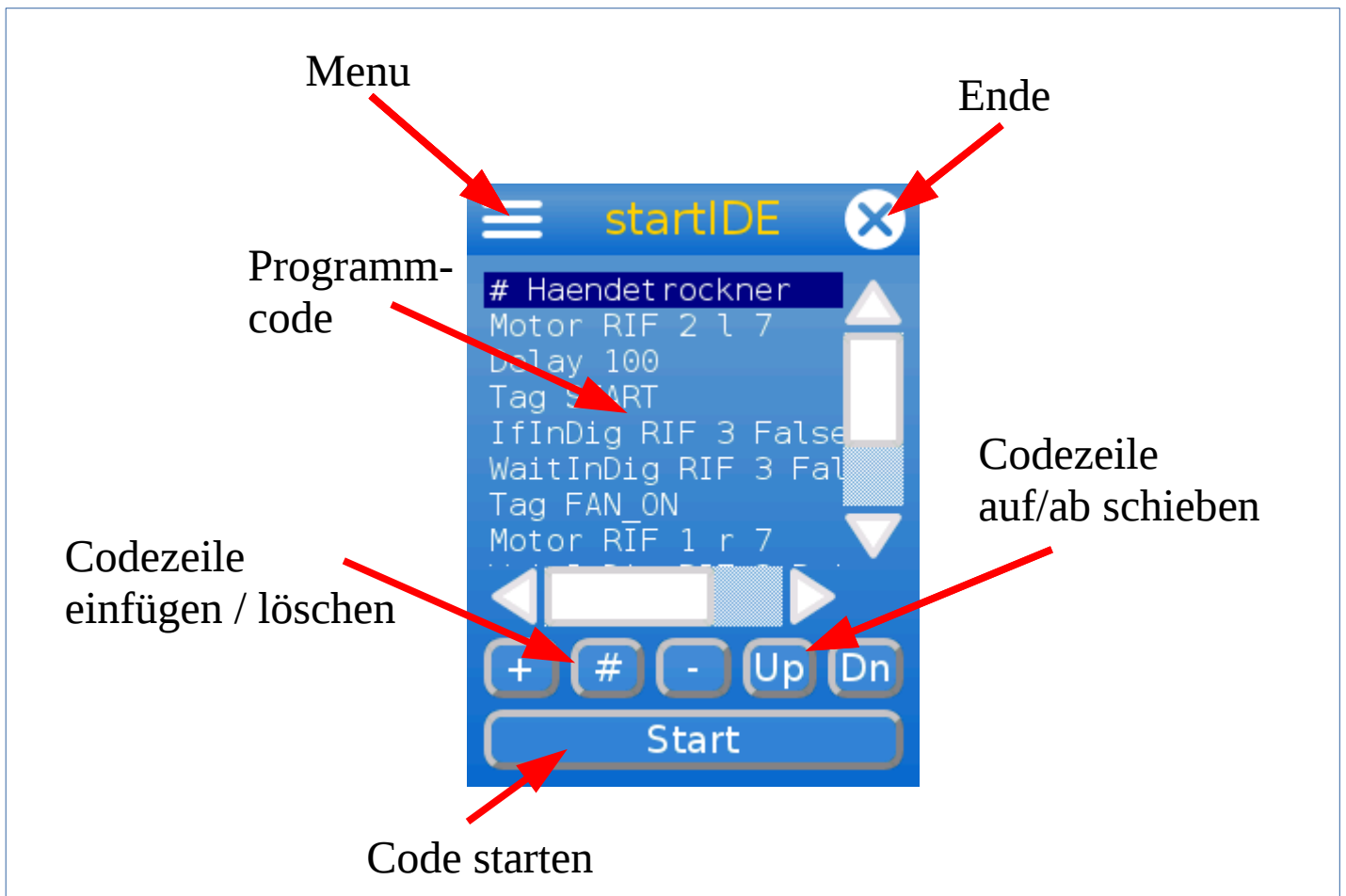


startIDE

v0.54

Schnelleinstieg

Handbuch 2017/09/15



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

startIDE © 2017 Peter Habermehl, peter.habermehl@gmx.de

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

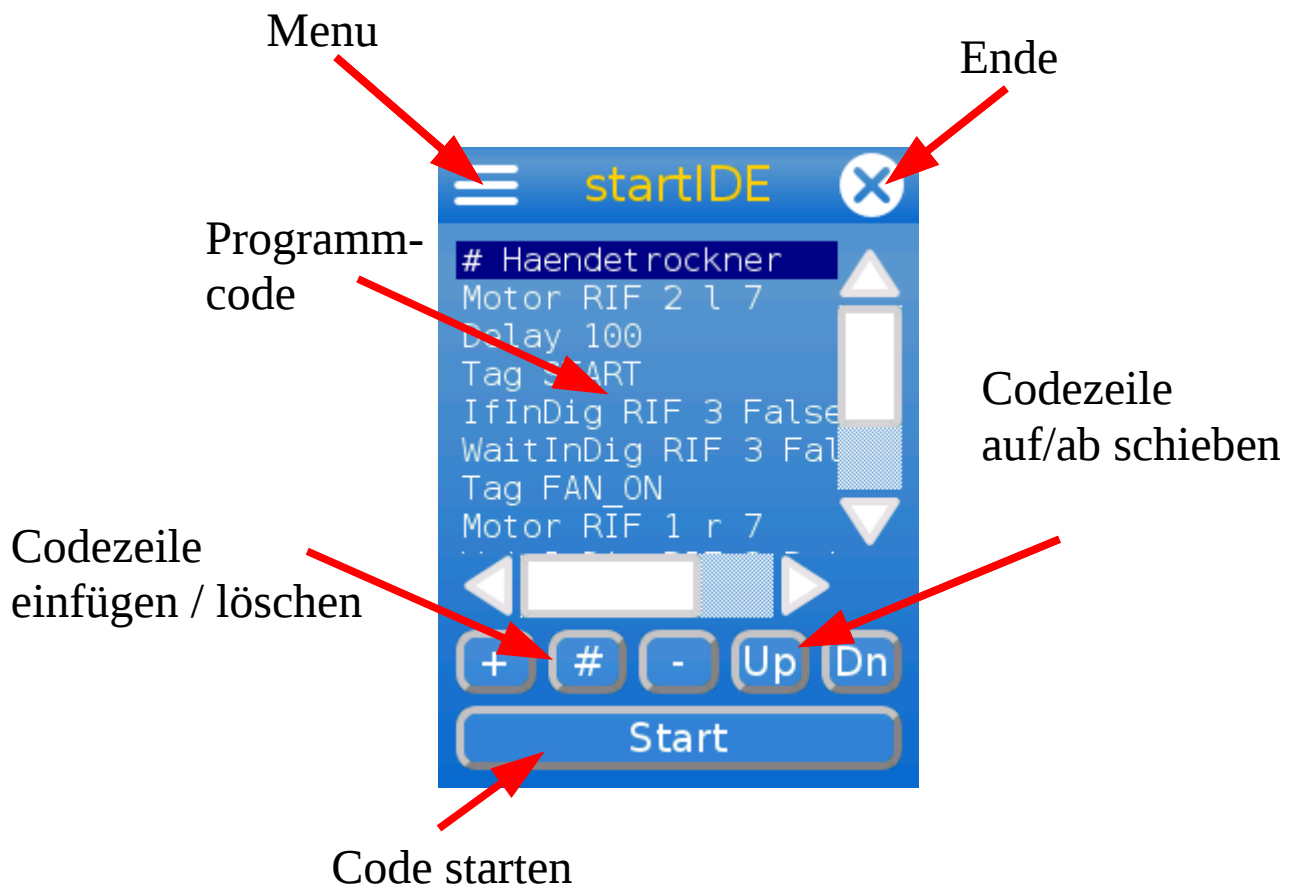
Besonderer Dank geht an **Till Harbaum** für den TX-Pi <https://github.com/harbaum/tx-pi> und seine Motivationsmaßnahmen. Weiterhin seien erwähnt **Richard Kunze**, **Raphael Jacob** und **Esther Mietzsch** vornehmlich für die Arbeit an der community firmware und **Torsten Stuehn** für ftrobopy <https://github.com/ftrobopy/ftrobopy> zur TXT-Programmierung sowie **Erik Andresen** für die libroboint <https://github.com/nxdefiant/libroboint> zum Zugriff auf die Robo Interfaces.

Inhaltsverzeichnis

1. Der Hauptbildschirm.....	5
2. Das Menu.....	6
3. Funktionsübersicht.....	8
3.1. Inputs.....	8
3.1.1. WaitForInputDig.....	9
3.1.2. IfInputDig.....	10
3.2. Outputs.....	11
3.2.1. Output.....	11
3.2.2. Motor.....	12
3.2.3. MotorPulsewheel.....	13
3.2.4. MotorEncoder.....	14
3.2.5. MotorEncoderSync.....	15
3.3. Controls.....	17
3.3.1. # comment.....	17
3.3.2. Tag.....	17
3.3.3. Jump.....	18
3.3.4. LoopTo.....	19
3.3.5. Delay.....	21
3.3.6. Stop.....	21
3.4. Modules.....	22
3.4.1. Call.....	23
3.4.2. Return.....	23
3.4.3. Module.....	23
3.4.4. MEnd.....	24
3.5. Interaction.....	25
3.5.1. Print.....	25
3.5.2. Clear.....	26

3.5.3. Message.....	26
3.5.4. Request.....	27
4. Beispiele.....	28
4.1. Mitgelieferte startIDE-Projekte.....	28
4.1.1. c_Ampel.....	28
4.1.2. c_Blink.....	29
4.1.3. c_Haendetrockner.....	29
4.1.4. c_Lauflicht und c_Signalfeuer.....	29
4.2. Programmieraufgaben.....	30
4.1.1. Hallo Welt 1.....	30
4.1.2. Hallo Welt 2.....	30
4.1.3. Hallo Welt 3.....	30
4.1.4. Gottes Rache.....	31
4.1.5. Es regnet immer noch.....	34
5. Tips und Tricks.....	35
5.1. Programmierhinweise.....	35
5.2. Kopieren von Programmteilen.....	36
5.3. Debugging / Fehlersuche.....	36
5.4. startIDE-Projekte auf PC übertragen.....	37
Anhang.....	39
Befehlsreferenz.....	40
Projekt „Turm von Hanoi“.....	41

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 **Rauten**- („#“) Knopf dupliziert die aktuelle Befehlszeile.

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) aktuell gefunden wurden.

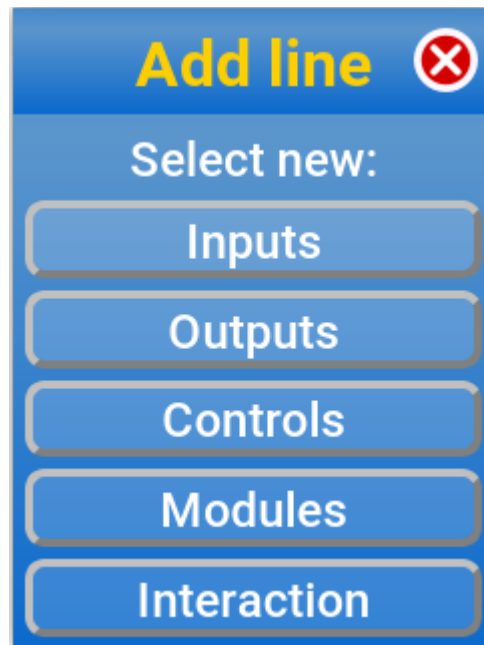
Wurde ein externes Interface nach dem Start von startIDE angeschlossen, so muß dieser Menüpunkt aufgerufen werden, um das Interface in startIDE nutzen zu können.

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.

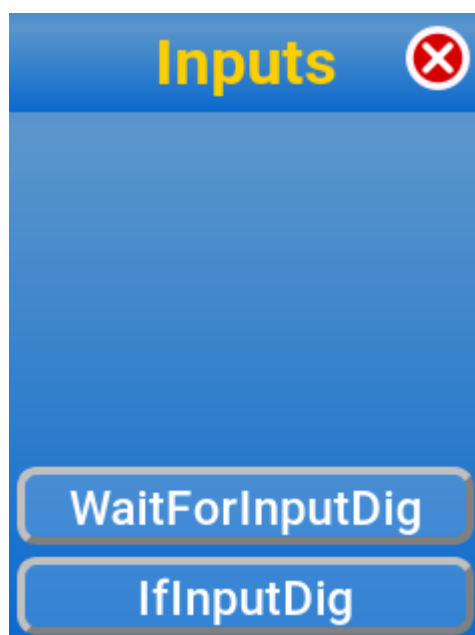
3. Funktionsübersicht

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



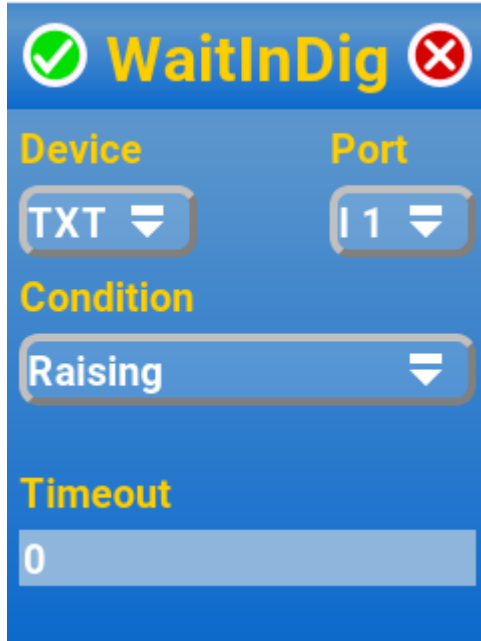
Mit dem Kreuz-Knopf oben rechts kann die Auswahl abgebrochen werden, ohne eine neue Code-Zeile einzufügen.

3.1. Inputs



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. 3.3.2., 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

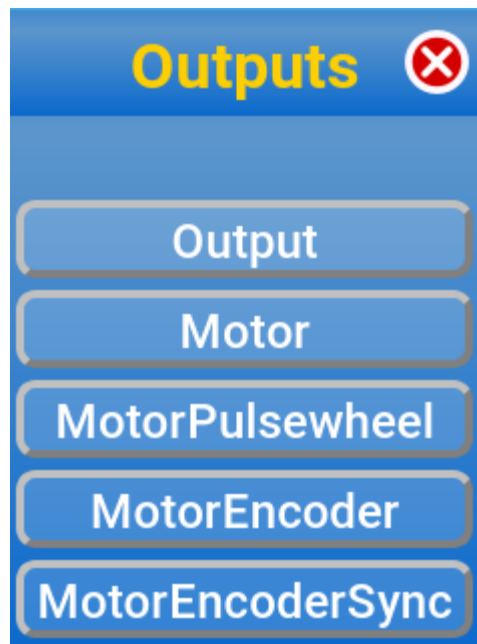
```
IfInDig <Device> <Port> <Condition> <Jump Tag>
```

also z.B.

```
IfInDig TXT 2 False ende
```

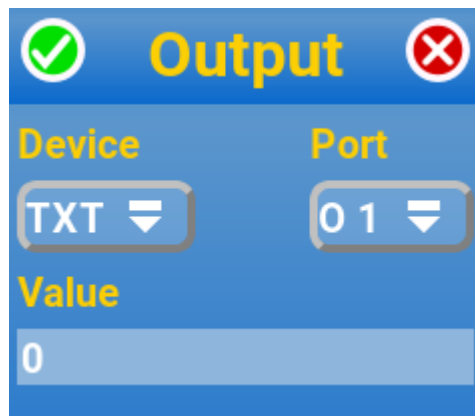
Springt zur Marke „ende“, wenn I2 am TXT Falsch (also Kontakt geöffnet) ist. Andernfalls wird mit der direkt folgenden Programmzeile fortgefahren.

3.2. Outputs



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> <Value> <Direction>
```

also z.B.

```
Motor TXT 1 512 r
```

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>  
      - 13 -
```

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>
```

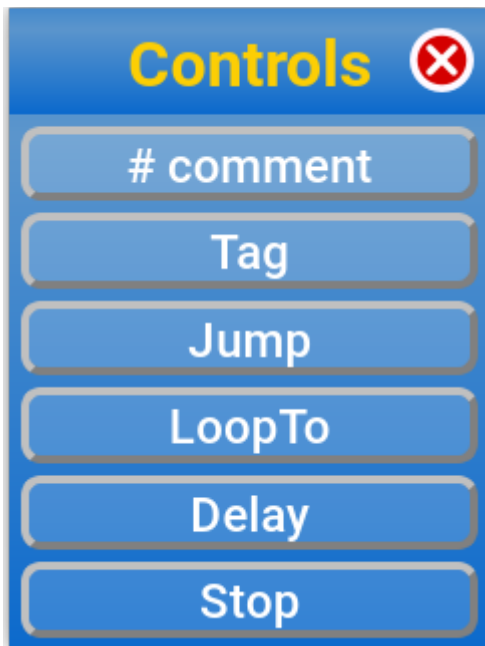
also z.B.

```
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. Controls

Hier sind die Funktionen



zur Programmablaufsteuerung zu finden.

3.3.1. # comment

Hiermit wird ein Kommentartext in den Programmcode eingefügt.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
# <Kommentar>
```

also z.B.

```
# Hier startet das Hauptprogramm
```

3.3.2. 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 <Jump Tag>“ wird die Programmausführung bei der Zeile „Tag <Jump Tag>“ fortgeführt.

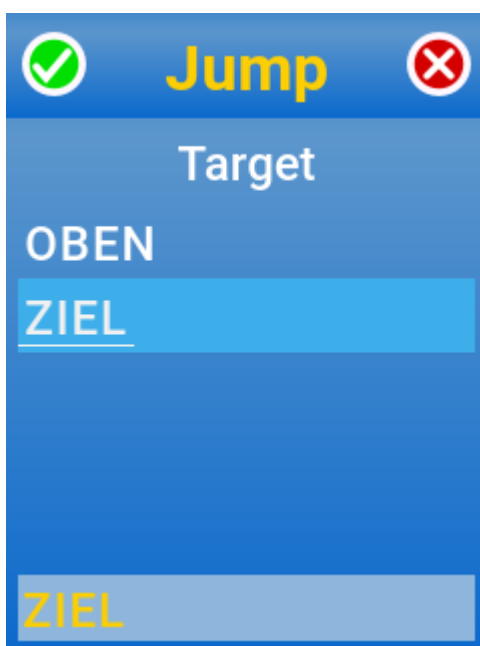
Im Codeabschnitt des Hauptfensters ist die Syntax

Tag <Jump Tag>

also z.B.

Tag START

3.3.3. 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.

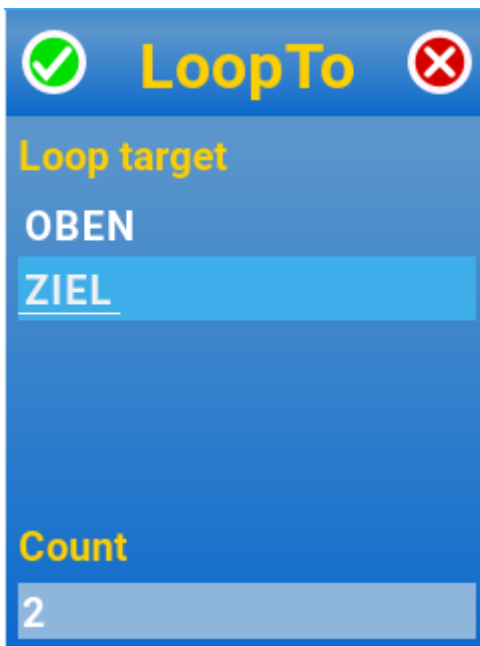
Jump START

3.3.4. 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

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.

3.3.5. 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.

3.3.6. 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
```

3.4. Modules

Module sind in sich geschlossene Programmblöcke.

Beispiel:

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

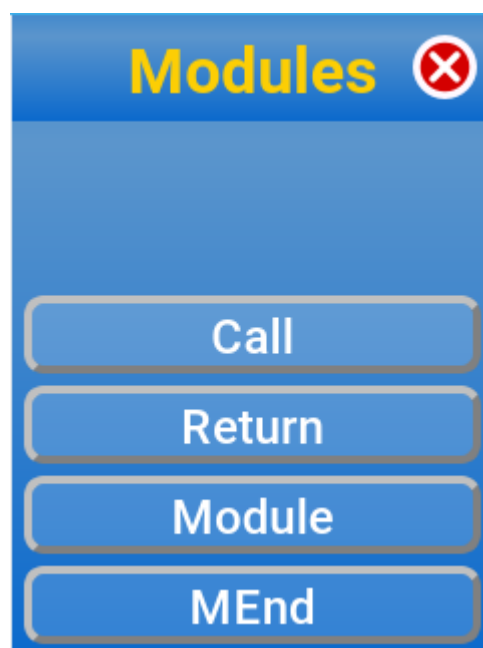
Dieses Modul mit dem Namen „LAMPEN_AN“ würde bei seiner Ausführung die Ausgänge 1 und 2 an einem Robo Interface auf Stufe 7 einschalten.

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.

Es gibt folgende Modul-Funktionen:



3.4.1. Call

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

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Call <Modulname>
```

also z.B.

```
Call LAMPEN_AN
```

Hier würde ein Modul mit dem Namen „LAMPEN_AN“ aufgerufen.

3.4.2. Return

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

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Return
```

3.4.3. 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
```

3.4.4. 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

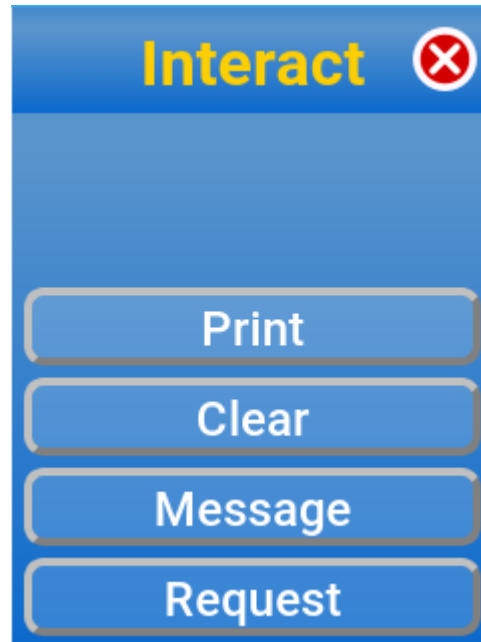
```
MEnd
```

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
```


3.5. Interaction

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



3.5.1. 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.

3.5.2. Clear

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

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Clear
```

3.5.3. 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.

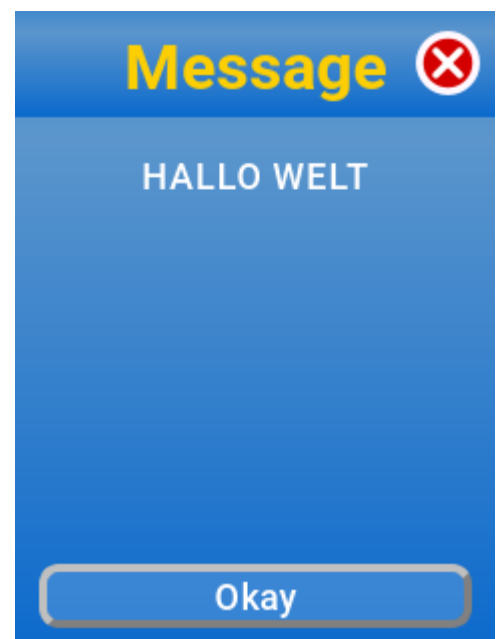
Im Codeabschnitt des Hauptfensters ist die Syntax

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

also z.B.

```
Message HALLO WELT'Okay
```

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



3.5.4. Request

Der Request-Befehl ist noch nicht implementiert.

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

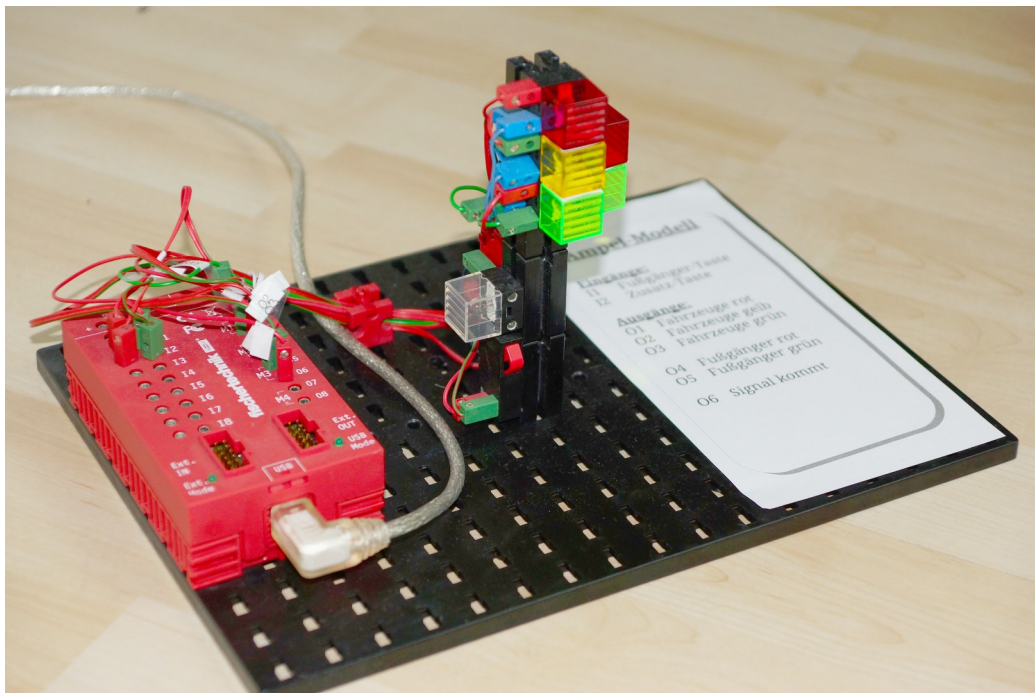
Dies ist eine einfache Fußgängerampel-Steuerung. Das Programm ist für die Robo Interface Familie (RIF) ausgelegt.

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.

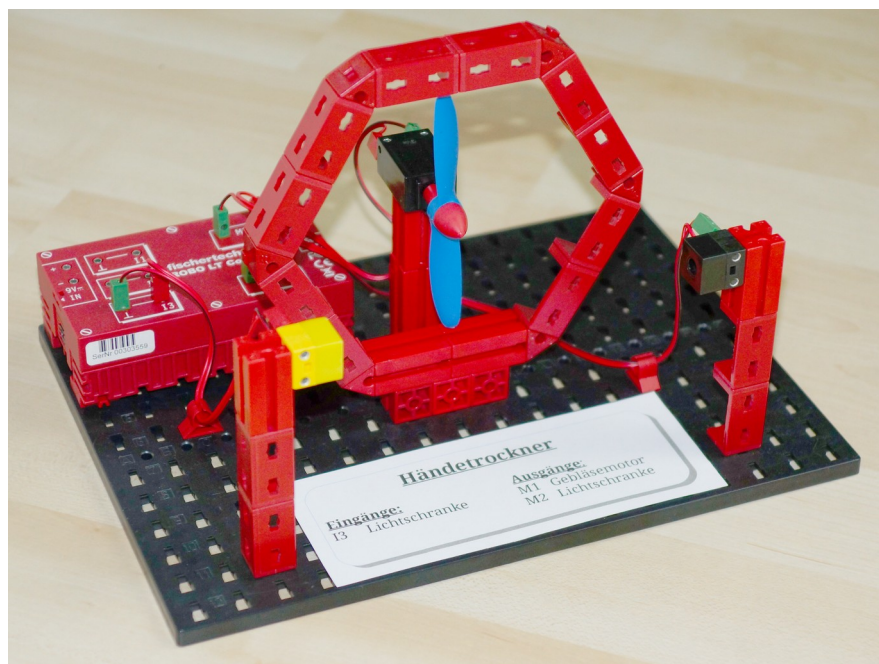
4.1.3. c_Haendetrockner

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
Call KURZ
Call KURZ
Call LANG
Call LANG
Call LANG
Call KURZ
Call KURZ
Call KURZ
# 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
Call KURZ
Call KURZ
Call LANG
Call LANG
Call LANG
Call KURZ
Call KURZ
Call KURZ
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

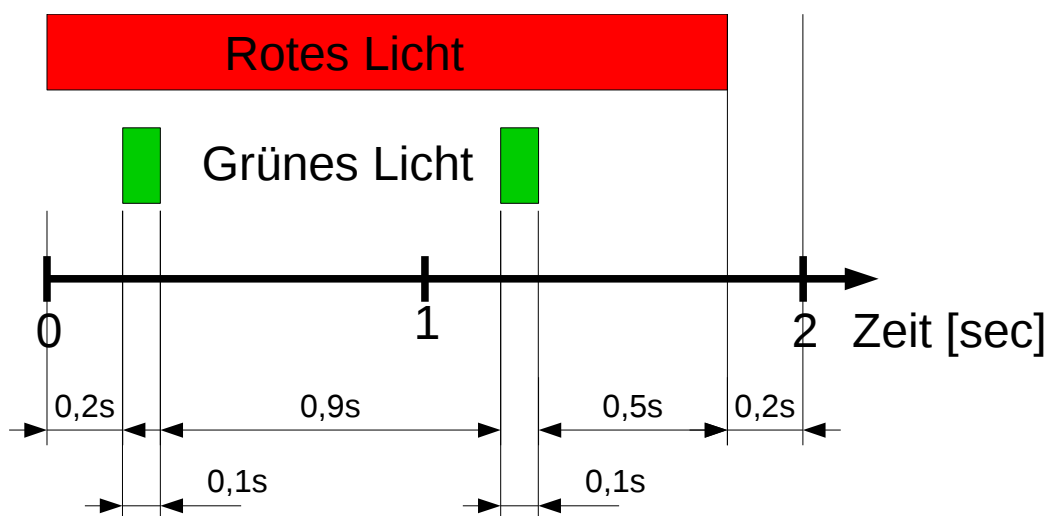
Es ist sinnvoll, vor Beginn der Programmerstellung einen Ablaufplan auf Papier zu skizzieren.

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

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.

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 an 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.

5.4. startIDE-Projekte auf PC übertragen

Wenn der Controller mit einem Netzwerk verbunden ist, so kann man per Secure Shell Fernzugriff erlangen.

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, daß 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.

Für die Zukunft ist geplant, startIDE mit einem Webinterface zu versehen, über das dann die Dateiübertragung geschehen kann.

Anhang

Befehlsreferenz

[illegible]

Ablaufplan „Turm von Hanoi“



- A:**
- 2 auf greifen
 - 1 auf
 - 2 rechts
 - 3 ab lösen
- B:**
- 2 auf
 - 2 links
 - 1 ab greifen
 - 1 auf
 - 1 rechts
 - 2 ab lösen
- C:**
- 1 auf
 - 1 rechts
 - 1 ab greifen
 - 2 auf
 - 1 links
 - 1 ab lösen
- D:**
- 1 auf
 - 1 links
 - 2 ab greifen
 - 3 auf
 - 2 rechts
 - 3 ab lösen
- E:**
- 2 auf
 - 1 links
 - 1 ab greifen
 - 1 auf
 - 1 links
 - 2 ab lösen
- F:**
- 1 auf
 - 1 rechts
 - 1 ab greifen
 - 2 auf
 - 1 rechts
 - 1 ab lösen
- G:**
- 1 auf
 - 2 links
 - 2 ab greifen
 - 3 auf
 - 2 rechts
 - 1 ab lösen
 - 2 ab

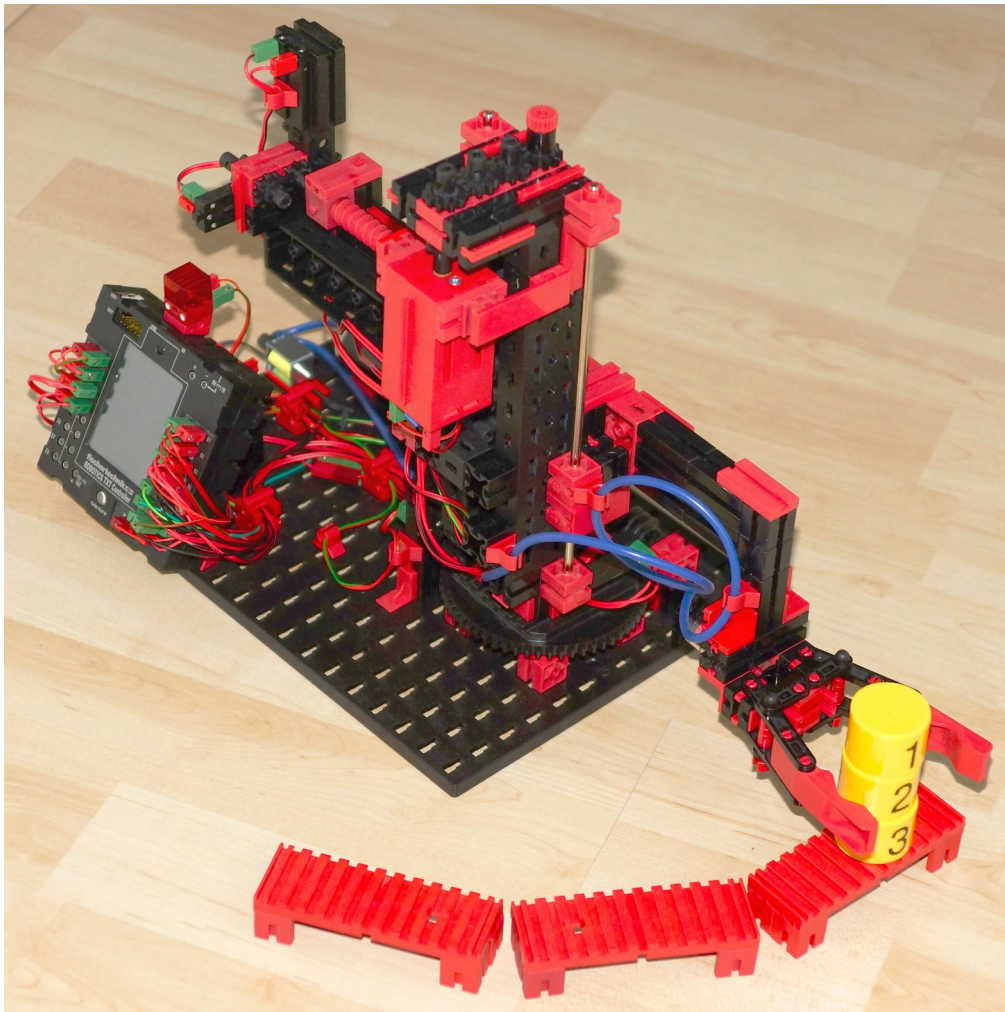
```

1  # Hanoi
2  #
3  Output TXT 7 512
4  Clear
5  Print Fahre Referenzpunkte an...
6  Call grundstellung
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 auf2
15 Call greifen
16 Call auf
17 Call dreh_rechts2
18 Call ab3
19 Call loslassen
20 # B
21 Print Schritt B
22 Call auf2
23 Call dreh_links2
24 Call ab
25 Call greifen
26 Call auf
27 Call dreh_rechts
28 Call ab2
29 Call loslassen
30 # C
31 Print Schritt C
32 Call auf
33 Call dreh_rechts
34 Call ab
35 Call greifen
36 Call auf2
37 Call dreh_links
38 Call ab
39 Call loslassen
40 # D
41 Print Schritt D
42 Call auf
43 Call dreh_links
44 Call ab2
45 Call greifen
46 Call auf3
47 Call dreh_rechts2
48 Call ab3
49 Call loslassen
50 # E
51 Print Schritt E
52 Call auf2
53 Call dreh_links
54 Call ab
55 Call greifen
56 Call auf
57 Call dreh_links
58 Call ab2
59 Call loslassen
60 # F
61 Print Schritt F
62 Call auf
63 Call dreh_rechts
64 Call ab
65 Call greifen
66 Call auf2
67 Call dreh_rechts
68 Call ab
69 Call loslassen
70 # G
71 Print Schritt G
72 Call auf
73 Call dreh_links2
74 Call ab2
75 Call greifen
76 Call auf3
77 Call dreh_rechts2
78 Call ab
79 Call loslassen
80 #

81 Call ab3
82 Call freiheben
83 Message Ablauf beendet.'Okay
84 Print ...und zurueck!
85 Call greifen
86 Call auf
87 Call dreh_links2
88 Call ab

89 Call loslassen
90 #
91 Print Hoehenreferenzierung
92 Call g2
93 Jump START
94 #
95 # Ende
96 #
97 Module grundstellung
98 Call loslassen
99 MotorP TXT 3 2 3 1 512 9999
100 MotorE TXT 2 4 1 512 9999
101 MotorE TXT 2 4 r 512 1150
102 MotorE TXT 1 1 1 512 9999
103 Call g2
104 MEnd
105 Module g2
106 MotorE TXT 2 4 1 512 9999
107 Call freiheben
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 1 512 350
119 MEnd
120 Module ab2
121 MotorE TXT 2 4 1 512 700
122 MEnd
123 Module ab3
124 MotorE TXT 2 4 1 512 1050
125 MEnd
126 Module auf
127 MotorE TXT 2 4 r 512 350
128 MEnd
129 Module auf2
130 MotorE TXT 2 4 r 512 700
131 MEnd
132 Module auf3
133 MotorE TXT 2 4 r 512 1050
134 MEnd
135 Module freiheben
136 MotorE TXT 2 4 r 512 85
137 MEnd
138 # Drehen
139 Module dreh_rechts
140 MotorE TXT 1 1 r 512 375
141 MEnd
142 Module dreh_rechts2
143 MotorE TXT 1 1 r 512 750
144 MEnd
145 Module dreh_links
146 MotorE TXT 1 1 1 512 375
147 MEnd
148 Module dreh_links2
149 MotorE TXT 1 1 1 512 750
150 MEnd
151 # Greifer
152 Module loslassen
153 Output TXT 8 0
154 Delay 100
155 MEnd
156 Module greifen
157 Output TXT 8 512
158 Delay 100
159 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**