

Inhalt

Variablen (Integer 16bit)

defvar definiert eine Variable
setvar weist einer Variable einen Wert zu
addvar addiert einen Wert zu einer Variable
subvar subtrahiert einen Wert von einer Variable
calc führt eine mathematische Funktion aus

Strings (Zeichen)

defstr definiert einen String
setstr weist einem String Zeichen zu
addstr fügt Zeichen an einen String an
strtovar wandelt ASCII-Werte aus Strings in Variablen
vartostr wandelt Variablen in Strings
getstrlen gibt die Länge eines Strings aus
find sucht nach Zeichen in einem String
bytetostr schreibt ein Zeichen in einen String
delstrpos löscht eine Stelle in einem String

Timer, Delay

retjumptimer1 definiert und aktiviert den Timer
settimer setzt das Timerintervall (Zeit)
timerstop stoppt den Timer
timergo startet den Timer nach timerstop
delay wartet für eine bestimmte Zeit

I/O's, Taster, Leds

din definiert einen Port als Input
dout definiert einen Port als Output
on schaltet einen Port/Led ein. Setzt einen Pullup Widerstand
off schaltet einen Port/Led aus. Löscht einen Pullup Widerstand
portin gibt den Status eines Ports zurück
blk lässt einen Port/Led blinken
getblk gibt die verbleibene Zahl von blinken zurück
adcin gibt den Wert der Spannung am Analogeingang zurück

Verzweigungen Programmsprünge

| | |
|------------|---|
| ::Marke | definiert eine Sprungmarke |
| jump | führt einen Programmsprung aus |
| retjump | springt in eine Unterroutine |
| return | beendet eine Unterroutine |
| if | leitet eine Bedingung ein |
| else | unterteilt if und endif (optional) |
| endif | beendet die if-Anweisung |
| press | fragt eine Taste/Port ab |
| holdpress | wartet bis eine Taste/Port freigegeben wird |
| endpress | beendet die press-Anweisung |
| push | reagiert auf einen Tastendruck |
| endpush | beendet die push-Anweisung |
| release | reagiert, wenn eine Taste losgelassen wird |
| endrelease | beendet die release-Anweisung |
| count | leitet eine Zählerschleife ein |
| endcount | beendet die count-Anweisung |

Schnittstellenfunktionen

| | |
|--------------------|---|
| setbaud | setzt die Baudrate für die RS232-Schnittstelle |
| setsetting | setzt die Settings für die RS232-Schnittstelle |
| send | sendet Daten auf die RS232-Schnittstelle |
| strtobuffer | definiert den Inputbuffer für die RS232-Schnittstelle |
| setwgpbaud | setzt die Baudrate für die WgP-Schnittstelle |
| setwgpsetting | setzt die Settings für die WgP-Schnittstelle |
| wgpsend | sendet Daten auf die WgP-Schnittstelle |
| strtowgpbuffer | definiert den Inputbuffer für die WgP-Schnittstelle |
| sendir | sendet Daten auf die IR-Schnittstelle |
| | |
| setbusrs232baud | setzt die Baudrate für die RS232-Busschnittstelle |
| setbusrs232setting | setzt die Settings für die RS232-Busschnittstelle |
| sendbus | sendet Daten auf die RS232-Busschnittstelle |
| strtobusbuffer | definiert den Inputbuffer für die RS232-Busschnittstelle |
| sendvartobus | sendet den Wert der Variable auf den Bus (Bargraph Display) |

diverse Funktionen

| | |
|-------------|--|
| getfirmware | gibt die aktuelle Firmware zurück |
| getproglen | gibt die länge des aktuellen Programm zurück |
| memwrite | schreibt in den EEPROM-Speicher |
| memread | liest Daten aus dem EEPROM-Speicher |

WgC-Compiler

| | |
|------------------|----------------------------------|
| compilieren C | compiliert das aktuelle Programm |
| programmieren P | programmiert die Steuerung |

Variablen (numerische Variablen)

numerische Variablen sind vom Typ INTEGER (16bit) und können Werte von -32768 bis +32768 annehmen.

defvar *definiert eine Variable*

Syntax 1: `defvar Variable = Wert`
Variable = Bezeichner

Beispiel: `defvar x = 0`
definiert die Variable Status und weist ihr den Wert 0 zu.

[<Übersicht>](#)

setvar *weisst einer Variable einen Wert zu*

Syntax 1: `setvar Variable = Wert`
Syntax 2: `setvar Variable = Variable`
Hinweis: `setvar` kann entfallen, also `Variable = Wert`

Beispiel: `setvar x = 20`
oder:
`x = 20`

[<Übersicht>](#)

addvar *addiert einen Wert zu einer Variable*

Syntax 1: `addvar Var, Wert`
Syntax 2: `addvar Var, Var`

Beispiel: `addvar x, 1`
`addvar x, y`

[<Übersicht>](#)

subvar *führt eine Subtraktion aus*

Syntax 1: `subvar Var, Wert`
Syntax 2: `subvar Var, Var`

Beispiel: `subvar x, 1`
`subvar x, y`

[<Übersicht>](#)

calc *führt eine Rechenoperation aus*

Syntax 1: `calc Var = Var + Wert`
Syntax 2: `calc Var = Var + Var`
Syntax 3: `calc Var = Wert + Var`
Syntax 4: `calc Var = Wert + Wert`
calc kann entfallen, also `Var = Var + Wert`

Operationen:

- + Addition
- Subtraktion
- * Multiplikation
- / Division
- | Oder
- & Und
- % Rest Division

Beispiel: `calc x = y + 10`
`x = 100 + 200`

[<Übersicht>](#)

Stringvariablen (Zeichen)

defstr definiert einen String

Syntax 1: defstr String, Wert
*String = Bezeichner, Wert = max. Länge des Strings
das erste Zeichen im String ist die Stelle 0 !*

Beispiel: defstr buffer, 10

[<Übersicht>](#)

setstr weist einem String Zeichen zu

Syntax 1: setstr String = Wert
Syntax 2: setstr String = Text
Syntax 3: setstr String = Variable
Syntax 4: setstr String = String
*setstr kann entfallen, also String = Text
Text: [0d] = hexadezimale Schreibweise, hier = CR (0x0d)*

Beispiel: defstr buffer, 10
 setstr buffer = "Hallo[0d]"

[<Übersicht>](#)

addstr fügt Zeichen an einen String an

Syntax 1: addstr String , Wert
Syntax 2: addstr String , Text
Syntax 3: addstr String , Variable
Syntax 4: addstr String , String

Beispiel: defstr buffer, 10
 setstr buffer = "Hallo"
 addstr buffer, "Welt"

[<Übersicht>](#)

strtovar wandelt ASCII-Werte aus Strings in Variablen

Syntax 1: strtovar Variable, String (Wert)
Syntax 2: strtovar Variable, String (Variable)

Beispiel: defvar x = 0
 defstr buffer, 10
 setstr buffer = "ABCDEF"
 strtovar x, buffer(2)

[<Übersicht>](#) ~x: 67

vartostr Wandelt Variablen in Strings

Syntax 1: vartostr String, Variable (Wert)
wert gibt die Anzahl Stellen an, 0 = nur soviele Stellen wie nötig

Beispiel: `defvar x = 0`
`defstr s, 10`
`x = 12`
`vartostr s, x(3)`

[<Übersicht>](#) ~s: 012

getstrlen gibt die Länge eines Strings aus

Syntax 1: getstrlen Variable, String

[<Übersicht>](#)

find sucht nach Zeichen in einem String

Syntax 1: find Variable, String (Wert)
Syntax 2: find Variable, String (Text)
Syntax 3: find Variable, String (Variable)
Syntax 4: find Variable, String (String)
gibt die Stelle des gefundenen Text zurück, -1 wenn der Text nicht gefunden wird.

Beispiel: `defvar x = 0`
`defstr buffer, 10`
`buffer = "0123456789"`
`find x, buffer ("345")`

[<Übersicht>](#) ~x: 3

bytetostr schreibt ein Zeichen in einen String

Syntax 1: bytetostr String, Variable (Wert)
Syntax 2: bytetostr String, Variable (Variable)
Syntax 3: bytetostr String, Wert (Wert)
Syntax 4: bytetostr String, Wert (Variable)
*der Wert in Klammern gibt die zu schreibende Stelle an
der Wert nach dem Komma gibt den zu schreibenden Wert an.*

Beispiel: `defstr buffer, 10`
`bytetostr buffer, 65 (0)`

[<Übersicht>](#) ~buffer: A (65=A wird an die erste Stelle geschrieben)

delstrpos löscht eine Stelle in einem String

Syntax 1: delstrpos String, Wert
delstrpos String, Variable

Beispiel: `defstr buffer, 10`
`buffer = "ABCD"`
`delstrpos buffer, 1`

[<Übersicht>](#) ~buffer: ACD (Stelle 1 wurde gelöscht)

Timer, Delay

retjumptimer1 definiert und aktiviert den Timer

Syntax 1: retjumptimer1 Marke
Die Sprungmarke muß mit einer return Anweisung abgeschlossen werden.

Beispiel: retjumptimer1 Timer

```
::Main
jump Main

::Timer
    send "test"
return
```

[<Übersicht>](#)

settimer setzt das Timerintervall (Zeit)

Syntax 1: settimer wert
100 = 1 Sekunde

[<Übersicht>](#)

timerstop stopt den Timer

Syntax 1: timerstop

[<Übersicht>](#)

timergo startet den Timer nach Timerstop

Syntax 1: timergo

[<Übersicht>](#)

delay wartet für eine bestimmte Zeit

Syntax 1: delay Wert
Syntax 2: delay Variable
100 = 1 Sekunde

Beispiel: delay 100 wartet 1 Sekunde

[<Übersicht>](#)

I/O's, Taster, Leds

din definiert einen Port als Input

Syntax 1: din Wert
Syntax 2: din Variable

Beispiel: din 1 Port 1 wird als Input definiert

[<Übersicht>](#)

dout definiert einen Port als Output

Syntax 1: dout Wert
Syntax 2: dout Variable

Beispiel: dout 2 Port 2 wird als Output definiert

[<Übersicht>](#)

on schaltet einen Port/Led ein. Setzt einen Pullup Widerstand

Syntax 1: on Wert
Syntax 2: on Variable
*Input / Output definiert mit din / dout:
Port ist Input: Pullupwiderstand ein, damit der
IO gegen GND geschaltet werden kann, z.B. mit einem Taster
Port ist Output: Port wird auf High geschaltet
(z.B. Led Ein)*

Beispiel: dout 2 Port 2 wird als Output definiert
 on 2 Port 2 wird eingeschaltet (High)

[<Übersicht>](#)

off schaltet einen Port/Led aus. Löscht einen Pullup Widerstand

Syntax 1: off Wert
Syntax 2: off Variable
*Input / Output definiert mit din / dout:
Port ist Input: Pullupwiderstand aus.
Port ist Output: Port wird auf Low geschaltet
(z.B. Led Aus)*

Beispiel: dout 2 Port 2 wird als Output definiert
 off 2 Port 2 wird ausgeschaltet (Low)

[<Übersicht>](#)

portin gibt den Status eines Ports zurück

Syntax 1: portin Variable, Wert
Syntax 2: portin Variable, Variable

Beispiel: defvar x = 0
 portin x, 3 der Status des IO 3 wird in x gespeichert
 (High = 1, Low = 0)

[<Übersicht>](#)

blk lässt einen Port/Led blinken

Syntax 1: blk Wert1, Wert2 (Wert3)
*Wert 1 gibt den IO an , Wert 2 die Anzahl der Blinkimpulse
wobei ungerade auf Aus endet und gerade auf Ein.
Wert 3 gibt die Zeit an (100 = 1 Sek.)*

Syntax 1: blk Variable, Wert2 (Wert3)

Beispiel: blk 2, 10(50) Max. 10 Ports gleichzeitig

[<Übersicht>](#)

getblk *gibt die verbleibene Zahl von blinken zurück*

Syntax 1: getblk Variable, Wert
 gibt die verbleibenden Blinkimpulse zurück

[<Übersicht>](#)

adcin *gibt den Wert der Spannung am Analogeingang zurück*

Syntax 1: adcin Variable, Wert
Syntax 2: adcin Variable, Variable

Beispiel: defvar x = 0
 adcin x, 1 *gibt die Spannung an Port 1 zurück*

[<Übersicht>](#)

Verzweigungen Programmsprünge

::Marke *definiert eine Sprungmarke*

Syntax 1: ::Marke

Beispiel: ::Main
 jump Main

[<Übersicht>](#)

jump *führt einen Programmsprung aus*

Syntax 1: jump Marke

Beispiel: ::Main
 jump Main

[<Übersicht>](#)

retjump *springt in eine Unterroutine*

Syntax 1: retjump Marke

Beispiel: ::Main
 retjump Unterroutine
 jump Main

```
::Unterroutine  
  send "test"  
  return
```

[<Übersicht>](#)

return *beendet eine Unterroutine*

Syntax 1: return

Beispiel:

```
::Main
    retjump Unterroutine
jump Main

::Unterroutine
    send "test"
return
```

[<Übersicht>](#)

if *leitet eine Bedingung ein*

Syntax 1: if Variable Bedingung Wert

Syntax 1: if Variable Bedingung Variable

vergleicht eine Variable mit einem Wert/Variable

verfügbare
Bedingungen

| | |
|----|----------------|
| < | kleiner als |
| > | grösser als |
| = | gleich |
| == | gleich |
| <= | kleiner gleich |
| >= | grösser gleich |
| != | nicht, unwahr |
| | oder |
| && | und |

Beispiel:

```
defvar x = 0

if x = 10
    send "test"
endif
```

[<Übersicht>](#)

else *unterteilt if und endif (optional)*

Syntax 1: else

*gibt die Möglichkeit, erfüllte und nicht erfüllte
Bedingungen zu unterscheiden*

Beispiel:

```
defvar x = 0

if x = 10
    send "x=10"
else
    send "x<>10"
endif
```

[<Übersicht>](#)

endif *beendet die if-Anweisung*

Syntax 1: endif

[<Übersicht>](#)

press fragt eine Taste/Port ab

Syntax 1: press Wert
Syntax 2: press Variable

Beispiel: `press 1` *fragt den Port 1 ab*
`send "1"`
`endpress`

[<Übersicht>](#)

holdpress wartet bis eine Taste/Port freigegeben wird

Syntax 1: holdpress Wert
Syntax 2: holdpress Variable

Beispiel: `press 1` *fragt Taste/Port 1 ab*
`send "start"`
`holdpress 1` *verweilt bis Taste/Port 1 losgelassen wird*
`send "stop"`
`endpress`

[<Übersicht>](#)

endpress beendet die press-Anweisung

Syntax 1: endpress

[<Übersicht>](#)

push reagiert auf einen Tastendruck

Syntax 1: push Wert
Syntax 2: push Variable

Beispiel: `push 1` *fragt den Port 1 ab*
`send "1"` *im Gegensatz zu press wird die Abfrage nur einmal ausgeführt und die Anweisungen zwischen push und endpush nur nach erneutem drücken wieder ausgeführt.*
`endpush`

[<Übersicht>](#)

endpush beendet die push-Anweisung

Syntax 1: endpush

[<Übersicht>](#)

release reagiert, wenn eine Taste losgelassen wird

Syntax 1: release Wert
Syntax 2: release Variable

Beispiel: `release 1` *fragt den Port 1 ab*
`send "1"` *im Gegensatz zu press wird die Abfrage nur einmal ausgeführt und die Anweisungen zwischen push und endpush nur nach erneutem drücken wieder ausgeführt.*
`endrelease`

[<Übersicht>](#)

endrelease beendet die release-Anweisung

Syntax 1: endrelease

[<Übersicht>](#)

count leitet eine Zählerschleife ein

Syntax 1: count Variable = Wert - Wert
Syntax 2: count Variable = Wert - Variable
Syntax 3: count Variable = Variable - Wert
Syntax 4: count Variable = Variable - Variable
Eröffnet eine Zählerschleife und zählt von Wert1 bis Wert2.

Beispiel: `defvar x = 0`

`count x = 1 - 10` *zählt von 1 - 10*
`send x`
`endcount`

[<Übersicht>](#)

endcount beendet die count-Anweisung

Syntax 1: endcount

[<Übersicht>](#)

Schnittstellenfunktionen

setbaud setzt die Baudrate für die RS232-Schnittstelle

Syntax 1: setbaud Wert
erlaubte Werte: 300 - 115200

Beispiel: `setbaud 9600` *setzt die Schnittstelle auf 9600 baud*

[<Übersicht>](#)

setsetting setzt die Settings für die RS232-Schnittstelle

Syntax 1: `setsetting "N,8,1"`
 möglich: "N/O/E, 7/8, 1/2"

Beispiel: `setsetting "0,8,2"` *setzt Odd Parität, 8 Datenbits, 2 Stopbits*

[<Übersicht>](#)

send sendet Daten auf die RS232-Schnittstelle

Syntax 1: `send Text`
Syntax 2: `send Wert`
Syntax 3: `send String`
Syntax 4: `send Variable`

Beispiel 1: `send "Power On[0d]"`

Beispiel 2: `defstr buffer, 10`
 `buffer = "0123456789"`
 `send buffer`

[<Übersicht>](#)

strtobuffer definiert den Inputbuffer für die RS232-Schnittstelle

Syntax 1: `strtobuffer String`
 Zeichen die auf der RS232-Schnittstelle eingehen werden
 in den angegebenen String geschrieben

Beispiel 1: `defstr buffer, 10`
 `strtobuffer buffer`

[<Übersicht>](#)

setwgpbaud setzt die Baudrate für die WgP-Schnittstelle

Syntax 1: `setwgpbaud Wert`
 erlaubte Werte: 300 - 115200

Beispiel: `setwgpbaud 9600` *setzt die Schnittstelle auf 9600 baud*

[<Übersicht>](#)

setwgpsetting setzt die Settings für die WgP-Schnittstelle

Syntax 1: `setwgpsetting "N,8,1"`
 möglich: "N/O/E, 7/8, 1/2"

Beispiel: `setwgpsetting "0,8,2"` *setzt Odd Parität, 8 Datenbits, 2 Stopbits*

[<Übersicht>](#)

wgpsend *sendet Daten auf die WgP-Schnittstelle*

Syntax 1: wgpsend Text
Syntax 2: wgpsend Wert
Syntax 3: wgpsend String
Syntax 4: wgpsend Variable

Beispiel 1: wgpsend "Power On[0d]"

Beispiel 2: defstr buffer, 10
buffer = "0123456789"
wgpsend buffer

[<Übersicht>](#)

strtowgpbuffer *definiert den Inputbuffer für die WgP-Schnittstelle*

Syntax 1: strtowgpbuffer String
*Zeichen die auf der WgP-Schnittstelle eingehen werden
in den angegebenen String geschrieben*

Beispiel 1: defstr wgpbuffer, 10
strtowgpbuffer wgpbuffer

[<Übersicht>](#)

sendir *sendet Daten auf die IR-Schnittstelle*

Syntax 1: sendir String
*Sendet IR-Kommandos auf die IR-Schnittstelle
Die IR-Daten können aus unsere IR-Datenbank entnommen werden
oder wir lesen die Daten für Sie ein.*

Beispiel 1: defstr ir, 50
ir = "[09][33][2e][2e][1e][00][b2][ac][ad][20]"
sendir ir *sendet die IR-Daten die ir zugewiesen wurden*

[<Übersicht>](#)

setbusrs232baud *setzt die Baudrate für die RS232-Busschnittstelle*

Syntax 1: setbusrs232baud Modul, Wert
erlaubte Werte: 300 - 115200

Beispiel: setbusrs232baud 1,9600 *setzt die Schnittstelle auf 9600 baud*

[<Übersicht>](#)

setbusrs232settiq *setzt die Settings für die RS232-Busschnittstelle*

Syntax 1: setbusrs232setting Modul, "N,8,1"
möglich: "N/O/E, 7/8, 1/2"

Beispiel: setbusrs232setting 1, "O,8,2"
setzt Odd Parität, 8 Datenbits, 2 Stopbits

[<Übersicht>](#)

sendbus *sendet Daten auf die RS232-Busschnittstelle*

Syntax 1: sendbus Modul, Text
Syntax 2: sendbus Modul, Wert
Syntax 3: sendbus Modul, String
Syntax 4: sendbus Modul, Variable

Beispiel 1: `sendbus 1, "Power On[0d]"`

Beispiel 2: `defstr buffer, 10
buffer = "0123456789"
sendbus 1, buffer`

[<Übersicht>](#)

strtobusbuffer *definiert den Inputbuffer für die RS232-Busschnittstelle*

Syntax 1: Sendvartobus Variable

Beispiel 1: `defvar x = 10
sendvartobus x`

[<Übersicht>](#)

sendvartobus *sendet den Wert der Variable auf den Bus (Bargraph Display)*

Syntax 1: strtobusbuffer Modul, String
*Zeichen die auf der RS232-Busschnittstelle eingehen werden
in den angegebenen String geschrieben*

Beispiel 1: `defstr buffer1, 10
strtobusbuffer 1, buffer1`

[<Übersicht>](#)

diverse Funktionen

getfirmware *gibt die aktuelle Firmware zurück*

Syntax 1: getfirmware String

Beispiel 1: `defstr version, 20
getfirmware version
send version`

[<Übersicht>](#)

getproglen *gibt die länge des aktuellen Programm zurück*

Syntax 1: getproglen Variable

Beispiel 1: `defvar len = 0
getproglen len`

[<Übersicht>](#)

memwrite schreibt in den EEPROM-Speicher

Syntax 1: memwrite Wert, Wert
Syntax 2: memwrite Wert, Variable
Syntax 3: memwrite Variable, Wert
Syntax 4: memwrite Variable, Variable
memwrite Adresse, Wert

Beispiel 1: `memwrite 1,255` *schreibt die 255 in Adresse 1*

[<Übersicht>](#)

memread liest Daten aus dem EEPROM-Speicher

Syntax 1: memread Variable, Wert
Syntax 2: memread Variable, Variable
memread Variable, Adresse

Beispiel 1: `defvar wert = 0`
`memread wert, 1` *liest den Wert aus Adresse 1*

[<Übersicht>](#)

WgC-Compiler

compilieren |C| *compiliert das aktuelle Programm*

> Iconleiste |C| Compiliert das aktuelle Programm.
Wird kein Fehler festgestellt, passiert nichts.
Wird ein Fehler festgestellt, öffnet sich das Fehlerfenster.
Durch Doppelclick auf einen Fehler gelangt man in die entsprechende
[<Übersicht>](#) Zeile.

programmieren |P| *compiliert das aktuelle Programm und öffnet das Programmierenfenster*

> Iconleiste |P| Compiliert das aktuelle Programm:
Wird kein Fehler festgestellt, öffnet sich das Programmfenster.
Wird ein Fehler festgestellt, öffnet sich das Fehlerfenster.
Durch Doppelclick auf einen Fehler gelangt man in die entsprechende
Zeile.
Um eine Steuerung zu programmieren, reicht es normalerweise, den
Program-Button zu drücken. Wurde im Programm die Baudrate der
WgP-Schnittstelle von 115200 Baud abgeändert, so muss die Steuerung
nach drücken des Program-Button kurz von der Spannung getrennt werden.
Nach wiederanlegen der Spannung wird dann das Programm aufgespielt.
Unter Comport wird die serielle Schnittstelle des Computers,
[<Übersicht>](#) die zum Aufspielen der Software benutzt wird, ausgewählt.