

startIDE

v0.5

Schnelleinstieg

Handbuch 2017/09/02

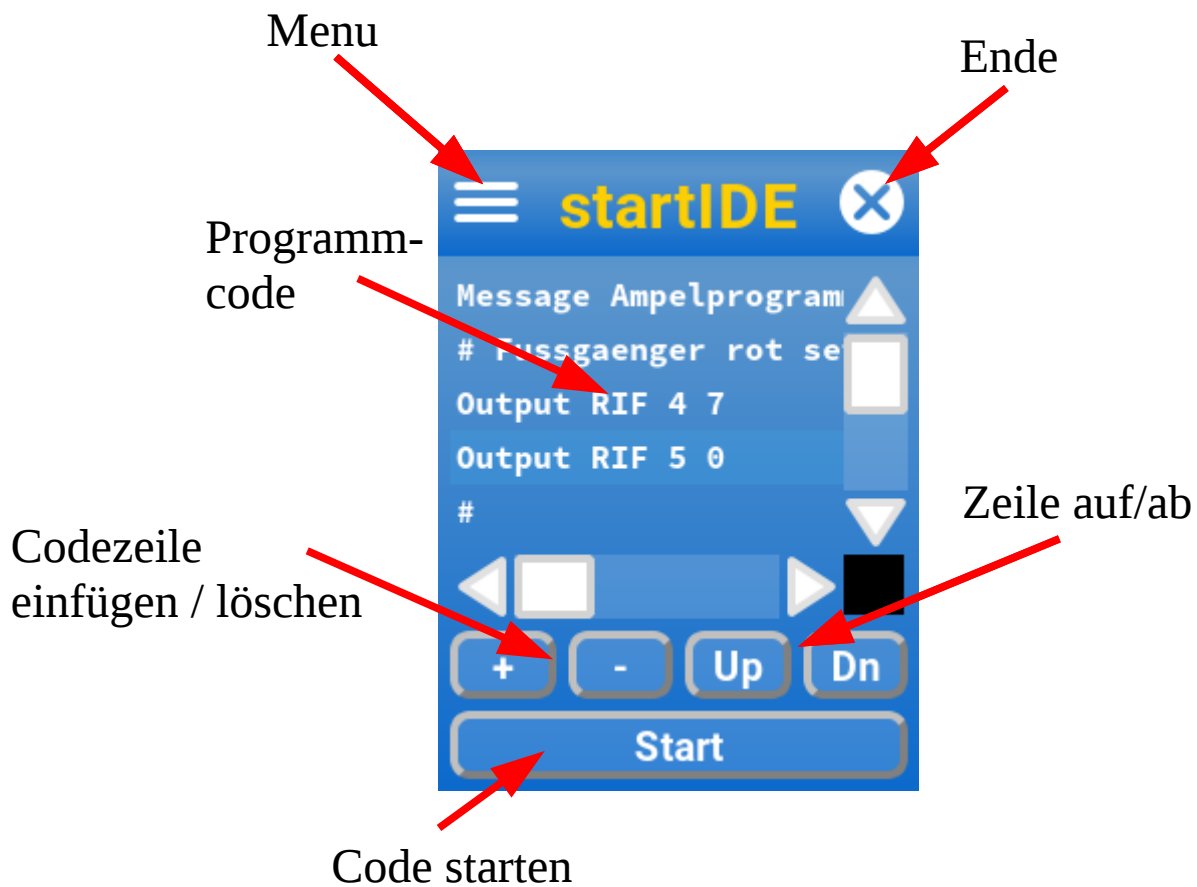


Abb. 1

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.

Inhalt:

1. Der Hauptbildschirm
2. Das Menu
3. Funktionsübersicht
 - 3.1. Inputs
 - 3.2. Outputs
 - 3.3. Controls
 - 3.4. Modules
 - 3.5. Interaction
4. Beispiele
5. Tips und Tricks
 - 5.1. Kopieren von Programmteilen
 - 5.2. Debugging / Fehlersuche

Anhang:

Befehlsreferenz-Karte

startIDE © 2007 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.

1. Der Hauptbildschirm

Die wesentlichen Elemente des Hauptbildschirms sind in **Abb. 1** 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.

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** wird u.a. 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.

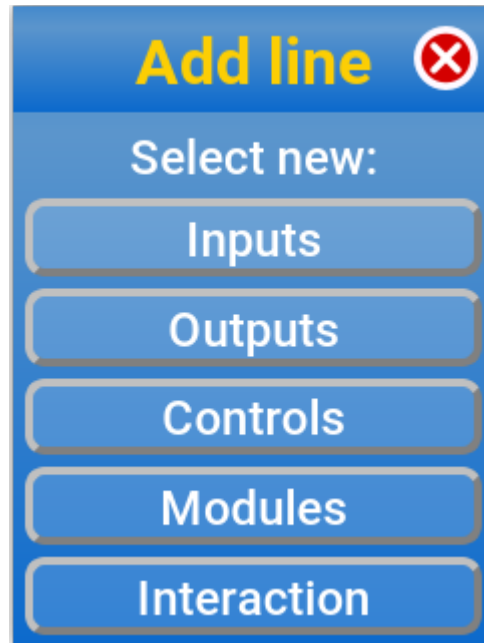
Wurde ein externes Interface nach dem Start von startIDE angeschlossen, so muß dieser Menüpunkt aufgerufen werden, um das Interface in startIDE nutzen zu können.

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.

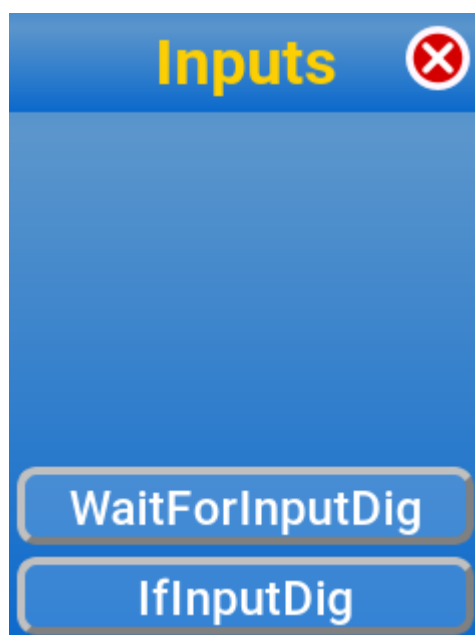
3. Funktionsübersicht

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



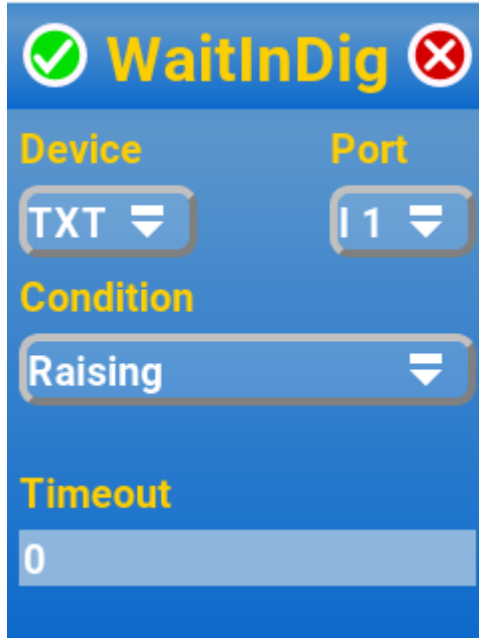
Mit dem Kreuz-Knopf oben rechts kann die Auswahl abgebrochen werden, ohne eine neue Code-Zeile einzufügen.

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

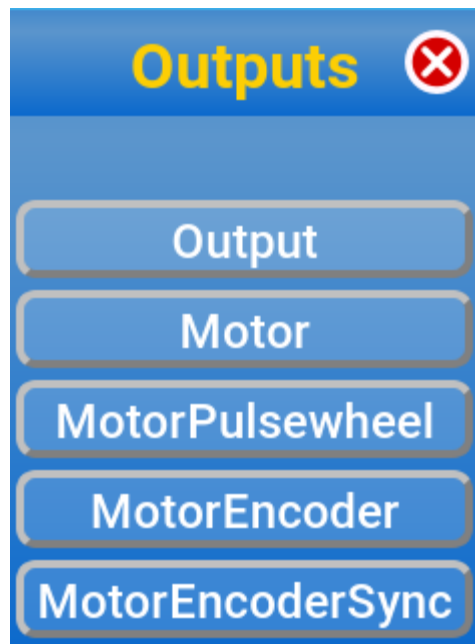
```
IfInDig <Device> <Port> <Condition> <Jump Tag>
```

also z.B.

```
IfInDig TXT 2 False ende
```

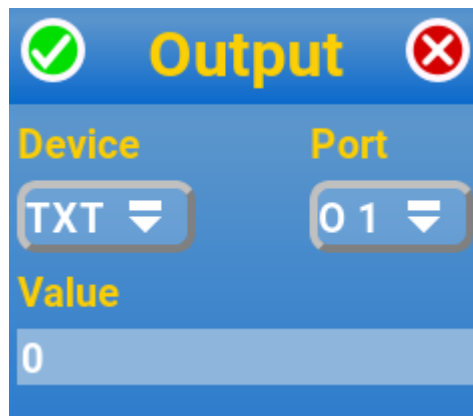
Springt zur Marke „ende“, wenn I2 am TXT Falsch (also Kontakt geöffnet) ist. Andernfalls wird mit der direkt folgenden Programmzeile fortgefahren.

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>
```

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>
```

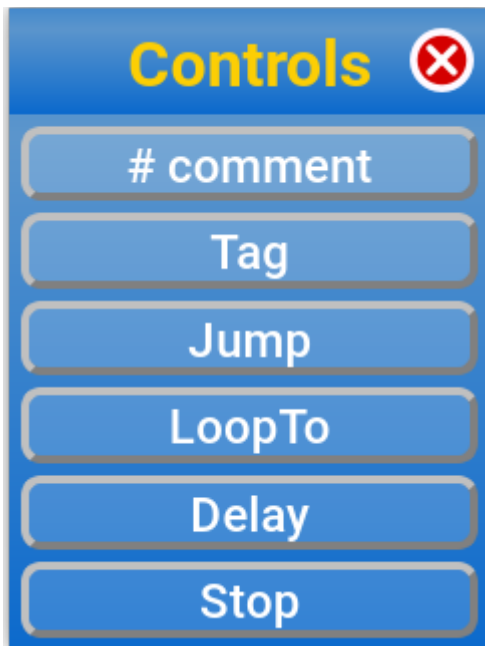
also z.B.

```
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 altenm 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.

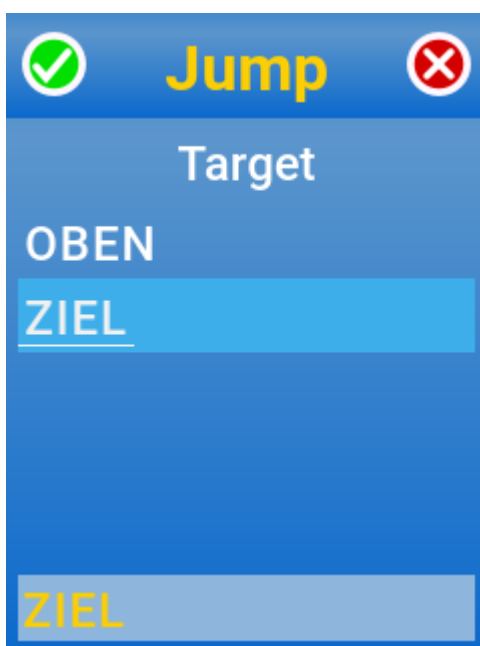
Im Codeabschnitt des Hauptfensters ist die Syntax

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.

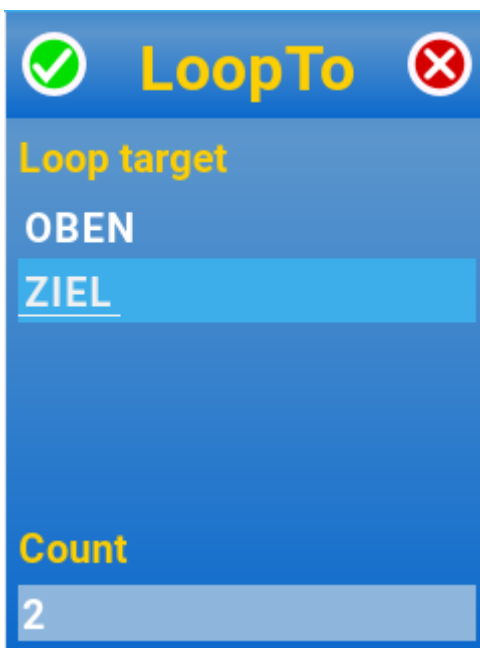
Jump START

3.3.3. 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.

3.3.3. 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.

Im Codeabschnitt des Hauptfensters ist die Syntax

```
Delay <Zeit in ms>
```

also z.B.

```
Delay 1500
```

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

3.3.4. 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
```

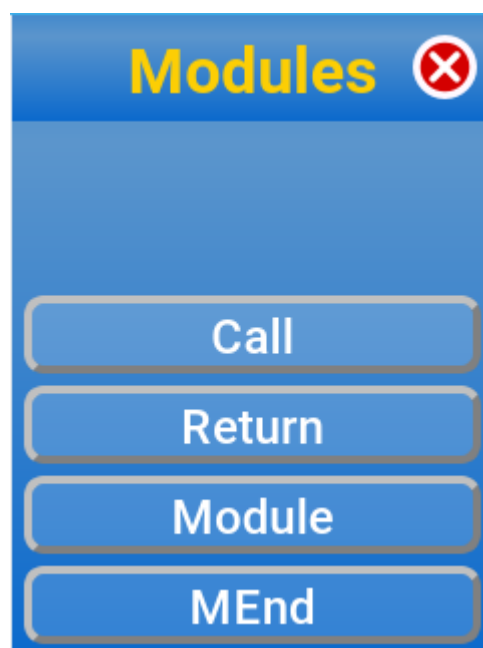
Dieses Modul mit dem Namen „LAMPEN_AN“ würde bei seiner Ausführung die Ausgänge 1 und 2 an einem Robo Interface auf Stufe 7 einschalten.

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
```

3.4.3. 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

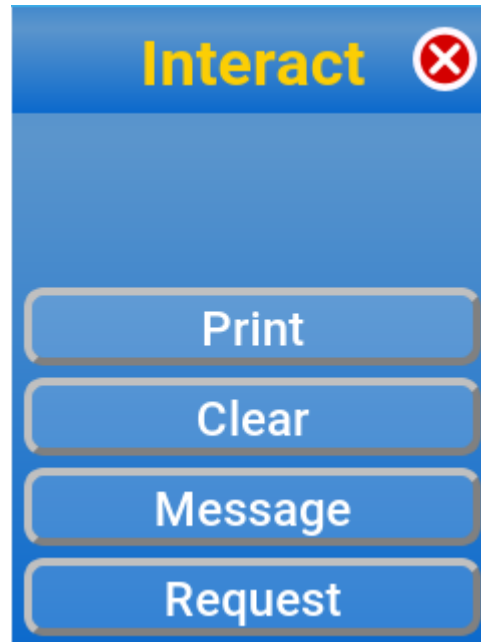
MEnd

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.

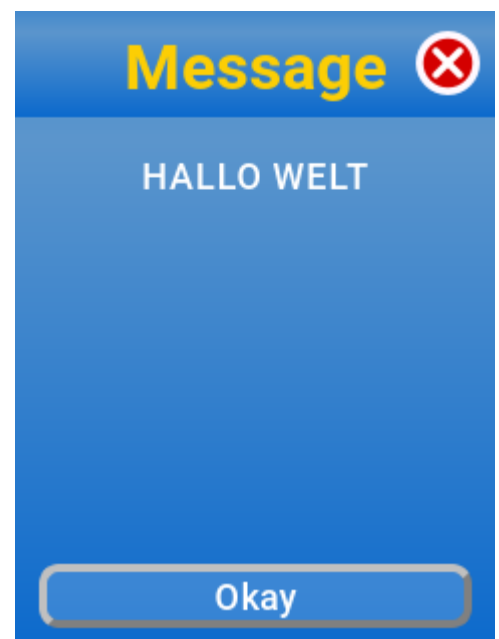
Im Codeabschnitt des Hauptfensters ist die Syntax

```
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. 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 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
```

5. Tips und Tricks

5.1. 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 sinnvollerweise wieder zu löschen.

5.2. 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.

Befehlsreferenz

Group	Function	0	Parameters		1	2	3	4	5	6	7	Impl. RIF	Impl. TXT
Output	Output	Motor			<interface>	<channel>	<v value>						
					<interface>	<channel>	<direction>	<speed>				X	X
	MotorP	MotorE	MotorES		<interface>	<channel>	<end switch input>	<pulse input>	<direction>	<speed>	<pulses>	X	X
					<interface>	<channel>	<end switch input>	<direction>	<speed>	<pulses>		...	X
Input	WaitIntDig	IfInDig			<interface>	<dig. Input>	<Raising Falling>	<timeout>				X	X
					<interface>	<dig. Input>	<True False>	<label>				X	X
	Control	#	Tag	Jump									
{	LoopTo	Delay	Stop		<label>	<times>							
	Module	Call	Return		<name>								
Interaction	Print	Clear	Message	Request	<text>	<button 1>	<button 2>	<jump target 1>	<jump target 2>				
					<text>	<button 1>	<button 2>	<jump target 1>	<jump target 2>				

Debugging

TRACEON
TRACEOFF
STEPON
STEPOFF

Code lines will be printed on the log screen during execution of the code
Ends code line printing
Code will be executed step by step. After each code line, tap on the output screen to execute the next line.
Ends step-by-step execution