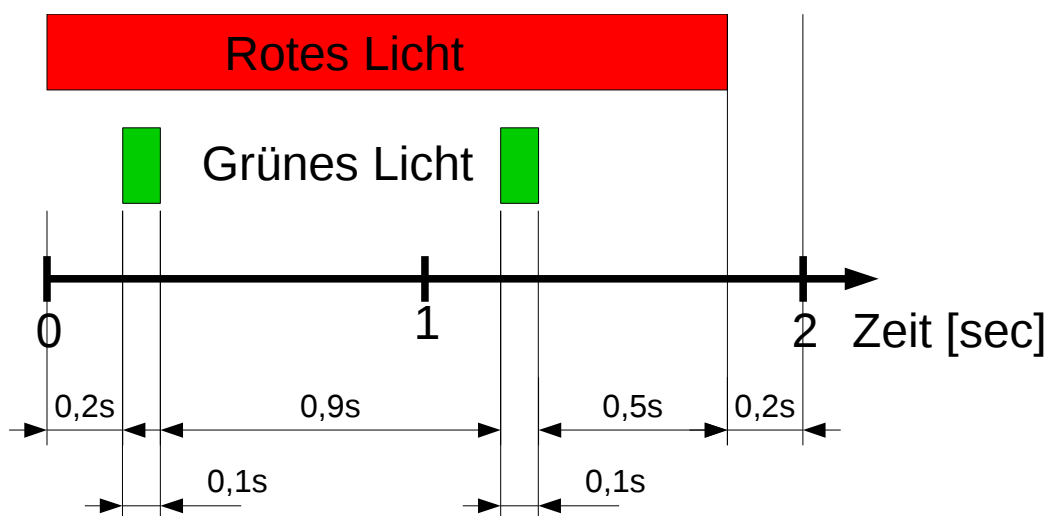
Es ist sinnvoll, vor Beginn der Programmerstellung einen Ablaufplan auf Papier zu skizzieren.

Auch ohne Multithreading lassen sich viele Aufgaben lösen, wenn man sie entsprechend umsetzt.

#### Ein Beispiel:

Eine rote und eine grüne Lampe sollen zyklisch blinken. Dabei soll die rote Lampe 1,8sec an und 0,2 sec aus sein, die grüne Lampe soll 1x pro Sekunde für 0,1 sec aufblitzen. Dabei soll sie zur roten Lampe 0,2 sec verzögert angehen.

Anstatt zwei parallele Prozesse zu starten, die die Lampen im ein- bzw. zwei-Sekunden-Zyklus schalten, werden die Ereignisse in eine serielle Reihenfolge gebracht:



Das Programm dazu:

```
# Signalfeuer
Tag ANFANG
Output RIF 1 7
Delay 200
Output RIF 3 7
Delay 100
```

```
Output RIF 3 0
Delay 900
Output RIF 3 7
Delay 100
Output RIF 3 0
Delay 500
Output RIF 1 0
Delay 200
Jump ANFANG
```

In diesem Beispiel ist die rote Lampe an Ausgang O1, die grüne Lampe an Ausgang O3 eine Robo Interface angeschlossen.

Im Anhang befindet sich als weiteres Beispiel ein Ablaufplan zur Steuerung eines Säulenroboters zur Lösung der „Turm von Hanoi“-Aufgabe.

## 5.2. Kopieren von Programmteilen

Möchte man einen größeren Block von Programmzeilen kopieren, so ist es sinnvoll, diesen in ein Modul zu verpacken (Vorangestelltes „Module <name>“ und nachgestelltes „MEnd“)

Dann kann man dieses Modul exportieren und beliebig oft wieder importieren.

Die Zeilen „Module <name>“ und „MEnd“ sind danach, ebenso wie das temporär gespeicherte Modul, sinnvollerweise wieder zu löschen.

## 5.3. Debugging / Fehlersuche

Da startIDE-Programme über das grafische Interface zusammengestellt werden, sollten Syntaxfehler grundsätzlich nicht auftreten.

startIDE kennt aber trotzdem zwei Ausnahmefehler, die auf dem Logscreen gemeldet werden und zum sofortigen Programmabbruch führen:

Der „**DontKnowWhatToDo**“-Fehler tritt auf, wenn startIDE bei der Programmausführung auf einen unbekannten Befehl trifft. Dies kann vorkommen, wenn ein Programm mit einer neueren startIDE-Version erstellt wurde, als jene, die nun zur Ausführung verwendet wird, und somit ein Befehl noch nicht implementiert ist.

Der „**CompleteConfusionError**“ tritt auf, wenn während der Ausführung eines Befehls Komplikationen auftreten. Dies kann insbesondere der Fall sein, wenn die Parametrierung des gerade ausgeführten Befehls in irgendeiner Form fehlerhaft ist.

Die beiden vorgenannten Ausnahmefehler können auch dann auftreten, wenn der Programmcode außerhalb von startIDE modifiziert wurde und dabei syntaktische oder logische Fehler gemacht wurden.

Um den Programmablauf nachverfolgen zu können, gibt es zwei Sonderfunktionen:

Mit dem Kommentar TRACEON (Code: # **TRACEON**) wird jede darauf folgende ausgeführte Codezeile auf dem Logscreen ausgegeben. Mit dem Kommentar TRACEOFF wird diese Funktion wieder abgeschaltet.

In Verbindung mit der Logdatei-Funktion können hiermit recht ausführliche Protokolle erstellt werden.

Mit dem Kommentar STEPON (Code: # **STEPON**) wird der Programmcode in Einzelschritten ausgeführt. Nach jeder Codezeile muß durch antippen des Bildschirms die Ausführung bestätigt werden. Diese Funktion wird mit dem Kommentar STEPOFF (Code: # **STEPOFF**) wieder abgeschaltet.

Außerdem kann man per Print-Befehl das Erreichen der jeweiligen Code-Stelle anzeigen. Auch der Message-Befehl eignet sich zum Verfolgen des Programmablaufs, da er die Nachricht mit einem Haltepunkt verbindet, so daß die Fortführung des Programmablaufs erst bestätigt werden muß.

Zur (Lauf-)Zeitmessung sind die Debug-Befehle **TIMERCLEAR** und **GETELAPSEDTIME** verfügbar.

Beim Start eines startIDE-Projektes wird ein interner Zeitstempel gesetzt. Die seit Programmstart verstrichene Zeit in Millisekunden als Fließkommazahl läßt sich mit „**# GETELAPSEDTIME**“ auf dem Log-Screen ausgeben.

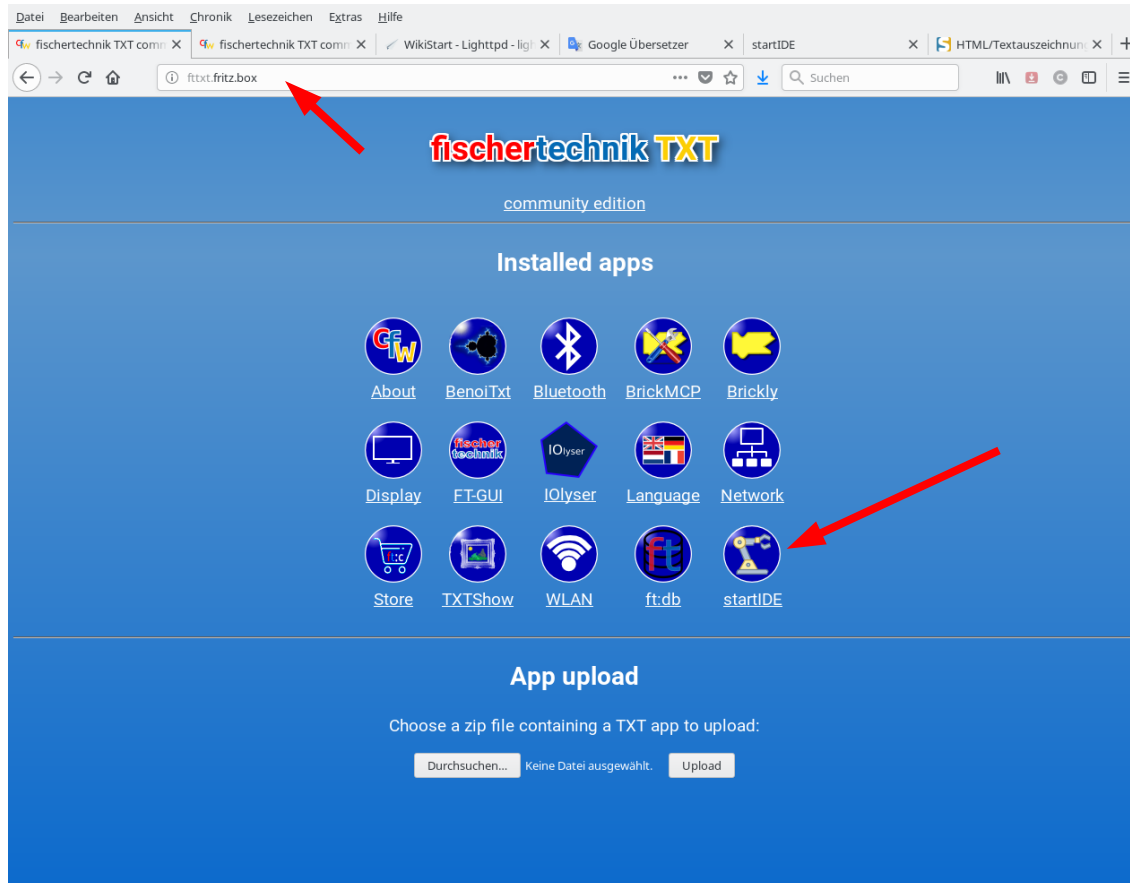
Mit „**# TIMERCLEAR**“ wird während des Programmablaufs ein neuer Zeitstempel gesetzt, so daß die Zeitmessung wieder bei Null beginnt.

Ein einfacher Reaktionstest (Lampe an M1, Taster an I1) sieht so aus:

```
# Reaktionstest
Print Bei Aufleuchten der
Print Lampe Taste druecken!
Delay 1000
Delay 5000 R
Motor TXT 1 l 255
# TIMERCLEAR
WaitInDig TXT 1 Raising 0
# GETELAPSEDTIME
```

## 5.4. startIDE-Projekte auf PC übertragen

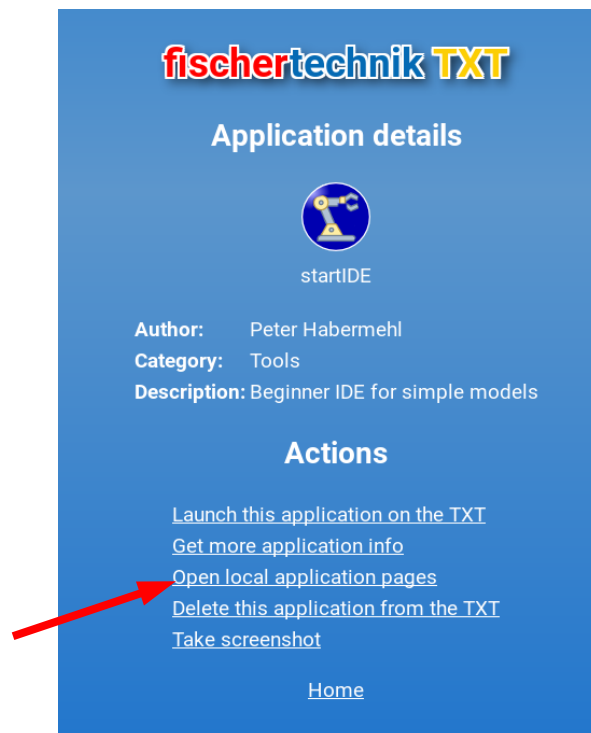
Wenn der Controller mit einem Netzwerk verbunden ist, so kann man per Webinterface auf die startIDE-Projekte und Module zugreifen:



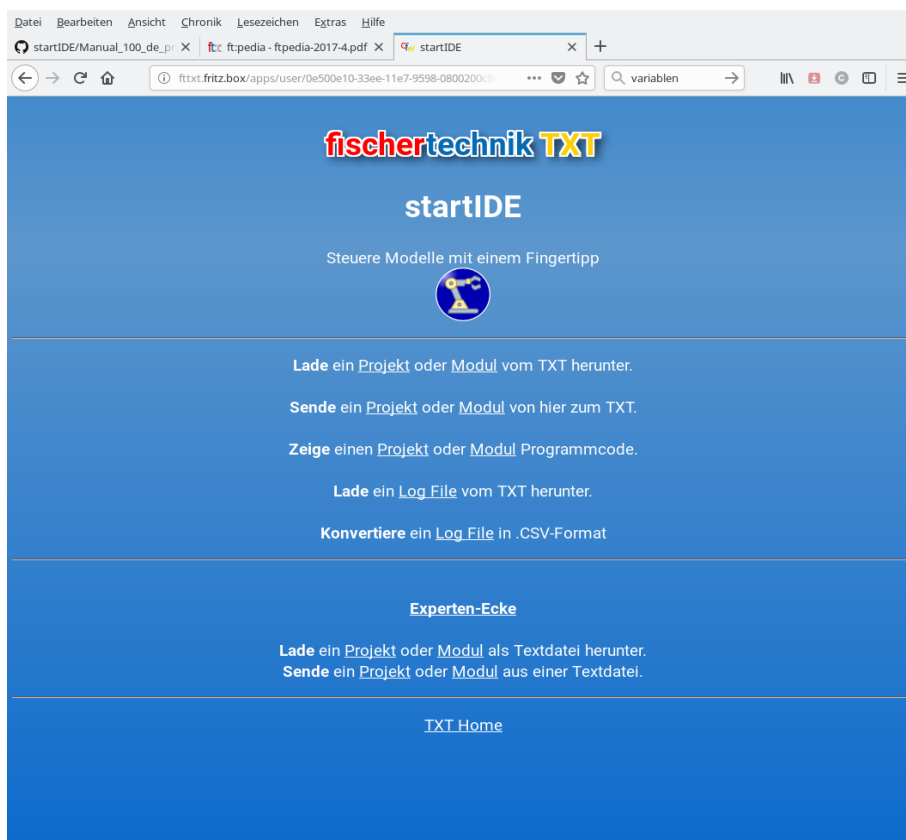
Um das Webinterface des TXT zu öffnen, ist die IP-Adresse des TXT in die Adresszeile des Webbrowsers einzutragen.



Mit einem Klick auf das startIDE-Symbol kommt man auf das startIDE-Webinterface:



Der Link „Open local application pages“ öffnet dann die Projektverwaltung:



Eine weitere Möglichkeit ist Secure Shell Fernzugriff:

Unter **Windows** gibt es dazu z.B. das Programm „puTTY“ (<http://www.putty.org/>) mit grafischer Benutzeroberfläche.

Unter **Linux** geschieht der Zugriff in einem Terminal-Fenster. Mit dem Befehl

```
ssh ftc@<IP-Adresse des TXT / TX-Pi>
```

loggt man sich auf dem Controller ein. Das Paßwort für den Benutzer ftc ist ebenfalls ftc.

Mit

```
cd apps/0e500e10-33ee-11e7-9598-0800200c9a66/
```

wechselt man in das startIDE-Verzeichnis. Nun kann man mit

```
ls projects/      bzw.      ls modules/
```

die gespeicherten Projekte bzw. Module auflisten lassen. Der ssh-Fernzugriff wird mit

```
exit
```

wieder beendet.

Wenn man den Namen des startIDE-Projekts, das man auf den PC übertragen möchte, ermittelt hat, kann man per Kopierbefehl

```
scp ftc@<IP-Adresse des TXT / TX-Pi>:apps/0e500e10-33ee-11e7-9598-0800200c9a66/projects/<Name des Projekts> <lokales Verzeichnis>/
```

das Projekt auf den PC kopieren.

Umgekehrt läßt sich mit

```
scp <lokales Verzeichnis>/<Name des Projekts>  
ftc@<IP-Adresse des TXT / TX-Pi>:apps/0e500e10-33ee-  
11e7-9598-0800200c9a66/projects/
```

ein Projekt wieder auf den Controller kopieren.

Das gleiche Vorgehen gilt analog für das Kopieren von Modulen, indem man in der Pfadangabe „projects“ durch „modules“ ersetzt.

## **Anhang**

## startIDE Befehlsreferenz

<b>Eingänge</b>	WaitInDig		<Gerät> <Port> <Bedingung> <Timeout>
	IfInDig		<Gerät> <Port> <Bedingung> <Sprungziel>
	WaitIn		<Gerät> <Port> <Eingangstyp> <Bedingung> <Vergl.-Wert> <Timeout>
	IfIn		<Gerät> <Port> <Eingangstyp> <Bedingung> <Vergl.-Wert> <Sprungziel>
	QueryIn		<Gerät> <Port> <Eingangstyp> <Text>
<b>Ausgänge</b>	Output		<Gerät> <Port> <Leistung>
	Motor		<Gerät> <Port> <Richtung> <Leistung>
	MotorP		<Gerät> <Port> <Endschalter> <Pulsgeber> <Richtung> <Leistung> <Pulszahl>
	MotorE		<Gerät> <Port> <Endschalter> <Richtung> <Leistung> <Pulszahl>
	MotorES		<Gerät> <Port> <Sync.Port> <Richtung> <Leistung> <Pulszahl>
<b>Variable</b>	Init		<Name> <Wert>
	<b>From...</b>	FromIn	<Gerät> <Port> <Eingangstyp> <Variable>
		FromKeypad	<Variable> <Min.-Wert> <Max.-Wert>
		FromDial	<Variable> <Min.-Wert> <Max.-Wert> <Text>
		FromButtons	<Variable> <Buttontext 1> ... <Buttontext 7>
	QueryVar		<Variable>
	IfVar		<Variable> <Bedingung> <Vergl.-Wert> <Sprungziel>
	Calc		<Zielvariable> <Operand 1> <Operator> <Operand 2>
<b>Steuerung</b>	# comment		
	Tag		<Sprungmarke>
	Jump		<Sprungziel>
	LoopTo		<Sprungmarke> <Anzahl>
	<b>Time</b>	Delay	<Wartezeit>
		TimerQuery	
		TimerClear	
		IfTimer	<After   Every> <Zeitdauer> <Sprungziel>
		Interrupt	<After   Every> <Zeitdauer> <Zielmodulname>
		QueryNow	
	Stop		
<b>Module</b>	Call		<Modulname> <Zähler>
	CallExt		<Modulname> <Zähler>
	Return		
	Module		<Name>
	MEnd		
<b>Interaktion</b>	Print		<Text>
	Clear		
	Message		<Text>'<Buttontext>
	Logfile		<0   1   C>

## Ablaufplan „Turm von Hanoi“



<b>A:</b> 2 auf greifen 1 auf 2 rechts 3 ab lösen	<b>B:</b> 2 auf 2 links 1 ab greifen 1 auf 1 rechts 2 ab lösen	<b>C:</b> 1 auf 1 rechts 1 ab greifen 2 auf 1 links 1 ab lösen	<b>D:</b> 1 auf 1 links 2 ab greifen 3 auf 2 rechts 3 ab lösen
<b>E:</b> 2 auf 1 links 1 ab greifen 1 auf 1 links 2 ab lösen	<b>F:</b> 1 auf 1 rechts 1 ab greifen 2 auf 1 rechts 1 ab lösen	<b>G:</b> 1 auf 2 links 2 ab greifen 3 auf 2 rechts 1 ab lösen 2 ab	

startIDE code listing:  
projects/c\_Hanoi3

=====

```

1:  # Hanoi
2:  #
3:  Output TXT 7 512
4:  Clear
5:  Print Fahre
   Referenzpunkte
   an...
6:  Call grundstellung 1
7:  Tag START
8:  Message Turm von
   Hanoi:<br>Bitte
   Tonnen in
   Startstellung
   bringen!'Start
9:  #
10: Clear
11: Print Programm
   laeuft...
12: # A
13: Print Schritt A
14: Call auf 2
15: Call greifen 1
16: Call auf 1
17: Call dreh_rechts 2
18: Call ab 3
19: Call loslassen 1
20: # B
21: Print Schritt B
22: Call auf 2
23: Call dreh_links 2
24: Call ab 1
25: Call greifen 1
26: Call auf 1
27: Call dreh_rechts 1
28: Call ab 2
29: Call loslassen 1
30: # C
31: Print Schritt C
32: Call auf 1
33: Call dreh_rechts 1
34: Call ab 1
35: Call greifen 1
36: Call auf 2
37: Call dreh_links 1
38: Call ab 1
39: Call loslassen 1
40: # D
41: Print Schritt D
42: Call auf 1
43: Call dreh_links 1
44: Call ab 2
45: Call greifen 1
46: Call auf 3
47: Call dreh_rechts 2
48: Call ab 3
49: Call loslassen 1
50: # E

```

```

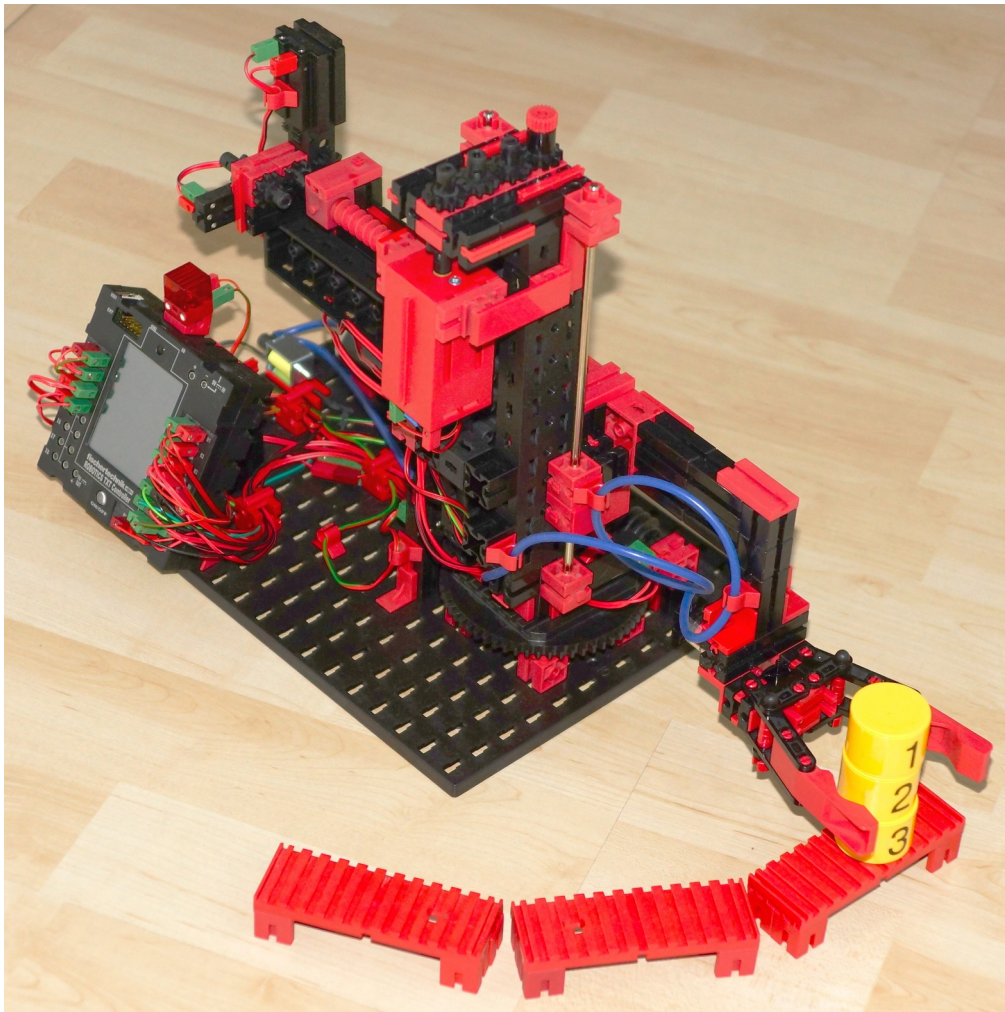
51: Print Schritt E
52: Call auf 2
53: Call dreh_links 1
54: Call ab 1
55: Call greifen 1
56: Call auf 1
57: Call dreh_links 1
58: Call ab 2
59: Call loslassen 1
60: # F
61: Print Schritt F
62: Call auf 1
63: Call dreh_rechts 1
64: Call ab 1
65: Call greifen 1
66: Call auf 2
67: Call dreh_rechts 1
68: Call ab 1
69: Call loslassen 1
70: # G
71: Print Schritt G
72: Call auf 1
73: Call dreh_links 2
74: Call ab 2
75: Call greifen 1
76: Call auf 3
77: Call dreh_rechts 2
78: Call ab 1
79: Call loslassen 1
80: #
81: Call ab 3
82: Call freiheben 1
83: Message Ablauf
   beendet.'Okay
84: Print ...und
   zurueck!
85: Call greifen 1
86: Call auf 1
87: Call dreh_links 2
88: Call ab 1
89: Call loslassen 1
90: #
91: Print
   Hoehen
   referenzierung
92: Call g2 1
93: Jump START
94: #
95: # Ende
96: #
97: Module
   grundstellung
98: Call loslassen
99: MotorP TXT 3 2 3 l
   512 9999
100: MotorE TXT 2 4 1
   512 9999

```

```

101: MotorE TXT 2 4 r
   512 1150
102: MotorE TXT 1 1 l
   512 9999
103: Call g2 1
104: MEnd
105: Module g2
106: MotorE TXT 2 4 l
   512 9999
107: Call freiheben 1
108: MEnd
109: # Arm
110: Module vor
111: MotorP TXT 3 2 3 r
   512 10
112: MEnd
113: Module zurueck
114: MotorP TXT 3 2 3 r
   512 10
115: MEnd
116: # Heben
117: Module ab
118: MotorE TXT 2 4 l
   512 350
119: MEnd
120: Module auf
121: MotorE TXT 2 4 r
   512 350
122: MEnd
123: Module freiheben
124: MotorE TXT 2 4 r
   512 85
125: MEnd
126: # Drehen
127: Module dreh_rechts
128: MotorE TXT 1 1 r
   512 375
129: MEnd
130: Module dreh_links
131: MotorE TXT 1 1 l
   512 375
132: MEnd
133: # Greifer
134: Module loslassen
135: Output TXT 8 0
136: Delay 100
137: MEnd
138: Module greifen
139: Output TXT 8 512
140: Delay 100
141: MEnd

```



## **I/O Belegungsplan des Hanoi-Roboters:**

### **Achse 1: Drehachse**

**Encodermotor an Ausgang M1 und Zähler C1, Endschalter I1  
Endschalter in Drehrichtung links.**

### **Achse 2: Hochachse**

**Encodermotor an Ausgang M2 und Zähler C2, Endschalter I4  
Endschalter in Drehrichtung links → unten.**

### **Achse 3: Längsachse**

**S-Motor an Ausgang M3 mit Pulsrad an I3 und Endschalter I2  
Endschalter in Drehrichtung links → hinten.**

**Magnetventil für Greifer und Kompressor parallel an O8  
Blink-LED an O7**