Befehlsübersicht von PropBasic

Basis-Präfix für Zahlenwerte und ASCII-Zeichen

`%`	Binärzahl
`%%`	Quaternärzahl (0..3)
`$`	Hexadezimalzahl (0..9, A..F)
`"x"`	ASCII-Zeichen

Datentypen

Hinweise

- Konstante Daten müssen immer vor dem Programmcode deklariert werden.
- Pins, globale Hub-Variablen und DATAs können von allen TASKs (Cogs) verwendet werden.

Lokale Variablen im Cog-RAM (VAR)

```
LONG
```

Globale Variablen im Hub-RAM (HUB)

```
BYTE

WORD

LONG

STRING
```

Konstante Daten im Hub-RAM (DATA, WDATA, LDATA)

Datentype (Byte, Word oder Long) ist Kommando abhängig. Ein String wird Byte-weise im Hub-RAM gespeichert.

Operatoren

Hinweise

- Operatoren können nur auf Werte angewendet werden, wenn das Ergebnis in eine Variable gespeichert wird.
- Operatoren können nicht innerhalb von Kommandos verwendet werden.
- Pro Programmzeile ist nur ein Operator erlaubt.

Unäre mathematische Operatoren

ABS Absoluten Wert zurückgeben

```
value1 = ABS value2
```

LEN String-Länge zurückgeben

```
value1 = LEN string1
```

VAL Wert eines Strings zurückgeben

```
value1 = VAL string1
```

GETADDR Adresse einer Hub-Variablen zurückgeben

```
value1 = GETADDR hubvar{(offset)}
```

SGN Vorzeichen eines Wertes zurückgeben

```
value1 = SGN value2
```

~ Bitweises NICHT

```
value1 = ~value2
```

- Wert negieren und zurückgeben

```
value1 = -value2
```

Binäre mathematische Operatoren

+ Addition

```
value1 = value2 + value3
```

- Subtraktion

```
value1 = value2 - value3
```

***** Multiplikation

```
value1 = value2 * value3
```

***/** Multiplikation, 16-Bit-Verschiebung

```
value1 = value2 */ value3
```

****** Multiplikation, 32-Bit-Verschiebung

```
value1 = value2 ** value3
```

/ Division

```
value1 = value2 / value3
```

// Modulo

```
value1 = value2 // value3
```

& AND Bitweises UND

```
value1 = value2 {&|AND} value3
```

|	**OR**	Bitweises ODER
		`value1 = value2 {||OR} value3`
^	**XOR**	Bitweises Exklusiv-ODER
		`value1 = value2 {^|XOR} value3`
&~	**ANDN**	Bitweises UND-NICHT
		`value1 = value2 {&~|ANDN} value3`
MIN		Minimum von zwei Werten zurückgeben
		`value1 = value2 MIN value3`
MAX		Maximum von zwei Werten zurückgeben
		`value1 = value2 MAX value3`
>>	**SHR**	Rechtsverschiebung
		`value1 = value2 {>>|SHR} value3`
<<	**SHL**	Linksverschiebung
		`value1 = value2 {<<|SHL} value3`

String-Operatoren

LEFT	Linken String-Abschnitt zurückgeben
	`string1 = LEFT string2, count`
RIGHT	Rechten String-Abschnitt zurückgeben
	`string1 = RIGHT string2, count`
MID	Mittleren String-Abschnitt zurückgeben
	`string1 = MID string2, start, count`
STR	Wert in String umwandeln
	`string1 = STR value1, digits{, option}`
+	Zwei Strings zusammenfassen
	`string1 = string2 + string3`

PropBasic-Kommandos

Hinweise

- Der Hauptprogrammcode befindet sich immer in Cog 0.
- VARs, SUBs und FUNCs können nur von dem TASK (Cog) verwendet werden, in dem sie deklariert worden sind.

Präprozessor-Direktiven

' Einzeiliger Kommentar

```
' kommentar
```

{ } Mehrzeiliger Kommentar-Block

```
{ kommentar }
```

LOAD PropBasic-Code aus externer .pbas-Datei einfügen

```
LOAD "dateiname.pbas"
```

Programmablaufsteuerbefehle

BRANCH Springe zu Label, welches in der Variablen angegeben ist

```
BRANCH var,label0,label1,label2{,label3{,etc}}
```

DJNZ Zähler dekrementieren, zu Label springen, wenn Zähler nicht gleich 0

```
DJNZ var,label
```

DO Wiederholende Programmschleife erstellen

WHILE

UNTIL

LOOP … Ende der wiederholenden Programmschleife

```
DO
  ...
LOOP

DO
  ...
LOOP var

DO WHILE var cond value
  ...
LOOP

DO
  ...
LOOP UNTIL var cond value
```

END Programmende

```
END
```

EXIT DO…LOOP- oder FOR…NEXT-Schleife beenden

```
IF var cond value THEN EXIT
```

FOR Programmschleife erstellen

TO

STEP

NEXT … Ende der Programmschleife

```
FOR var = startvalue TO endvalue
  ...
NEXT

FOR var = startvalue TO endvalue STEP deltavalue
  ...
NEXT
```

FUNC Funktion mit Rückgabewert erstellen

ENDFUNC … Ende des Funktion-Blocks

```
name FUNC {minparam{,maxparam}}

FUNC name
  ...
  ENDFUNC
```

GOSUB Springe zu Subroutine

```
GOSUB subroutine
```

GOTO Springe zu Label

```
GOTO label
```

IF Bedingten Code-Block erstellen

OR|AND

ELSE|ELSEIF

THEN

ENDIF … Ende des bedingten Code-Blocks

```
IF var cond value THEN label

IF var cond value THEN
  ...
ENDIF

IF var cond value THEN
  ...
ELSE
  ...
ENDIF
```

```
IF var cond value THEN
  ...
ELSEIF var cond value THEN
  ...
ENDIF

IF var cond value OR
 var cond value THEN
  ...
ELSE
  ...
ENDIF

IF var cond value OR
 var cond value AND
 var cond value THEN
  ...
ELSE
  ...
ENDIF
```

ON Durch Index-Variable bedingter Sprung

GOTO|GOSUB … zum Label oder Subroutine

```
ON var GOTO label1,label2{,label3{,etc}}

ON var = value1,value2,value3 GOTO label1,label2,label3

ON var GOSUB label1,label2{,label3{,etc}}

ON var = value1,value2,value3 GOSUB label1,label2,label3
```

PAUSE Programm für x Millisekunden anhalten

```
PAUSE value
```

PAUSEUS Programm für x Mikrosekunden anhalten

```
PAUSEUS value
```

PROGRAM Programmeinstiegs-Label setzen

```
PROGRAM startlabel {LMM|PASD}
```

RETURN Rücksprung aus Subroutine

```
RETURN value1{,value2{,value3{,value4}}}
```

SUB Subroutine mit Parametern erstellen

ENDSUB … Ende Subroutine-Block

```
name SUB {minparam{,maxparam}}

SUB name
  ...
  ENDSUB
```

Ein- und Ausgabe-Befehle

CON Konstante mit einem Wert oder String erstellen

```
name CON value
```

DATA Konstante Daten (8 Bit) im Hub-RAM erstellen

```
{label} DATA value1{,value2{,value3{,etc}}}
```

WDATA Konstante Daten (16 Bit) im Hub-RAM erstellen

```
{label} WDATA value1{,value2{,value3{,etc}}}
```

LDATA Konstante Daten (32 Bit) im Hub-RAM erstellen

```
{label} LDATA value1{,value2{,value3{,etc}}}
```

DEC Wert einer Variablen mit 1 (oder einem andern Wert) subtrahieren

```
DEC var{,value}
```

FILE Binärdatei laden, Bytes wie bei DATA in Hub-RAM einfügen

```
{label} FILE "dateiname.bin"
```

HUB Hub-Variable erstellen

```
name HUB datentyp{(elemente)} {= value}
```

INC Wert einer Variablen mit 1 (oder einem andern Wert) addieren

```
INC var{,value}
```

LET Variablenausrichtung im Speicher einstellen (optional)

```
optional
```

RANDOM Zufallszahl erzeugen

```
RANDOM seedvar{,copyvar}
```

RDBYTE Wert (8 Bit) aus Hub-Variable oder DATA lesen

```
RDBYTE hubvar{(offset)},var1{,var2{,var3{,etc}}}
```

RDSBYTE Wert mit Vorzeichen (8 Bit) aus Hub-Variable oder DATA lesen

```
RDSBYTE hubvar{(offset)},var1{,var2{,var3{,etc}}}
```

RDWORD Wert (16 Bit) aus Hub-Variable oder DATA lesen

```
RDWORD hubvar{(offset)},var1{,var2{,var3{,etc}}}
```

RDSWORD Wert mit Vorzeichen (16 Bit) aus Hub-Variable oder DATA lesen

```
RDSWORD hubvar{(offset)},var1{,var2{,var3{,etc}}}
```

RDLONG Wert (32 Bit) aus Hub-Variable oder DATA lesen

```
RDLONG hubvar{(offset)},var1{,var2{,var3{,etc}}}
```

VAR Variable erstellen

```
name VAR LONG{(elemente)} {= value}
```

WRBYTE Neuen Wert (8 Bit) in Hub-Variable schreiben

```
WRBYTE hubvar{(offset)},value1{,value2{,value3{,etc}}}
```

WRWORD Neuen Wert (16 Bit) in Hub-Variable schreiben

```
WRWORD hubvar{(offset)},value1{,value2{,value3{,etc}}}
```

WRLONG Neuen Wert (32 Bit) in Hub-Variable schreiben

```
WRLONG hubvar{(offset)},value1{,value2{,value3{,etc}}}
```

Hardware-Funktionen

_FREQ Konstante mit ursprünglicher Taktfrequenz

CLKSET Taktmodus des Propeller-Chips setzen

```
CLKSET mode,freq
```

COUNTERA Hardware-Counter-Parameter einstellen (Counter A)

```
COUNTERA mode{,apin{,bpin{,frqx{,phsx}}}}
```

COUNTERB Hardware-Counter-Parameter einstellen (Counter B)

```
COUNTERB mode{,apin{,bpin{,frqx{,phsx}}}}
```

DEVICE Gerätetype und -Parameter festlegen

```
DEVICE deviceid{,settings{,settings}}
```

FREQ Taktfrequenz (nach PLL-Multiplikator) angeben

```
FREQ freq
```

LOCKCLR Lock-ID löschen

```
LOCKCLR value{,var}
```

LOCKNEW Neue Lock-ID anfordern

```
LOCKNEW var
```

LOCKRET Lock-ID zurückgeben

```
LOCKRET var
```

LOCKSET Lock-ID setzen

```
LOCKSET value{,var}
```

NOP Keine Ausführung der Cog für eine Befehlslänge

```
NOP
```

WAITCNT Warten bis Systemcounter den Zielwert erreicht hat

```
WAITCNT target,delta
```

WAITPEQ Warten bis Pin(s) gleich Maskierungswert

```
WAITPEQ state,mask
```

WAITPNE Warten bis Pin(s) nicht mehr gleich Maskierungswert

```
WAITPNE state,mask
```

WAITVID Warten bis Video-Sequenzer neue Daten erhalten kann

```
WAITVID colours,pixels
```

XIN Externe Taktfrequenz (vor PLL-Multiplikator) angeben

```
XIN freq
```

I/O-Pin-Funktionen

HIGH Pin auf Ausgang schalten und auf High setzen

```
HIGH {pinname|pinnum}
```

INPUT Pin auf Eingang schalten

```
INPUT {pinname|pinnum}
```

LOW Pin auf Ausgang schalten und auf Low setzen

```
LOW {pinname|pinnum}
```

OUTPUT Pin auf Ausgang schalten

```
OUTPUT {pinname|pinnum}
```

PIN Pin-Variable erstellen

```
name PIN {pinnum|msbpin..lsbpin} {modifizierer}
```

PULSIN Eingangsimpulslänge in Mikrosekunden messen

```
PULSIN {pinname|pinnum},state,var
```

PULSOUT Ausgangsimpuls mit bestimmter Länge und Taktung erzeugen

```
PULSOUT {pinname|pinnum},value
```

RCTIME Zustandswechselfrequenz am Pin in Mikrosekunden messen

```
RCTIME {pinname|pinnum},state,var
```

REVERSE Pin-Richtung (Eingang/Ausgang) vertauschen

```
REVERSE {pinname|pinnum}
```

TOGGLE Pin-Zustand (High/Low) vertauschen

```
TOGGLE {pinname|pinnum}
```

Multitasking

COGID Cog-ID der ausführenden Cog abfragen

```
COGID var
```

COGINIT Neuen Task in der Cog mit einer bestimmten ID starten

```
COGINIT taskname,cogid
```

COGSTART Neuen Task in der nächsten freien Cog starten

```
COGSTART taskname{,cogid}
```

COGSTOP Cog mit einer bestimmten ID oder ausführende Cog stoppen

```
COGSTOP {cogid}
```

STACK Stack-Größe festlegen

```
STACK value
```

TASK Code-Block für eine separate Cog erstellen

ENDTASK … Ende Code-Block

```
name TASK {LMM} {AUTO}

TASK name
  ...
  ENDTASK
```

Serielle Schnittstellen

I2CREAD Ein Byte über I²C-Bus lesen

```
I2CREAD sdapin,sclpin,var,ackbitvalue
```

I2CSPEED I²C-Bus-Take einstellen

```
I2CSPEED multipizierer
```

I2CSTART I²C-Bus starten

```
I2CSTART sdapin,sclpin
```

I2CSTOP I²C-Bus stoppen

```
I2CSTOP sdapin,sclpin
```

I2CWRITE Ein Byte über I²C-Bus senden

```
I2CREAD sdapin,sclpin,value{,ackbitvar}
```

OWREAD Ein Byte über 1-Wire-Bus lesen

```
OWREAD dqpin,var{\bits}
```

OWRESET Reset-Signal über 1-Wire-Bus senden

```
OWRESET dqpin{,statusvar}
```

OWWRITE Ein Byte über 1-Wire-Bus senden

```
OWWRITE dqpin,value{\bits}
```

SERIN Serieller Eingang

```
SERIN {pinname|pinnum},baud,var{,timeout{,label}}
```

SEROUT Serieller Ausgang

```
SEROUT {pinname|pinnum},{T|N|OT|ON}baud,{char|string}
```

SHIFTIN SPI-Eingang

```
SHIFTIN datapin,clkpin,mode,var{\bits}{,speed}
```

SHIFTOUT SPI-Ausgang

```
SHIFTOUT datapin,clkpin,mode,value{\bits}{,speed}
```

Assembler-Code-Einfügen

**** Einzeiliger Assembler-Code

```
\ pasmcode
```

ASM Mehrzeiliger Assembler-Code-Block

ENDASM … Blockende

```
ASM
  pasmcode
  ENDASM
```

INCLUDE Assemblercode aus externer .spin-Datei einfügen

```
INCLUDE "dateiname.spin"
```

Debugger-Funktionen

BREAK Setzt Break-Point für Debugger

```
BREAK
```

WATCH Variablen updaten, wenn Debugger benutzt wird

```
WATCH
```