BasicBeetle

Befehlsreferenz

Für BasicBeetle 1.31 oder höher

1. Ausgabe

Inhaltsverzeichnis

A

Addition, +

Verknüpfung

Syntax:

<Ergebnis>=<Wert1>+<Wert2>

Beschreibung:

Bildet die Summe beider angegebenen Werte.

Beispiel:

```
>PRINT 34+17
51
Ready
>
```

AND

Operand

Syntax:

<Ergebnis>=<Wert1> AND <Wert2>

Beschreibung:

Verknüpft zwei Werte bitweise durch ein logisches Und miteinander.

Beispiel:

```
>PRINT 170 AND 144
128
Ready
>
```

AFTER - GOSUB
Befehl

Syntax:

AFTER <Zeit>,<Timer> GOSUB <Zeile>

Beschreibung:

Startet einen einmaligen Timer-Interrupt. Die angegebene Zeile wird nach der angegebenen Zeit, angegeben in 1/100tel Sekunden, angesprungen. Hierbei sind 3 unterschiedliche Interrupts möglich (Timer 1-3). Nach auslösen des AFTER-Interrupts wird dieser für den angegebenen Timer gelöscht und kann neu gestartet werden.

Bei einer Zeitangabe von 0 wird der angegebene Timer-Interrupt gestoppt. Der Zeitwert darf 65535/100tel Sekunden nicht überschreiten.

Die 3 Timer des AFTER-Befehls arbeiten unabhängig von den 3 EVERY-Timern. Es sind also jeweils 3 Timer möglich.

Beispiel:

```
>10 DIM ESC
>20 PRINT "Die Bombe explodiert in 10 Sekunden ...":AFTER 1000,1 GOSUB 50
>30 IF ESC THEN PRINT "Booooom!":END
>40 GOTO 30
>50 ESC=TRUE:RETURN
>RUN
Die Bombe explodiert in 10 Sekunden ...
(10 Sekunden warten)
Booooom!
Ready
>
```

ASC
Funktion

Syntax:

<Zeichennummer>=ASC(<Zeichen>)

Beschreibung:

Ermittelt die Ascii-Nummer des angegebenen Zeichens zurück. Ist der übergebene String leer, wird ein Fehler erzeugt. Enthält der String mehr als 1 Zeichen, wird nur das erste zurück gegeben.

Beispiel:

```
>PRINT ASC("A")
65
Ready
>
```

B

BAUD
Variable

Syntax:

BAUD=<Baudrate>
<Baudrate>=BAUD

Beschreibung:

Mit BAUD ist es möglich, eine andere Baudrate, als die 19200 Baud, als Standard festgelegte, auszuwählen. Die Baudrate muss nur einmalig geändert werden. Der neue Wert bleibt dauerhaft, auch nach einem Reset oder dem ausschalten des BasicBeetle, erhalten. Es sind folgende Übertragungsraten erlaubt: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200.

Es ist zu beachten, dass nach dem Ändern der Baudrate, auch das Terminal mit dieser Rate arbeiten muss. Daher sollte auch das Terminal auf die neue Übertragungsrate umgestellt werden.

Wird die Variable BAUD ausgelesen, so gibt diese die aktuell gewählte Baudrate zurück.

Beispiel:

```
>BAUD=19200
Ready
>PRINT BAUD
19200
Ready
>
```

BIN$
Funktion

Syntax:

<Binärstring>=BIN$(<Wert>)

Beschreibung:

Erstellt eine Zeichenkette, welche den angegebenen Wert als Binärwert in einem String zurück gibt.

Beispiel:

```
>PRINT BIN$(570)
1000111010
Ready
>
```

BIT
Funktion

Syntax:

<Bitstatus>=BIT(<Wert>,<Bitnummer>)

Beschreibung:

Ermittelt den Status eines einzelnen Bits im angegebenen Wert. Ist das entsprechende Bit gesetzt, wird TRUE (=65535) zurück gegeben. Ansonsten erhält man den Wert 0 (für FALSE).

Es können nur 16 Bit Werte überprüft werden. 32 Bit Zahlen müssen mit HIWORD oder LOWORD aufgeteilt werden.

Beispiel:

```
>PRINT BIT(144,4):PRINT BIT(144,5)
65535
0
Ready
>
```

BREAK
Befehl

Syntax:

BREAK <Zeile>

Beschreibung:

Verlässt die aktuell aktive Schleife vorzeitig und springt zu einer anderen Programmposition.

Beispiel:

```
>10 DIM CNT
>20 FOR CNT=1 TO 10
>30 PRINT "Durchlauf ";CNT
>40 IF CNT=5 THEN BREAK 60
>50 NEXT
>60 PRINT "Und Ende!"
>RUN
Durchlauf 1
Durchlauf 2
Durchlauf 3
Durchlauf 4
Durchlauf 5
Und Ende!
Ready
>
```

C

CASE

Befehl

Syntax:

CASE <Wert>:{Befehle}

Beschreibung:

Vergleicht den mit dem SELECT-Befehl festgelegten Wert mit dem Angegebenen. Wird Gleichheit festgestellt, werden die Befehle in der aktuellen Zeile ausgeführt. Wenn keine Gleichheit vorhanden ist, wird der Rest der Zeile ignoriert und mit der nächsten verfügbaren Programmzeile fortgefahren.

Beispiel:

```
>10 DIM NOTE
>20 INPUT "Note : ",NOTE:SELECT NOTE
>30 CASE 1:PRINT "Sehr gut"
>40 CASE 2:PRINT "Gut"
>50 CASE 3:PRINT "Befriedigend"
>60 CASE 4:PRINT "Ausreichend"
>70 CASE 5:PRINT "Mangelhaft"
>80 CASE 6:PRINT "Ungenuegend"
>90 CASEELSE:PRINT "Note nicht erlaubt!"
>RUN
Note : 3
Befriedigend
Ready
>
```

CASEELSE
Befehl

Syntax:

CASEELSE <Wert>:{Befehle}

Beschreibung:

Der Rest der Zeile wird ausgeführt wenn die vorhergehenden CASE-Anweisungen nicht zum Erfolg geführt haben. Würde eine frühere CASE-Anweisung bei der aktuellen Tabellanabfrage ausgeführt, ignoriert CASEELSE die aktuelle Zeile und führt das Programm normal in der nächsten Zeile fort.

Beispiel:

```
>10 DIM NOTE
>20 INPUT "Note : ",NOTE:SELECT NOTE
>30 CASE 1:PRINT "Sehr gut"
>40 CASE 2:PRINT "Gut"
>50 CASE 3:PRINT "Befriedigend"
>60 CASE 4:PRINT "Ausreichend"
>70 CASE 5:PRINT "Mangelhaft"
>80 CASE 6:PRINT "Ungenuegend"
>90 CASEELSE:PRINT "Note nicht erlaubt!"
>RUN
Note : 3
Befriedigend
Ready
>
```

CAT
Befehl

Syntax:

CAT

Beschreibung:

Listet das Inhaltsverzeichnis des aktuell ausgewählten Speichermoduls auf.

Beispiel:

```
>CAT
Drive 2
  MINI-LA   .PRG  5394
  TESTPROG  .PRG  117
  IO-TEST   .PRG  463
3 files
59514 bytes free
Ready
>
```

CHAIN
Befehl

Syntax:

CHAIN <Dateiname>

Beschreibung:

CHAIN ermöglicht das nachladen von Programmteilen. Das aktuell im Speicher befindliche Programm wird hierbei gelöscht. Die Variablen jedoch, welche aktuell definiert sind, bleiben vollständig erhalten. Das nachgeladene Programm wird automatisch gestartet. Dadurch sind sehr komplexe Programme und Programmsysteme möglich.

Beispiel:

(Hauptprogramm 'START.PRG' auf Speichermodul speichern)

```
10 DIM V$:V$="TEST":CHAIN "MAIN.CHN"
```

(Nachzuladender Programmteil unter 'MAIN.CHN' speichern)

```
10 PRINT "Dies ist ein ";V$
```

```
>RUN "START"
Dies ist ein Test
Ready
>
```

CHANGE$
Funktion

Syntax:

<Neustring>=CHANGE$(<Alt>,<Neu>,<String>)

Beschreibung:

Ersetzt bestimmte Zeichen in einem String durch andere und gibt den so umgewandelten String zurück. Hierbei können die alten sowie die neuen Zeichen als Ascii-Werte oder als String übergeben werden. Bei Übergabe als String wird nur das erste Zeichen berücksichtigt.

Beispiel:

```
>PRINT CHANGE$("e","*","Dies ist der BasicBeetle")
Di*s ist d*r BasicB**tl*
Ready
>
```

CHR$
Funktion

Syntax:

<Zeichen>=CHR$(<Zeichennummer>)

Beschreibung:

Gibt das Zeichen als String zurück, welches zu der angegebenen Nummer gehört.

Beispiel:

```
>PRINT CHR$(65)
A
Ready
>
```

CLEAR
Befehl

Syntax:

CLEAR

Beschreibung:

Löscht alle vorhandenen Variablen. Sollen nur einzelne Variablen aus dem Speicher entfernt werden, muss der Befehl 'ERASE' verwendet werden.

Beispiel:

```
>PRINT N:CLEAR
543
Ready
>PRINT N

Error 1: Syntax error
Ready
>
```

CLS
Befehl

Syntax:

CLS

Beschreibung:

Sendet das Ascii-Zeichen 12 (0C Hex) an das angeschlossene Terminal. Dieses sollte dies Zeichen als Befehl zum Bildschirmlöschen verstehen.

Beispiel:

```
>CLS
```
(Löscht den Bildschirm des Terminals)
```
Ready
>
```

CONSOLE
Variable

Syntax:

CONSOLE=<Status>
<Status>=CONSOLE

Beschreibung:

Mit der Variablen CONSOLE ist es möglich die Ausgaben vom Befehl PRINT, LIST und CAT auf das Terminal umzuleiten. Hierzu muss der Variablen CONSOLE einen Wert ungleich 0 zugewiesen werden. Bei Zuweisung einer 0 wird die Ausgabe an den, an dem am parallelen Druckerport angeschlossenen Drucker, ausgegeben. Standardmäßig ist die Terminalausgabe aktiviert.

Soll der aktuelle Status abgefragt werden, gibt CONSOLE True (=65535) zurück, wenn zur Zeit der Terminalmodus aktiv ist. Andernfalls wird False (=0) zurück geliefert.

Beispiel:

>**CONSELE=FALSE:LIST:CONSOLE=TRUE**
(Das aktuelle Programm wird auf den Drucker ausgegeben)
Ready
>

COPY
Befehl

Syntax:

COPY <Datei>,<Ziel>

Beschreibung:

Kopiert die angegebene Datei des aktuellen Speichermoduls auf das angegebene Ziel-Speichermodul. Wird die Dateierweiterung weggelassen, wird als Standard '.PRG' angenommen.

Beispiel:

>**COPY "TESTPROG",2**
Ready
>

D

DATA
Befehl

Syntax:

DATA <Wert>[,<Wert>]...

Beschreibung:

Definiert Daten, welche mit dem READ-Befehl ausgelesen werden können. Der Datentyp muss mit den Daten des READ-Befehls übereinstimmen. Hierbei sind auch Strings oder Berechnungen möglich. Berechnungen werden, bevor sie dem READ-Befehl übergeben werden, aufgelöst. Soll eine Zeile als DATA-Zeile deklariert werden so muss der DATA-Befehl direkt hinter der Zeilennummer erfolgen.

Wurde ein Datenwert ausgelesen, wird der Zeiger auf den nächsten Datenwert gesetzt. Sind keine Datenwerte mehr vorhanden, wird ein Fehler ausgelöst. DATA-Zeilen können an jeder beliebigen Position im Programm stehen. Ebenso können DATA-Zeilen durch Anweisungen unterbrochen werden.

Trifft der BasicBeetle beim Ausführen des Programms auf DATA-Anweisungen, werden die entsprechenden Zeilen ignoriert.

Beispiel:

```
>10 DIM VNAME$,NNAME$,ALTER
>20 RESTORE 100
>30 READ VNAME$,NNAME$,ALTER
>40 PRINT VNAME$;" ";NNAME$;", ";ALTER;" Jahre":END
>100 DATA "Hans","Heinrich",2009-1970
RUN
Hans Heinrich, 39 Jahre
Ready
>
```

DEC
Befehl

Syntax:

DEC <Variable>

Beschreibung:

Verringert den Wert der angegebenen Variablen um 1. Es findet keine Überlaufprüfung statt.

Beispiel:

```
>DIM WERT:WERT=13:DEC WERT:PRINT WERT
12
Ready
>
```

DELAY
Befehl

Syntax:

DELAY <Wartezeit>

Beschreibung:

Hält das Programm für die angegebene Zeit in x/100 Sekunden an. Interrupts werden während dieser Zeit auch nicht ausgeführt. Ctrl+C ist während der Ausführung dieses Befehls wirkungslos.

Beispiel:

```
>DELAY 100
(Wartet 1 Sekunde)
Ready
>
```

DEVCOPY
Befehl

Syntax:

DEVCOPY <Ziel>

Beschreibung:

Kopiert das momentan aktive Speichermodul auf ein anderes im angegeben Ziellaufwerk. Es ist zu beachten, dass sämtliche Daten auf dem Ziel-Speichermodul überschrieben werden.

Beispiel:

```
>DEVCOPY 2
Ready
>
```

DI
Befehl

Syntax:

DI

Beschreibung:

Verbietet das Ausführung von Interrupts. Ausgelöste Interrupts werden ignoriert. Die Sperre kann durch den Befehl EI oder durch eine Neudefinition eines AFTER oder EVERY-Befehls aufgehoben werden.

Beispiel:

```
>10 EVERY 100,1 GOSUB 100
>20 DI
>30 GOTO 30
>100 PRINT "*";:RETURN
Ready
>RUN
(Es erfolgt keine Ausgabe)
```

DIM
Befehl

Syntax:

DIM <Var>[(<Anzahl>)][*<Länge>][,...<Var>[(Anzahl>)][*<Länge>]]

Beschreibung:

Definiert einfache oder Feldvariablen. Durch Anhängen eines Identifikationszeichens an den Variablennamen wird der Datentyp der Variablen bestimmt. Hierbei sind:

&: Byte, 1 Byte Größe, speichert Werte von 0-255
#: Wort, 2 Byte Größe, speichert Werte von 0-65535
@: Quad-Wort, 4 Byte Größe, speichert Werte von 0-4294967295
$: Strings, 1-256 Byte Größe, speichert Strings von 0-255 Zeichen

Wird das Identifikationszeichen hinter dem Variablennamen weggelassen, definiert der Computer automatisch Variablen vom Typ Wort.

Wird eine String-Variable definiert, so wird eine Standard-Maximallänge des möglichen Strings von 16 Zeichen angenommen. Soll eine andere Größe definiert werden, so muss die gewünschte Länge mit '*Länge' hinter dem Variablennamen angegeben werden.

Sollen Feldvariablen definiert werden, so muss die gewünschte Anzahl in runden Klammern hinter dem Variablennamen angegeben werden. Hierbei ist zu beachten, dass die Elementezählung mit 0 beginnt.

Beispiel:

```
>DIM VARBYTE&,VARWORD1,VARWORD2#,VARQUAD@,VARSTRING1$,VARSTRING2$*40
Ready
>DIM FELDBYTE&(9),FELDWORD1(9),FELDWORD2#(9),FELDQUAD@(9)
Ready
>DIM FELDSTRING1$(9),FELDSTRING2$(9)*10
Ready
>
```

DIR$
Funktion

Syntax:

<Eintrag>=DIR$(<Dateinummer>)

Beschreibung:

Ermittelt den Namen und die Dateigröße einer Datei mit der angegebenen Positionsnummer auf dem aktuellen Laufwerk. Existiert keine Datei mehr an der angegebenen Position, wird ein Leerstring zurück gegeben.

Der String besteht aus der Dateigröße, Komma und gefolgt von dem Dateinamen.

Beispiel:

```
>PRINT DIR$(1)
572,LPTTEST.PRG
Ready
>
```

Division, /

Verknüpfung

Syntax:

<Ergebnis>=<Wert1>/<Wert2>

Beschreibung:

Bildet den Quotienten beider angegebenen Werte.

Beispiel:

```
>PRINT 34/17
2
Ready
>
```

DO

Befehl

Syntax:

DO - LOOP
DO - UNTIL <Bedingung>
DO - WHILE <Bedingung>

Beschreibung:

Eröffnet eine Programmschleife, welche nie (Abschluss mit LOOP), bei wahrer Bedingung (Abschluss mit UNTIL) oder bei falscher Bedingung (Abschluss mit WHILE) beendet wird.

Mehrere DO-Schleifen können geschachtelt werden.

Beispiel:

```
>10 DIM CNT
>20 DO:PRINT "*";:INC CNT:WHILE CNT<10
>RUN
*********
Ready
>
```

DRIVE
Variable

Syntax:

<Laufwerk>=DRIVE
DRIVE=<Laufwerk>

Beschreibung:

Setzt oder liefert das aktuelle EEPROM-Laufwerk. Es können die Parameter 0-3 angegeben werden. Wird ein Laufwerk gesetzt, erfolgen die nächsten Zugriffe nur auf diesem.

Beispiel:

```
>DRIVE=2
Ready
>PRINT DRIVE
2
Ready
>
```

E

ELSE
Befehl

Syntax:

IF <Bedingung> THEN <Wahr-Befehle> [ELSE <Falsch-Befehle>]

Beschreibung:

Die folgendende Befehle werden nur ausgeführt, wenn der vorangegangene IF-Befehl eine falsche Bedingung oder der vorangegangene IFF-Befehl eine wahre Bedingung ermittelt hat.

Beispiel:

```
>IF 5=5 THEN PRINT "Stimmt"
Stimmt
Ready
>IF 5=3 THEN PRINT "Stimmt"
Ready
>IF 5=3 THEN PRINT "Stimmt" ELSE PRINT "Stimmt nicht"
Stimmt nicht
Ready
>
```

END
Befehl

Syntax:

END

Beschreibung:

Stoppt das laufende Programm und kehrt in den Direktmodus zurück.

Beispiel:

```
>10 PRINT "Nur ein Befehl":END
>20 PRINT "und noch ein Befehl"
>RUN
Nur ein Befehl
Ready
>
```

ENDREPEAT
Befehl

Syntax:

REPEAT <Anzahl> - ENDREPEAT

Beschreibung:

Schliesst eine Schleife, welche mit dem REPEAT-Befehl geöffnet wurde, ab.

Beispiel:

```
>REPEAT 10:PRINT "*";:ENDREPEAT
**********
Ready
>
```

EOF
Funktion

Syntax:

<Dateiende>=EOF

Beschreibung:

Ist das Dateiende bei einer geöffneten lesenden Datendatei erreicht, wird True zurückgeliefert. Andernfalls False.

Beispiel:

```
>10 OPENR "DATEN.DAT":DO
>20 GET WERT(CNT):INC CNT:WHILE NOT(EOF):CLOSER
>RUN
Ready
>
```

ERASE
Befehl

Syntax:

ERASE <Variable1>[,<Variable2>[,<Variable..>]]

Beschreibung:

Entfernt die angegebenen Variablen aus dem Arbeitsspeicher. Bei Feldvariablen muss nur der eigentliche Variablenname angegeben werden.

Beispiel:

```
>10 DIM WERT,FELD(10)
>20 WERT=456:FELD(5)=941
>30 PRINT WERT:PRINT FELD(5)
>40 ERASE WERT,FELD
>50 PRINT WERT:PRINT FELD(5)
>RUN
456
941
Error 1: Syntax Error in 50
Ready
>
```

ERL
Funktion

Syntax:

<Fehlerzeile>=ERL

Beschreibung:

Gibt die Zeile zurück in der der letzte Fehler erzeugt wurde. Die Fehlerzeile wird nach Abfrage von ERL gelöscht. Wird diese öfters benötigt, sollte diese einer Variablen zugewiesen werden.

Beispiel:

```
>10 PRIT "Hallo"
>RUN
Error 1: Syntax Error in 10
Ready
>PRINT ERL
10
Ready
>
```

ERR
Funktion

Syntax:

<Fehlernummer>=ERR

Beschreibung:

Gibt die letzte erzeugte Fehlernummer zurück. Die Fehlernummer wird nach Abfrage von ERR gelöscht. Wird diese öfters benötigt, sollte diese einer Variablen zugewiesen werden.

Beispiel:

```
>PRIT "Hallo"
Error 1: Syntax Error
Ready
>PRINT ERR
1
Ready
>
```

ERROR
Befehl

Syntax:

ERROR <Fehlernummer>

Beschreibung:

Erzeugt eine Fehlermeldung mit der angegebenen Nummer. Hierdurch ist es möglich, programmgesteuert bestimmte Fehler zu erzeugen.

Beispiel:

```
>ERROR 1
Error 1: Syntax Error
Ready
>
```

EVERY - GOSUB
Befehl

Syntax:

EVERY <Zeit>,<Timer> GOSUB <Zeile>

Beschreibung:

Startet einen, sich immer wiederholenden, Timer-Interrupt. Die angegebene Zeile wird in regelmäßigen Zeitabständen, angegeben in 1/100tel Sekunden, angesprungen. Hierbei sind 3 unterschiedliche Interrupts möglich (Timer 1-3).

Bei einer Zeitangabe von 0 wird der angegebene Timer-Interrupt gestoppt. Der Zeitwert darf 65535/100tel Sekunden nicht überschreiten.

Die 3 Timer des EVERY-Befehls arbeiten unabhängig von den 3 AFTER-Timern. Es sind also jeweils 3 Timer möglich.

Beispiel:

```
>10 EVERY 100,1 GOSUB 50
>20 DO:LOOP
>50 PRINT "*";:RETURN
>RUN
*****  (Jede Sekunde erscheint ein neues Zeichen)
```

EXIST
Funktion

Syntax:

<Vorhanden>=EXIST(<Dateiname>)

Beschreibung:

Gibt Wahr (=65535) zurück, wenn die angegebene Datei auf dem aktiven Speichermodul vorhanden ist. Andernfalls wird Falsch (=0) zurück gegeben.

Beispiel:

```
>PRINT EXIST("ANYONE.PRG")
0
Ready
>
```

F

FALSE
Funktion

Syntax:

65535=FALSE

Beschreibung:

Gibt den Wert für False (=0) zurück.

Beispiel:

```
>PRINT FALSE
0
Ready
>
```

FAST
Befehl

Syntax:

FAST

Beschreibung:

Aktiviert für den System-Bus eine Taktfrequenz von ca. 400 kHz. Mit dem System-Bus werden z.B. die Speichermodule und der Drucker gesteuert. Beim Neustart des BasicBeetle ist die Taktfrequenz des System-Buses ca. 100 kHz (SLOW-Befehl).

Beispiel:

```
>FAST
Ready
>
```

FOR
Befehl

Syntax:

FOR <Variable>=<Startwert> TO <Endwert> [STEP <Weite>] - NEXT

Beschreibung:

Mit FOR wird eine Zählschleife eingeleitet. Einer angegebene Variable wird der Startwert zugewiesen. Die Befehle zwischen dem FOR-Konstrukt und dem NEXT-Befehl werden nun ausgeführt. Beim Ausführen des NEXT-Befehls wird der Variablenwert um 1 erhöht und die Befehle erneut ausgeführt. Dies wiederholt sich sooft, bis die Variable den angegebenen Endwert erreicht hat. Danach fährt das Programm nach dem NEXT-Befehl fort.

Wird eine STEP-Anweisung angegeben, erhöht sich die Variable bei jedem Durchlauf um den angegebenen Wert.

Beispiel:

```
>DIM CNT:FOR CNT=1 TO 5:PRINT "Nummer ";CNT:NEXT
Nummer 1
Nummer 2
Nummer 3
Nummer 4
Nummer 5
Ready
>
```

FORMAT
Befehl

Syntax:

FORMAT

Beschreibung:

Formatiert das Speichermodul, welches sich zur Zeit im aktiven Laufwerk befindet. Es werden sämtliche Dateien gelöscht. Eine Sicherheitsabfrage findet nicht statt. Es muss daher sorgfältig geprüft werden, ob das richtige Speichermodul sich im aktiven Slot befindet.

Beispiel:

```
>FORMAT
(Das Formatieren kann bis zu 1,5 Minuten dauern)
Ready
>
```

FREE
Funktion

Syntax:

<Speicher>=FREE

Beschreibung:

Ermittelt die Größe des noch frei verfügbaren Arbeitsspeichers. Der XRAM-Bereich bleibt hierbei unberücksichtigt.

Beispiel:

```
>PRINT FREE
28734
Ready
>
```

FREEDISC
Funktion

Syntax:

<Speicher>=FREEDISC

Beschreibung:

Berechnet den noch freien Speicher des aktiven Speichermoduls.

Beispiel:

```
>PRINT FREEDISC
15824
Ready
>
```

G

GET
Befehl

Syntax:

GET <Var>[,<Var>[,...]]

Beschreibung:

Liest Daten aus einer geöffneten Lesedatei und schreibt diese in die angegebenen Variable. Es ist darauf zu achten, dass der gleiche Datentyp eingelesen wird, welcher zuvor mit dem PUT-Befehl geschrieben wurde. Mehrere Variablen können, durch Komma getrennt, angegeben werden.

Beispiel:

```
>GET WERT
Ready
>
```

GODIR
Befehl

Syntax:

GODIR <Zeilenadresse>

Beschreibung:

Springt zu der Programmzeile, welche sich im Arbeitsspeicher ab der angegebenen Adresse befindet. Es muss sichergestellt sein, dass sich die Adresse auch auf den Anfang einer Programmzeile bezieht. Ansonsten kommt es zu unvorhersagbaren Reaktionen. Die Zeilenadresse einer Programmzeile kann man mit der LINEPOS-Funktion ermitteln.

Dieser Befehl ist hilfreich, um eine hohe Geschwindigkeit bei größeren Programmen zu erreichen da hier das suchen der Zeile entfällt.

Beispiel:

```
>10 DIM ADR:ADR=LINEPOS(20)
>20 PRINT "#";:DELAY 10:GODIR ADR
>RUN
########  (Wird unendlich fortgeführt)
```

GOSUB
Befehl

Syntax:

GOSUB <Zeilennummer>

Beschreibung:

Ruft ein Unterprogramm auf, welches sich ab der angegebenen Zeilennummer im Speicher befindet. Der Unterroutine muss mit RETURN wieder verlassen werden. Das Programm wird dann nach dem GOSUB-Befehl weiter abgearbeitet.

Beispiel:

```
>10 PRINT "Hallo ";:GOSUB 50
>20 END
>50 PRINT "User":RETURN
>RUN
Hallo User
Ready
>
```

GOTO
Befehl

Syntax:

GOTO <Zeilennummer>

Beschreibung:

Springt die angegebene Zeilennummer innerhalb des Programms an. Die Zeilennummer kann auch aus einer Variablen bestehen.

Beispiel:

```
>10 PRINT "*";
>20 GOTO 10
>RUN
******* (Wird unendlich fortgeführt. Kann nur durch Reset oder CTRL+C beendet werden)
```

H

HEX$
Funktion

Syntax:

<Hexstring>=HEX$(<Wert>)

Beschreibung:

Erstellt eine Zeichenkette, welche den angegebenen Wert als Hexadezimalwert in einem String zurück gibt.

Beispiel:

```
>PRINT HEX$(570)
23A
Ready
>
```

HI
Funktion

Syntax:

<MSB>=HI(<Wort>)

Beschreibung:

Liefert das höherwertige Byte (MSB) des angegebenen Wortes zurück.

Beispiel:

```
>PRINT HI(570)
2
Ready
>
```

HIDE
Befehl

Syntax:

HIDE <Dateiname>

Beschreibung:

Markiert die angegebene Datei als unsichtbar. Eine so markierte Datei taucht weder beim CAT-Befehl noch bei der DIR$-Funktion auf. Diese Datei kann aber wie gewohnt geladen oder geöffnet werden. Wird beim Dateinamen keine Extension angegeben, so wird '.PRG' angenommen.

Soll eine als unsichtbar markierte Datei wieder sichtbar gemacht werden, so muss der UNHIDE-Befehl angewendet werden.

Beispiel:

```
>CAT
Drive 2
  MINI-LA   .PRG  5394
  TESTPROG  .PRG  117
  IO-TEST   .PRG  463
3 files
59514 bytes free
Ready
>HIDE "TESTPROG"
Ready
>CAT
Drive 2
  MINI-LA   .PRG  5394
  IO-TEST   .PRG  463
2 files
59514 bytes free
Ready
>
```

HIMEM
Funktion

Syntax:

<Speicher>=HIMEM

Beschreibung:

Die höchste Speicheradresse, welche das Basic für Programme und Variablen verwenden darf, wird mit HIMEM zurück geliefert. Wurde der Speicher nicht mit dem MEMORY-Befehl begrenzt, enthält HIMEM den maximal verfügbaren Arbeitsspeicher. Der XRAM-Bereich wird hierbei nicht berücksichtigt.

Beispiel:

```
>PRINT HIMEM
65535
Ready
>
```

HIWORD
Funktion

Syntax:

<MSB>=HIWORD(<Wert>)

Beschreibung:

Liefert das höherwertige Wort (MSB) des angegebenen Quad-Wortes zurück.

Beispiel:

```
>PRINT HIWORD(570000)
8
Ready
>
```

HOME
Befehl

Syntax:

HOME

Beschreibung:

Sendet das Ascii-Zeichen 11 (0B Hex) an das angeschlossene Terminal. Dieses sollte dies Zeichen als Befehl zum Setzen des Cursors an die linke obere Bildschirmecke verstehen.

Beispiel:

```
>HOME
(Cursor springt in die erste Zeile)
Ready
>
```

I

IF
Befehl

Syntax:

IF <Bedingung> THEN <Wahr-Befehle> [ELSE <Falsch-Befehle>]

Beschreibung:

Prüft ob die angegebene Bedingung Wahr (Wert ist nicht 0) ist und führt davon abhängig die Befehle nach dem THEN-Statement aus.

Liefert die Bedingung das Ergebnis Falsch (Wert ist 0) zurück werden die Befehle nach dem ELSE-Statement ausgeführt, sofern dieser angegeben wurde.

Beispiel:

```
>IF 5=5 THEN PRINT "Stimmt"
Stimmt
Ready
>IF 5=3 THEN PRINT "Stimmt"
Ready
>IF 5=3 THEN PRINT "Stimmt" ELSE PRINT "Stimmt nicht"
Stimmt nicht
Ready
>
```

IFF
Befehl

Syntax:

IFF <Bedingung> THEN <Falsch-Befehle> [ELSE <Wahr-Befehle>]

Beschreibung:

Prüft ob die angegebene Bedingung Falsch (Wert ist 0) ist und führt davon abhängig die Befehle nach dem THEN-Statement aus.

Liefert die Bedingung das Ergebnis Wahr (Wert ist nicht 0) zurück werden die Befehle nach dem ELSE-Statement ausgeführt, sofern dieser angegeben wurde.

Beispiel:

```
>IFF 5=5 THEN PRINT "Stimmt nicht"
Ready
>IFF 5=3 THEN PRINT "Stimmt nicht"
Stimmt nicht
Ready
>IFF 5=5 THEN PRINT "Stimmt nicht" ELSE PRINT "Stimmt"
Stimmt
Ready
>
```

IICREAD
Funktion

Syntax:

<Wert>=IICREAD(<Acknowledge>)

Beschreibung:

Liest ein Wert von einem Baustein, welches zuvor mit IICWRITE adressiert wurde. Dieser Baustein muss am Systembus angeschlossen werden.

Mit Hilfe des Parameters wird bestimmt, ob ein Acknowledge-Signal (Parameter=True) oder ein No Acknowledge (Parameter=False) gesendet werden soll. Wird ein Acknowledge gesendet, können einige IIC-Bausteine Daten blockweise zum Beetle senden.

Beispiel:

```
>IICSTART:IICWRITE &HF0:PRINT IICREAD(FALSE):IICSTOP
52
Ready
>
```

IICSTART
Befehl

Syntax:

IICSTART

Beschreibung:

Startet die Kommunikation auf dem Systembus. Es ist zu beachten, dass nach der Ausführung von IICSTART Zugriffe auf die systemeigenen Komponenten mit BasicBeetle-internen Befehlen evtl. mit einem Fehler quittiert werden.

Beispiel:

```
>IICSTART:IICWRITE &HF0:IICWRITE 0:IICSTOP
Ready
>
```

IICSTOP
Befehl

Syntax:

IICSTOP

Beschreibung:

Schließt den Systemeigenen IIC-Bus. Die Kommunikation mit Systemmodulen ist nun wieder problemlos möglich.

Beispiel:

```
>IICSTART:IICWRITE &HF0:IICWRITE 0:IICSTOP
Ready
>
```

IICWRITE
Befehl

Syntax:

IICWRITE <Wert>

Beschreibung:

Schreibt den angegebenen Wert auf den systemeigenen IIC-Bus. Gibt es keinen passenden Empfänger für diesen Wert, erzeugt der BasicBeetle eine Fehlermeldung.

Beispiel:

```
>IICSTART:IICWRITE &HF0:IICWRITE 0:IICSTOP
Ready
>
```

IN
Funktion

Syntax:

<Wert>=IN(<Portadresse>)

Beschreibung:

Liest ein Byte vom Erweiterungsmodul mit der angegebenen Portadresse. Werte und genaue Adressen sind vom entsprechenden Modul abhängig.

Beispiel:

```
>PRINT IN(64)
57
Ready
>
```

INC
Befehl

Syntax:

INC <Variable>

Beschreibung:

Erhöht den Wert der angegebenen Variablen um 1. Es findet keine Überlaufprüfung statt.

Beispiel:

```
>DIM WERT:WERT=12:INC WERT:PRINT WERT
13
Ready
>
```

INKEY$
Funktion

Syntax:

<Taste>=INKEY$

Beschreibung:

Überprüft, ob eine Tastenbetätigung über die serielle Schnittstelle gesendet wurde. Gab es keine Tastenbetätigung, wird ein Leerstring zurück geliefert. Andernfalls liefert INKEY$ ein String mit betätigter Taste.

Beispiel:

>**DIM K$:DO:K$=INKEY$:WHILE K$="":PRINT K$**
(Z.b. Taste 'g' betätigen)
g
Ready
>

INPUT
Befehl

Syntax:

INPUT <Ausgabetext>,<Variable>

Beschreibung:

Gibt den <Ausgabetext> auf dem Terminal aus und erwartet vom Benutzer eine Eingabe. Die Eingabe muss mit [Enter] abgeschlossen werden und wird dann in der angegebenen Variable gespeichert. Wurde eine Eingabe vorgenommen, welche nicht in die angegebene Variable zu speichern gehen, fordert INPUT erneut zur Eingabe auf.

Beispiel:

```
>10 DIM N$*20
>20 INPUT "Dein Name : ",N$
>30 PRINT "Hallo ";N$
>RUN
Dein Name : Hans-Peter
Hallo Hans-Peter
Ready
>
```

INSTR
Funktion

Syntax:

<Position>=INSTR(<Suchzeichen>,<String>)

Beschreibung:

Sucht ein Zeichen im angegebenen String. Wurde dieses Zeichen gefunden, wird die Position zurückgegeben. Gibt es keinen Fund, wird 0 zurück geliefert.

Beispiel:

```
>PRINT INSTR("U","Hallo User")
7
Ready
>
```

KILL
Befehl

Syntax:

KILL <Dateiname>

Beschreibung:

Markiert die angegebene Datei als gelöscht. Die Datei bleibt noch unsichtbar auf dem Speichermodul erhalten, bis der PACK-Befehl ausgeführt wird.

Wird beim Dateinamen keine Extension angegeben, wird automatisch .PRG hinzugefügt.

Beispiel:

```
>CAT
Drive 2
  MINI-LA   .PRG  5394
  TESTPROG  .PRG  117
  IO-TEST   .PRG  463
3 files
59514 bytes free
Ready
>KILL "TESTPROG"
Ready
>CAT
Drive 2
  MINI-LA   .PRG  5394
  IO-TEST   .PRG  463
2 files
59514 bytes free
Ready
>
```

L

LEFT$
Funktion

Syntax:
<String>=LEFT$(<Anzahl>,<Quellstring>)

Beschreibung:
Liefert eine bestimmte Anzahl von Zeichen von der linken Seite des Quellstrings zurück.

Beispiel:
```
>PRINT LEFT$(5,"Hallo User")
Hallo
Ready
>
```

LEN
Funktion

Syntax:
<Länge>=LEN(<String>)

Beschreibung:
Ermittelt die Länge des angegebenen Strings und liefert diese zurück.

Beispiel:
```
>PRINT LEN("Hallo User")
10
Ready
>
```

LINE
Funktion

Syntax:
<Zeile>=LINE

Beschreibung:
Liefert die aktuell aktive Zeilennummer zurück. Durch die LINE-Funktion ist es möglich, relozierbare Programmteile zu erstellen.

Beispiel:
```
>10 PRINT "#";:GOTO LINE
>RUN
########  (Wird unendlich fortgeführt)
```

LINEPOS
Funktion

Syntax:

<Adresse>=LINEPOS(<Zeile>)

Beschreibung:

Gibt die Speicheradresse von der angegeben Zeile zurück. Dieser Wert wird für die GODIR und SUBDIR-Befehle benötigt.

Beispiel:

```
>10 DIM ADR:ADR=LINEPOS(20)
>20 PRINT "#";:DELAY 10:GODIR ADR
>RUN
########  (Wird unendlich fortgeführt)
```

LIST
Befehl

Syntax:

LIST [<Zeilenbereich>]

Beschreibung:

Listet das Programm oder Teile davon, welches sich zur Zeit im Arbeitsspeicher befindet, auf. Wird ein Zeilenbereich, in der Form: [Von]-[Bis] angegeben, wird nur der gewünschte Teil aufgelistet. Beim weglassen des Zeilenbereichs zeigt LIST das ganze Programm.

Beispiel:

```
>LIST
(Das ganze Programm wird gezeigt)
Ready
>LIST -100
(Es wird bis Zeile 100 aufgelistet)
Ready
>LIST 10000-
(Das Programm ab Zeile 10000 wird gezeigt)
Ready
>LIST 500-1000
(Die Zeilen zwischen 500 bis 1000 werden ausgegeben)
Ready
>LIST 100
(Nur Zeile 100 wird dargestellt)
Ready
>
```

LO
Funktion

Syntax:

<LSB>=LO(<Wort>)

Beschreibung:

Ermittelt das niederwertigere (LSB) Byte des angegebenen Wortes und liefert dieses zurück.

Beispiel:

```
>PRINT LO(8765)
61
Ready
>
```

LOAD
Befehl

Syntax:

LOAD <Dateiname>

Beschreibung:

Lädt ein Programm vom aktiven Speichermodul mit dem angegeben Dateinamen in den Arbeitsspeicher. Wird die Extension weggelassen, fügt der BasicBeetle automatisch .PRG hinzu.

Beispiel:

```
>LOAD "TESTPROG"
Ready
>
```

LOADBIN
Befehl

Syntax:

LOADBIN <Zieladresse>,<Dateiname>

Beschreibung:

Lädt eine Binärdatei vom aktiven Speichermodul mit dem angegeben Dateinamen an die angegebene Adresse im Arbeitsspeicher. Hierbei wird nur in den Hauptspeicher geladen. Wird die Extension weggelassen, fügt der BasicBeetle automatisch .BIN hinzu.

Beispiel:

```
>MEMORY &H8000:LOADBIN &H8000,"AD-TAB"
Ready
>
```

LOMEM
Funktion

Syntax:

<Speicheranfang>=LOMEM

Beschreibung:

Ergibt den Beginn des, für Anwenderprogramme und Variablen, vorhandenen Arbeitsspeichers. Unterhalb dieser Adresse sollte man in keinem Fall hinein schreiben z.B. mit dem POKE-Befehl. Dies kann unter Umständen zum Absturz des BasicBeetle führen.

Beispiel:

```
>PRINT LOMEM
1280
Ready
>
```

LOOP
Befehl

Syntax:

DO - LOOP

Beschreibung:

Beendet eine mit DO eröffnete Schleife. Diese Schleife wird unendlich wiederholt. Soll diese verlassen werden, muss der BREAK-Befehl angewandt werden.

.

Beispiel:

```
>10 DO:PRINT "*";:LOOP
>RUN
********* (wird unendlich fortgeführt)
```

LOWER$
Funktion

Syntax:

<String>=LOWER$(<String>)

Beschreibung:

Wandelt den übergebenen String in Kleinbuchstaben um. Umlaute bzw. länderspezifische Sonderzeichen werden nicht berücksichtigt.

Beispiel:

```
>PRINT LOWER$("Hallo User")
hallo user
Ready
>
```

LOWORD
Funktion

Syntax:

<LSB>=LOWORD(<Wert>)

Beschreibung:

Liefert das niederwertigere Wort (LSB) des angegebenen Quad-Wortes zurück.

Beispiel:

```
>PRINT LOWORD(570000)
45712
Ready
>
```

LTRIM$
Funktion

Syntax:

<String>=LTRIM$(<Zeichen>,<String>)

Beschreibung:

Entfernt die eventuellen angegebenen führenden Zeichen vom String und liefert den Rest zurück. Das gewünschte Zeichen kann als Ascii-Nummer oder als String angegeben werden. Bei der Angabe als String wird nur das erste Zeichen verwendet.

Beispiel:

```
>PRINT LTRIM$("*",RTRIM$("*","***BasicBeetle***"))
BasicBeetle
Ready
>
```

M

MAX
Funktion

Syntax:

<Maximalwert>=MAX(<Wert1>,<Wert2>[,...<Wertn>])

Beschreibung:

Gibt den größten Wert aus einer Liste von diversen Werten zurück.

Beispiel:

```
>PRINT MAX(57,17,325)
325
Ready
>
```

MEMORY
Befehl

Syntax:

MEMORY <Speichergrenze>

Beschreibung:

Legt eine neue Speichergrenze fest. Es wird hierbei nicht geprüft, ob der Speicher real existiert. Wird die Speichergrenze so niedrig angesetzt, dass dieser mit dem aktuellen Programm oder den Daten kollidiert, wird ein Fehler erzeugt. XRAM-Speicher bleibt hierbei unberücksichtigt.

Mit MEMORY ist es möglich, Speicher für Binärdaten zu reservieren, welcher vom BeetleBasic nicht angetastet wird.

Beispiel:

```
>MEMORY 32767:PRINT FREE
31487
Ready
>
```

MID$
Funktion

Syntax:

<Ausschnitt>=MID$(<Start>,<Anzahl>,<String>)

Beschreibung:

Liefert einen Teil eines Strings zurück. Start gibt hierbei an, ab welchem Zeichen zurückgegeben werden soll. Die Anzahl schreibt die Menge der Zeichen vor, welche ab dem Start zurückgeliefert werden soll.

Ist Start größer als die Länge des aufgeführten Strings, resultiert daraus eine leere Zeichenkette.

Ergibt sich mit Start und der Anzahl eine größere Stringlänge als im angegebenen String vorhanden ist, werden nur die vorhanden Zeichen ausgeliefert.

Beispiel:

```
>PRINT MID$(6,3,"Dies ist der BasicBeetle")
ist
Ready
>
```

MIN
Funktion

Syntax:

<Minimalwert>=MIN(<Wert1>,<Wert2>[,...<Wertn>])

Beschreibung:

Gibt den kleinsten Wert aus einer Liste von diversen Werten zurück.

Beispiel:

```
>PRINT MIN(57,17,325)
17
Ready
>
```

MOD / %
Operator

Syntax:

<Rest>=<Dividend> MOD <Divisor>
<Rest>=<Dividend>%<Divisor>

Beschreibung:

Berechnet den Restwert bei einer Quotientenbildung.

Beispiel:

```
>PRINT 11 MOD 4
3
Ready
>
```

Multiplikation, *

Verknüpfung

Syntax:

<Ergebnis>=<Wert1>*<Wert2>

Beschreibung:

Bildet das Produkt beider angegebenen Werte.

Beispiel:

```
>PRINT 34*17
578
Ready
>
```

N

NEW
Befehl

Syntax:

NEW

Beschreibung:

Löscht das aktuelle Programm aus dem Arbeitsspeicher. Auch vorher definierte Variablen werden gelöscht.

Beispiel:

```
>LIST
10 PRINT "Hallo User"
Ready
>NEW
Ready
>LIST
Ready
>
```

NEXT
Befehl

Syntax:

FOR <Variable>=<Startwert> TO <Endwert> [STEP <Weite>] - NEXT

Beschreibung:

Schließt eine FOR-Zählschleife ab. Die Befehle ab dem FOR-Befehl werden wiederholt, wenn die angegebene Zählvariable den Endwert noch nicht erreicht hat.

Beispiel:

```
>DIM CNT:FOR CNT=1 TO 5:PRINT "Nummer ";CNT:NEXT
Nummer 1
Nummer 2
Nummer 3
Nummer 4
Nummer 5
Ready
>
```

NUM
Funktion

Syntax:

<Ergebnis>=NUM(<Zahlstring>)

Beschreibung:

Überprüft den angegebenen String darauf, ob ein umwandelbarer numerischer Wert enthalten ist. Es wird TRUE zurück geliefert, wenn die Prüfung positiv verläuft.

Beispiel:

```
>PRINT NUM("765")
65535
Ready
>PRINT NUM("Hallo")
0
Ready
>
```

O

ON - GOSUB
Befehl

Syntax:

ON <Wert> GOSUB <Zeile 1>[,<Zeile 2>]...[,<Zeile n>]

Beschreibung:

Ruft ein Unterprogramm, abhängig vom angegeben Wert, auf dessen Zeilennummer sich in der angegeben Zeilenliste befindet. Bei einem Wert von 1 springt er zur 1. angegeben Zeile. Bei einem Wert von 2 wird die 2. angegebene Zeile aufgerufen. Wird 0 als Wert angegeben, wird der ON-GOSUB-Befehl ignoriert. Ist der Wert größer als die vorhandenen Sprungziele, so wird eine Fehlermeldung ausgelöst.

Beispiel:

```
>10 DIM WERT
>20 INPUT "Welcher Wert : ",WERT
>30 ON WERT GOSUB 100,200,300
>40 END
>100 PRINT "Erster Eintrag!":RETURN
>200 PRINT "Zweiter Eintrag!":RETURN
>300 PRINT "Dritter Eintrag!":RETURN
>RUN
Welcher Wert : 2
Zweiter Wert
Ready
>
```

ON - GOTO
Befehl

Syntax:

ON <Wert> GOTO <Zeile 1>[,<Zeile 2>]...[,<Zeile n>]

Beschreibung:

Springt eine der Zeilen aus der aufgeführten Zeilenliste in Abhängigkeit vom angegebenen Wert an. Bei einem Wert von 1 springt zur 1. angegeben Zeile. Bei einem Wert von 2 wird die 2. angegebene Zeile aufgerufen. Wird 0 als Wert angegeben, wird der ON-GOTO-Befehl ignoriert. Ist der Wert größer als die vorhandenen Sprungziele, so wird eine Fehlermeldung ausgelöst.

Beispiel:

```
>10 DIM WERT
>20 INPUT "Welcher Wert : ",WERT
>30 ON WERT GOTO 100,200,300
>40 PRINT "Null gewaehlt!":END
>100 PRINT "Erster Eintrag!":END
>200 PRINT "Zweiter Eintrag!":END
>300 PRINT "Dritter Eintrag!":END
>RUN
Welcher Wert : 2
Zweiter Wert
Ready
>
```

ON ERROR GOTO
Befehl

Syntax:

ON ERROR GOTO <Zeilennummer>

Beschreibung:

Definiert eine Fehlerbehandlungsroutine welche sich ab der angegebenen Zeile im Arbeitsspeicher befindet. Tritt nun ein Fehler auf, wird keine Fehlermeldung mehr ausgegeben sondern es wird die angegebene Zeile angesprungen. Mit dem RESUME-Befehl wird der normale Programmablauf fortgeführt.

Soll die Fehlerbehandlung deaktiviert werden, muss als Zeilennummer die Zeile 0 angegeben werden.

Beispiel:

```
>10 ON ERROR GOTO 60000
>20 PRIT Hallo
>30 PRINT "Programmende":END
>60000 PRINT "Es gibt einen Fehler ";ERR;" in Zeile ";ERL
>60010 RESUME 30
>RUN
Es gibt einen Fehler 1 in Zeile 20
Programmende
Ready
>
```

ON KEY GOSUB
Befehl

Syntax:

ON KEY(<Taste>) GOSUB <Zeilennummer>

Beschreibung:

Definiert eine Überwachungstaste. Das Hauptprogramm wird ab diesem Befehl unterbrochen und das Unterprogramm ab der angegebenen Zeile aufgerufen.

Bei einer Tastennummer von 0, wird die Tastenüberwachung abgeschaltet.

Beispiel:

```
>10 ON KEY(27) GOSUB 100
>20 GOTO 20
>100 PRINT "[ESC] wurde betaetigt.":RETURN
>RUN
([ESC] drücken)
[ESC] wurde betaetigt.
```

OPENR
Befehl

Syntax:
OPENR <Dateiname>

Beschreibung:
Öffnet die angegebene Datendatei auf dem aktuellen Speichermodul für Lesezugriffe. Anschließend können mit dem GET-Befehl Daten ausgelesen werden.

Wird beim Dateinamen die Extension weggelassen, ergänzt OPENR automatisch '.DAT'.

Beispiel:
```
>OPENR "TEST.DAT"
Ready
>
```

OPENW
Befehl

Syntax:
OPENW <Dateiname>

Beschreibung:
Legt eine Datendatei auf dem aktuellen Speichermodul unter dem angegebenen Dateinamen an. Anschließend können mit dem PUT-Befehl Daten in die Datei geschrieben werden. Ist die angegebene Datei bereits vorhanden, wird diese als gelöscht markiert.

Wird beim Dateinamen die Extension weggelassen, ergänzt OPENW automatisch '.DAT'.

Beispiel:
```
>OPENW "TEST.DAT"
Ready
>
```

OR
Operand

Syntax:
<Ergebnis>=<Wert1> OR <Wert2>

Beschreibung:
Verknüpft zwei Werte bitweise durch ein logisches Oder miteinander.

Beispiel:
```
>PRINT 128 OR 16
144
Ready
>
```

OUT
Befehl

Syntax:

OUT <Portadresse>,<Wert>

Beschreibung:

Schreibt den Wert in das an der Portadresse vorhandene Erweiterungsmodul.

Beispiel:

```
>OUT 64,57
Ready
>
```

P

PACK
Befehl

Syntax:

PACK

Beschreibung:

Entfernt vom aktiven Speichermodul alle gelöschten und überschriebenen Dateien. Dieser Vorgang kann unter Umständen eine Weile dauern.

Beispiel:

```
>CAT
Drive 2
  MINI-LA   .PRG  5394
  IO-TEST   .PRG  463
2 files
59544 bytes free
Ready
>PACK
Ready
>CAT
Drive 2
  MINI-LA   .PRG  5394
  IO-TEST   .PRG  463
2 files
59682 bytes free
Ready
>
```

PEEK / PEEKW / PEEKQ
Funktion

Syntax:

<Bytewert>=PEEK(<Adresse>)
<Wortwert>=PEEKW(<Adresse>)
<Quadwert>=PEEKQ(<Adresse>)

Beschreibung:

Liest ein Byte, Wort, Quadwert von der angegebenen Speicheradresse aus dem Arbeitsspeicher. Dieser Wert kann mit POKE, POKEW, POKEQ in den Speicher geschrieben werden.

Beispiel:

```
>POKE 64000,255:PRINT PEEK(64000)
255
Ready
>POKEW 64000,65535:PRINT PEEKW(64000)
65535
Ready
>POKEQ 64000,4294967295:PRINT PEEKQ(64000)
4294967295
Ready
>
```

PEEK$
Funktion

Syntax:

<String>=PEEK$(<Adresse>,<Anzahl>)

Beschreibung:

Liest einen Datenblock, mit der angegebenen Größe, von der angegebenen Speicheradresse aus dem Arbeitsspeicher und liefert diesen als String zurück. Strings können mit POKES in den Speicher geschrieben werden.

Beispiel:

```
>POKES 64000,"BasicBeetle":PRINT PEEK$(64000,11)
BasicBeetle
Ready
>
```

POKE / POKEW / POKEQ
Befehl

Syntax:

POKE <Adresse>,<Bytewert>
POKEW <Adresse>,<Wortwert>
POKEQ <Adresse>,<Quadwert>

Beschreibung:

Schreibt ein Byte, Wort, Quadwert an die angegebene Speicheradresse im Arbeitsspeicher. Dieser Wert kann mit PEEK, PEEKW, PEEKQ zurück gelesen werden.

Beispiel:

```
>POKE 64000,255:PRINT PEEK(64000)
255
Ready
>POKEW 64000,65535:PRINT PEEKW(64000)
65535
Ready
>POKEQ 64000,4294967295:PRINT PEEKQ(64000)
4294967295
Ready
>
```

POKES
Befehl

Syntax:

POKES <Adresse>,<String>

Beschreibung:

Schreibt den Inhalt eines Strings an die angegebene Speicheradresse in den Arbeitsspeicher. Strings können mit PEEK$ aus den Speicher ausgelesen werden.

Beispiel:

```
>POKES 64000,"BasicBeetle":PRINT PEEK$(64000,11)
BasicBeetle
Ready
>
```

PORT
Befehl

Syntax:

PORT <Name>=<Adresse>[,<Name>=<Adresse>]

Beschreibung:

Weist einem Namen eine bestimmte Adresse eines Erweiterungsmoduls zu. Im Programm kann der Name wie eine Systemvariable verwendet werden. Hier kann auf die IN und OUT-Befehle verzichtet werden. Es werden dann Werte, bei einer Zuweisung, an die gegebene Portadresse geschickt oder auch abgefragt.

Bei der Zuweisung eines Strings werden, direkt hintereinander, die einzelne Zeichen des Strings zum angegebenen Port geschickt. Dies ist z.B. hilfreich bei LCD-Modulen.

Soll eine Portdefinition wieder gelöscht werden, kann dies mit dem ERASE-Befehl geschehen. Portdefinitionen werden wie normale Variablen behandelt und gehen somit auch bei einem NEW-Befehl, CLEAR oder dem abändern des Programms verloren.

Beispiel:

```
>PORT LCDC=&HF1,LCDD=&HF0
Ready
>LCDC=1:LCDD="BasicBeetle 1.31"
```

(Das angeschlossene LCD-Modul wird gelöscht und es erscheint 'BasicBeetle 1.31' auf dem Display)

```
Ready
>
```

Potenz, ^
Verknüpfung

Syntax:

<Ergebnis>=<Wert1>^<Wert2>

Beschreibung:

Bildet die Potenz beider angegebenen Werte.

Beispiel:

```
>PRINT 2^8
256
Ready
>
```

PRINT / ?
Befehl

Syntax:

PRINT <Argument>[;/,][<Argument>]...
? <Argument>[;/,][<Argument>]...

Beschreibung:

Gibt Daten/Texte über den aktuellen Kanal aus. Es können als Argument alle möglichen Datenarten angegeben werden. Sollen mehrere Argumente ausgegeben werden, so müssen diese durch Komma oder Semikolon getrennt werden. Das verwendete Zeichen bestimmt, wo das nächste Argument ausgegeben wird. Dabei bedeutet:

';': Das nächste Argument wird direkt im Anschluss an die letzte Ausgabe angehängt.
',': Zwischen den Argumenten wird ein Tabulatorzeichen an das Terminal ausgeben (Ascii 9). Dies veranlasst das Terminal in die nächste Tabulatorposition zu springen, bevor die nächste Ausgabe folgt.

Folgt nach dem Trennzeichen kein weiteres Argument, bleibt der Cursor in der neuen Position. Wird nach dem Argument kein Trennzeichen gesetzt, erfolgt eine Zeilenschaltung.

Soll nur eine Zeilenschaltung erfolgen, kann man PRINT ohne irgendein Argument verwenden.

Beispiel:

```
>PRINT "BasicBeetle"
BasicBeetle
Ready
>PRINT "Basic";"Beetle"
BasicBeetle
Ready
>PRINT "CPU:","BasicBeetle"
CPU:      BasicBeetle
Ready
>PRINT "CPU:",:PRINT "Basic";:PRINT "Beetle";
CPU:      BasicBeetle
Ready
>
```

PRINTER
Variable

Syntax:

PRINTER=<Status>
<Status>=PRINTER

Beschreibung:

Mit der Variablen PRINTER ist es möglich die Ausgaben vom Befehl PRINT, LIST und CAT auf den, am parallelen Druckerport angeschlossenen, Drucker umzuleiten. Hierzu muss der Variablen PRINTER einen Wert ungleich 0 zugewiesen werden. Nach dem Ausdruck kann man mit der Zuweisung von 0 an den Befehl wieder auf Terminalausgabe zurück schalten. Alternativ kann hierzu auch die Variable CONSOLE verwendet werden.

Soll der aktuelle Status abgefragt werden, gibt PRINTER True (=65535) zurück, wenn zur Zeit der Druckstatus aktiv ist. Andernfalls wird False (=0) zurück geliefert.

Beispiel:

>**PRINTER=TRUE:LIST:PRINTER=FALSE**
(Das aktuelle Programm wird auf den Drucker ausgegeben)
Ready
>

PUT
Befehl

Syntax:

PUT <Var>[,<Var>[,...]]

Beschreibung:

Schreibt den Inhalt der angegebenen Variablen in eine geöffnete Schreibdatei. Mehrere Variablen können, durch Komma getrennt, angegeben werden.

Beispiel:

>**PUT WERT**
Ready
>

R

RBIT
Funktion

Syntax:

<Wert>=RBIT(<Wert>,<Bitnummer>)

Beschreibung:

Gibt einen Wert zurück, indem das angegebene Bit gelöscht wurde. Diese Funktion ist für max. 16 Bit ausgelegt. Größere Werte müssen zuvor mit LOWORD und HIWORD zerlegt werden.

Beispiel:

```
>PRINT RBIT(144,4)
128
Ready
>
```

READ
Befehl

Syntax:

READ <Var>[,<Var>]...

Beschreibung:

Liest einen Wert aus der aktuellen DATA-Position und speichert das Ergebnis in der angegebenen Variablen. Hierbei müssen ausgelesener Datentyp und Variablentyp identisch sein. Der Datenzeiger wird anschließend auf den nächsten Datenwert gesetzt.

Wird der READ-Befehl das erste Mal aufgerufen, werden die Daten aus den ersten gefundenen DATA-Zeilen gelesen. Möchte man aus bestimmten DATA-Zeilen lesen, so muss der RESTORE-Befehl eingesetzt werden.

Beispiel:

```
>10 DIM VNAME$,NNAME$,ALTER
>20 RESTORE 100
>30 READ VNAME$,NNAME$,ALTER
>40 PRINT VNAME$;" ";NNAME$;", ";ALTER;" Jahre":END
>100 DATA "Hans","Heinrich",2009-1970
>RUN
Hans Heinrich, 39 Jahre
Ready
>
```

READBLOCK$
Funktion

Syntax:
<Datenstring>=READBLOCK$(<Blocknummer>)

Beschreibung:
Liest einen Datenblock von der angegebenen Blocknummer vom aktiven Speicherlaufwerk und liefert das Ergebnis als String zurück.

Beispiel:
```
>SAVEBLOCK 1000,STRING$(16,"#"):PRINT READBLOCK$(1000)
################
Ready
>
```

REM / '
Befehl

Syntax:
REM <Kommentar>
' <Kommentar>

Beschreibung:
Definiert den Rest der Zeile als Kommentar. Die Zeile wird nicht weiter beachtet.

Beispiel:
```
>10 REM *** Programmstart ***
>RUN
(Zeile 10 wird ignoriert)
Ready
>
```

RENAME
Befehl

Syntax:
RENAME <Altname>,<Neuname>

Beschreibung:
Benennt eine Datei um. Wurde keine Extension angegeben, wird automatisch '.PRG' angenommen.

Beispiel:
```
>RENAME "TESTPROG","STANZE"
Ready
>
```

REPEAT
Befehl

Syntax:

REPEAT <Anzahl> - ENDREPEAT

Beschreibung:

Eröffnet eine Schleife, welche eine bestimmte Anzahl durchlaufen wird. Es wird keine Laufvariable benötigt. Die Schleife muss mit ENDREPEAT geschlossen werden.

Beispiel:

```
>REPEAT 10:PRINT "*";:ENDREPEAT
**********
Ready
>
```

RESET
Befehl

Syntax:

RESET

Beschreibung:

Setzt den BasicBeetle zurück und startet das System neu.

Beispiel:

```
>RESET
(Bildschirm wird gelöscht)
*** BasicBeetle 1.31 ***
(C) 2007-2009 by Thomas Krueger
64256 bytes RAM found
Ready
>
```

RESTORE
Befehl

Syntax:

RESTORE <Zeile>

Beschreibung:

Setzt den Zeiger für DATA-Zeilen auf eine neue Zeile. Beim nächsten READ-Befehl werden die Daten dann aus dieser Zeile gelesen.

Beispiel:

```
>10 DIM VNAME$,NNAME$,ALTER
>20 RESTORE 100
>30 READ VNAME$,NNAME$,ALTER
>40 PRINT VNAME$;" ";NNAME$;", ";ALTER;" Jahre":END
>100 DATA "Hans","Heinrich",2009-1970
>RUN
Hans Heinrich, 39 Jahre
Ready
>
```

RESUME
Befehl

Syntax:

RESUME <Zeilennummer>

Beschreibung:

Beendet eine Fehlerbehandlungsroutine und führt das Programm an der angegeben Stelle fort.

Beispiel:

```
>10 ON ERROR GOTO 60000
>20 PRIT Hallo
>30 PRINT "Programmende":END
>60000 PRINT "Es gibt einen Fehler ";ERR;" in Zeile ";ERL
>60010 RESUME 30
>RUN
Es gibt einen Fehler 1 in Zeile 20
Programmende
Ready
>
```

RETURN
Befehl

Syntax:

RETURN [<Zeile>]

Beschreibung:

Kehrt von einem Unterprogramm wieder ins Hauptprogramm zurück. Das Programm wird nach dem aufrufenden GOSUB-Befehl weiter abgearbeitet.

Wird zusätzlich eine Zeilennummer angegeben, so springt der Computer an die angegebene Zeile. Die alte Rücksprungposition wird ignoriert.

Beispiel:

```
>10 PRINT "Hallo ";:GOSUB 50
>20 END
>50 PRINT "User":RETURN
>RUN
Hallo User
Ready
>
```

RIGHT$
Funktion

Syntax:

<String>=RIGHT$(<Anzahl>,<Quellstring>)

Beschreibung:

Liefert eine bestimmte Anzahl von Zeichen von der rechten Seite des Quellstrings zurück.

Beispiel:

```
>PRINT RIGHT$(4,"Hallo User")
User
Ready
>
```

RND
Funktion

Syntax:

<Zahl>=RND

Beschreibung:

Generiert eine zufällige Zahl zwischen 0 und 65535.

Beispiel:

```
>PRINT RND
4512  (Zahl ist bei jedem Aufruf anders)
Ready
>
```

RTRIM$
Funktion

Syntax:

<String>=RTRIM$(<Zeichen>,<String>)

Beschreibung:

Entfernt die eventuellen angegebenen folgenden Zeichen vom String und liefert den Rest zurück. Das gewünschte Zeichen kann als Ascii-Nummer oder als String angegeben werden. Bei der Angabe als String wird nur das erste Zeichen verwendet.

Beispiel:

```
>PRINT LTRIM$("*",RTRIM$("*","***BasicBeetle***"))
BasicBeetle
Ready
>
```

RUN
Befehl

Syntax:

RUN
RUN <Dateiname>

Beschreibung:

Wird der Befehl ohne Parameter eingegeben startet das Programm, welches sich zur Zeit im Arbeitsspeicher befindet.

Ergänzt man RUN mit einem Dateinamen, wird zuerst das angegebene Programm vom Speichermodul aus dem aktuellen Laufwerk geladen und anschließend sofort gestartet.

Beispiel:

```
>10 PRINT "Hallo User"
>RUN
Hallo User
Ready
>RUN "Mini-LA"
```

S

SAVE
Befehl

Syntax:

SAVE <Dateiname>

Beschreibung:

Speichert das Programm, welches sich zur Zeit im Arbeitsspeicher befindet, auf das aktive Speichermodul unter dem angegebenen Dateinamen. Wird die Extension weggelassen, fügt der BasicBeetle automatisch .PRG hinzu.

Beispiel:

```
>SAVE "TESTPROG"
Ready
>
```

SAVEBIN
Befehl

Syntax:

SAVEBIN <Startadresse>,<Anzahl>,<Dateiname>

Beschreibung:

Speichert einen Teil des Arbeitsspeichers, unter dem angegebenen Dateinamen auf dem aktuellen Speichermodul ab. Wird die Extension weggelassen, fügt der BasicBeetle automatisch .BIN hinzu. Hierbei werden von der Startadresse an, die angegeben Anzahl Bytes gespeichert.

Beispiel:

```
>SAVEBIN &H8000,1024,"AD-TAB"
Ready
>
```

SAVEBLOCK
Befehl

Syntax:

SAVEBLOCK <Blocknummer>,<Datenstring>

Beschreibung:

Speichert den Datenstring auf der angegebenen Blocknummer auf dem aktiven Speichermodul. Der Datenstring kann bis zu 16 Zeichen enthalten. Enthält dieser weniger, werden die restlichen Zeichen mit Ascii 255 ausgefüllt. Ein Leerstring löscht also den angegebenen Datenblock. Ist der String länger als 16 Byte, werden nur die ersten 16 Byte berücksichtigt.

Beispiel:

```
>SAVEBLOCK 1000,STRING$(16,"#"):PRINT READBLOCK$(1000)
################
Ready
>
```

SBIT
Funktion

Syntax:

<Wert>=SBIT(<Wert>,<Bitnummer>)

Beschreibung:

Gibt einen Wert zurück, indem das angegebene Bit gesetzt wurde. Diese Funktion ist für max. 16 Bit ausgelegt. Größere Werte müssen zuvor mit LOWORD und HIWORD zerlegt werden.

Beispiel:

```
>PRINT SBIT(128,4)
144
Ready
>
```

SEEK
Befehl

Syntax:

SEEK <Dateiposition>

Beschreibung:

Setzt den Positionszeiger für eine geöffnete lesende Datendatei auf die angegebene Position. Hierdurch kann man relative Dateizugriffe gestalten.

Beispiel:

```
>GET WERT:PRINT WERT
656
Ready
>SEEK 0:GET WERT:PRINT WERT
656
Ready
>
```

SELECT
Befehl

Syntax:

SELECT <Wert>

Beschreibung:

Startet die Tabellenabfrage eines Wertes. Durch die nachfolgenden CASE-Anweisungen ist eine schnelle Abfrage eines Wertes möglich.

Der Wert darf höchstens 16 Bit haben. Quad-Werte und Strings sind nicht erlaubt.

Beispiel:

```
>10 DIM NOTE
>20 INPUT "Note : ",NOTE:SELECT NOTE
>30 CASE 1:PRINT "Sehr gut"
>40 CASE 2:PRINT "Gut"
>50 CASE 3:PRINT "Befriedigend"
>60 CASE 4:PRINT "Ausreichend"
>70 CASE 5:PRINT "Mangelhaft"
>80 CASE 6:PRINT "Ungenuegend"
>90 CASEELSE:PRINT "Note nicht erlaubt!"
>RUN
Note : 3
Befriedigend
Ready
>
```

SHL
Operator

Syntax:

<Neuwert>=<Wert> SHL <Stellen>

Beschreibung:

Schiebt den angegeben Wert Bitweise um die angegebenen Stellen nach links.

Beispiel:

```
>PRINT 4 SHL 2
16
Ready
>
```

SHR
Operator

Syntax:

<Neuwert>=<Wert> SHR <Stellen>

Beschreibung:

Schiebt den angegeben Wert Bitweise um die angegebenen Stellen nach rechts.

Beispiel:

```
>PRINT 16 SHL 2
4
Ready
>
```

SLOW
Befehl

Syntax:

SLOW

Beschreibung:

Aktiviert für den System-Bus eine Taktfrequenz von ca. 100 kHz. Mit dem System-Bus werden z.B. die Speichermodule und der Drucker gesteuert. Beim Neustart des BasicBeetle ist diese Taktfrequenz des System-Buses aktiviert. Mit dem FAST-Befehl kann die Frequenz erhöht werden.

Beispiel:

```
>SLOW
Ready
>
```

SPACE$
Funktion

Syntax:

<String>=SPACE$(<Anzahl>)

Beschreibung:

Liefert eine Zeichenkette zurück, welche eine bestimmte Anzahl von Leerzeichen enthält.

Beispiel:

```
>PRINT SPACE$(10);"*"
          *
Ready
>
```

SQRT
Funktion

Syntax:

<Wurzel>=SQRT(<Wert>)

Beschreibung:

Berechnet die Quadratwurzel des angegebenen Wertes.

Beispiel:

```
>PRINT SQRT(144)
12
Ready
>
```

STEP
Befehl

Syntax:

FOR <Variable>=<Start> TO <Ende> STEP <Weite>

Beschreibung:

Bestimmt die Schrittweite beim FOR-NEXT-Befehl. Für die Schrittweite sind nur Wert bis 1-65535 erlaubt.

Beispiel:

```
>DIM T:FOR T=0 TO 10 STEP 2:PRINT T:NEXT
0
2
4
6
8
10
Ready
>
```

STR$
Funktion

Syntax:

<String>=STR$(<Wert>)

Beschreibung:

Wandelt den angegeben Wert in eine Zeichenkette um. Als Wert ist jeder Zahlentyp zulässig.

Beispiel:

```
>PRINT STR$(768)
768
Ready
>
```

STRING$
Funktion

Syntax:

<String>=STRING$(<Anzahl>,<Zeichen>)

Beschreibung:

Liefert eine Zeichenkette zurück, welche eine bestimmte Anzahl von dem angegeben Zeichen enthält. Das Zeichen kann man hierbei als Ascii-Nummer oder als String angeben. Wird ein String übergeben, wird hierbei nur das erste Zeichen beachtet.

Beispiel:

```
>PRINT STRING$(10,42)
**********
Ready
>PRINT STRING$(10,"-")
----------
Ready
>
```

SUBDIR
Befehl

Syntax:

SUBDIR <Zeilenadresse>

Beschreibung:

Ruft ein Unterprogramm auf, welches sich im Arbeitsspeicher ab der angegebenen Adresse befindet. Es muss sichergestellt sein, dass sich die Adresse auch auf den Anfang einer Programmzeile bezieht. Ansonsten kommt es zu unvorhersagbaren Reaktionen. Die Zeilenadresse einer Programmzeile kann man mit der LINEPOS-Funktion ermitteln.

Dieser Befehl ist hilfreich, um eine hohe Geschwindigkeit bei größeren Programmen zu erreichen da hier das suchen der Zeile entfällt.

Beispiel:

```
>10 DIM ADR:ADR=LINEPOS(50)
>20 DO:SUBDIR ADR:LOOP
>50 PRINT "#";:DELAY 10:RETURN
>RUN
########  (Wird unendlich fortgeführt)
```

Subtraktion, -

Verknüpfung

Syntax:

<Ergebnis>=<Wert1>-<Wert2>

Beschreibung:

Bildet die Differnz beider angegebenen Werte.

Beispiel:

```
>PRINT 34-17
17
Ready
>
```

SWAP

Befehl

Syntax:

SWAP <Var1>,<Var2>

Beschreibung:

Vertauscht den Inhalt zweier numerischer Variablen. Beide Variablen müssen vom gleicher Datentyps sein. Strings sind nicht erlaubt.

Beispiel:

```
>DIM T1,T2:T1=123:T2=56:SWAP T1,T2:PRINT T1,T2
56       123
Ready
>
```

T

THEN
Befehl

Syntax:

IF <Bedingung> THEN <Wahr-Befehle> [ELSE <Falsch-Befehle>]

Beschreibung:

Die folgendende Befehle werden nur ausgeführt, wenn der vorangegangene IF-Befehl eine wahre Bedingung oder der vorangegangene IFF-Befehl eine falsche Bedingung ermittelt hat.

Beispiel:

```
>IF 5=5 THEN PRINT "Stimmt"
Stimmt
Ready
>IF 5=3 THEN PRINT "Stimmt"
Ready
>IF 5=3 THEN PRINT "Stimmt" ELSE PRINT "Stimmt nicht"
Stimmt nicht
Ready
>
```

TIMER
Variable

Syntax:

<Wert>=TIMER
TIMER=<Wert>

Beschreibung:

Die Variable Timer ist ein interner Zähler, welcher automatisch jede Sekunde um 1 erhöht wird. Mit Hilfe von TIMER ist es möglich eine einfache Zeitmessung vorzunehmen.

Die Variable TIMER hat einen Umfang von 16 Bit. Nach 65536 Sekunden wird der Timer wieder auf 0 gesetzt.

Beispiel:

```
>TIMER=0:DELAY 1000:PRINT "Das hat ";TIMER;" Sekunden gedauert."
Das hat 10 Sekunden gedauert.
Ready
>
```

TICKER
Variable

Syntax:

<Wert>=TICKER
TICKER=<Wert>

Beschreibung:

Die Variable Timer ist ein interner Zähler, welcher automatisch jede 100tel Sekunde um 1 erhöht wird. Mit Hilfe von TICKER ist es möglich eine einfache Zeitmessung vorzunehmen.

Die Variable TICKER hat einen Umfang von 16 Bit. Nach 655,36 Sekunden wird der Timer wieder auf 0 gesetzt.

Beispiel:

```
>TICKER=0:DELAY 1000:PRINT "Das hat ";TIMER;" 100tel Sekunden gedauert."
Das hat 1000 100tel Sekunden gedauert.
Ready
>
```

TO
Befehl

Syntax:

FOR <Variable>=<Startwert> TO <Endwert> [STEP <Weite>] - NEXT

Beschreibung:

Dieser Befehl gehört zum FOR-Befehl und hat alleine keine Funktion.

Beispiel:

```
>DIM CNT:FOR CNT=1 TO 5:PRINT "Nummer ";CNT:NEXT
Nummer 1
Nummer 2
Nummer 3
Nummer 4
Nummer 5
Ready
>
```

TRIM$
Funktion

Syntax:

<String>=TRIM$(<Zeichen>,<String>)

Beschreibung:

Entfernt die eventuellen angegebenen führenden und folgenden Zeichen vom String und liefert den Rest zurück. Das gewünschte Zeichen kann als Ascii-Nummer oder als String angegeben werden. Bei der Angabe als String wird nur das erste Zeichen verwendet.

Beispiel:

```
>PRINT TRIM$("*","***BasicBeetle***")
BasicBeetle
Ready
>
```

TRUE
Funktion

Syntax:

65535=TRUE

Beschreibung:

Gibt den Wert für True (=65535) zurück.

Beispiel:

```
>PRINT TRUE
65535
Ready
>
```

U

UNHIDE
Befehl

Syntax:

UNHIDE <Dateiname>

Beschreibung:

Entfernt die 'Unsichtbar'-Markierung von der angegebenen Datei, welche zuvor mit dem HIDE-Befehl entsprechend markiert worden ist. Die Datei erscheint wieder beim CAT-Befehl oder der DIR$-Funktion. Wird keine Extension angegeben, so wird automatisch '.PRG' angenommen.

Beispiel:

```
>CAT
Drive 2
  MINI-LA   .PRG  5394
  IO-TEST   .PRG  463
2 files
59514 bytes free
Ready
>UNHIDE "TESTPROG"
Ready
>CAT
Drive 2
  MINI-LA   .PRG  5394
  TESTPROG  .PRG  117
  IO-TEST   .PRG  463
3 files
59514 bytes free
Ready
>
```

UNTIL
Befehl

Syntax:

DO - UNTIL <Bedingung>

Beschreibung:

Führt eine mit DO eröffnete Schleife fort, solange die angegebene Bedingung falsch ist.

Beispiel:

```
>10 DIM CNT
>20 DO:PRINT "*";:INC CNT:UNTIL CNT=10
>RUN
*********
Ready
>
```

UPPER$

Funktion

Syntax:

<String>=UPPER$(<String>)

Beschreibung:

Wandelt den übergebenen String in Großbuchstaben um. Umlaute bzw. länderspezifische Sonderzeichen werden nicht berücksichtigt.

Beispiel:

```
>PRINT UPPER$("Hallo User")
HALLO USER
Ready
>
```

V

VAL
Funktion

Syntax:

<Wert>=VAL(<String>)

Beschreibung:

Wandelt die Zahl, die in der Form eines Strings vorliegt, um. Kann die Zahl nicht umgewandelt werden, so wird 0 zurück gegeben.

Beispiel:

```
>PRINT VAL("753")
753
Ready
>
```

VARPTR
Funktion

Syntax:

<Adresse>=VARPTR(<Variable>)

Beschreibung:

Liefert die Adresse des Datenbereichs der angegebenen Variablen. Hierdurch kann man den Inhalt der entsprechenden Variablen durch POKE/PEEK ausgelesen oder verändert werden.

Beispiel:

```
>DIM WERT:WERT=753:PRINT PEEKW(VARPTR(WERT))
753
Ready
>
```

VER$
Funktion

Syntax:

<Version>=VER$

Beschreibung:

Liefert die aktuelle BasicBeetle-Version als String zurück.

Beispiel:

```
>PRINT VER$
1.31
Ready
>
```

Vergleiche
Operand

Syntax:

<Ergebnis>=<Wert1>[=,>,>=,=>,<,<=,=<,<>,><]<Wert2>

Beschreibung:

Vergleicht die angegebene Werte miteinander und gibt, je nach gewünschter Vergleichsoperation, Wahr (=65535) zurück wenn das Vergleichsergebnis stimmt. Andernfalls wird Falsch (=0) zurück gegeben.

Beispiel:

```
>PRINT 170=144
0
Ready
>PRINT 170>144
65535
Ready
>
```

Verkettung, +
Verknüpfung

Syntax:

<String>=<Teilstring1>+<Teilstring2>

Beschreibung:

Fügt zwei oder mehr Strings zu einem Großen zusammen und liefert diesen zurück.

Beispiel:

```
>PRINT "Basic"+"Beetle"
BasicBeetle
Ready
>
```

W

WAITS
Befehl

Syntax:

WAITS <Zyklen>

Beschreibung:

Setzt die Anzahl der Wartezyklen für die Ein und Ausgabe auf Erweiterungs-Module neu. Standardmäßig sind 0 Wartezyklen aktiviert. Bei langsamen Erweiterungsmodulen können die Wartezyklen bis 65535 erhöht werden. Hierbei bedeutet jede Erhöhung um 1 eine Verlängerung der Ausgabezeit auf dem Portbus um ca. 1µS.

Beispiel:

```
>WAITS 10
Ready
>
```

WHILE
Befehl

Syntax:

DO - WHILE <Bedingung>

Beschreibung:

Führt eine mit DO eröffnete Schleife fort, solange die angegebene Bedingung wahr ist.

Beispiel:

```
>10 DIM CNT
>20 DO:PRINT "*";:INC CNT:WHILE CNT<10
>RUN
*********
Ready
>
```

X

XOR
Operand

Syntax:
<Ergebnis>=<Wert1> XOR <Wert2>

Beschreibung:
Verknüpft zwei Werte bitweise durch ein logisches Exklusiv-Oder miteinander.

Beispiel:
```
>PRINT 170 XOR 128
42
Ready
>
```

XPEEK / XPEEKW / XPEEKQ
Funktion

Syntax:
<Bytewert>=XPEEK(<Adresse>)
<Wortwert>=XPEEKW(<Adresse>)
<Quadwert>=XPEEKQ(<Adresse>)

Beschreibung:
Liest ein Byte, Wort, Quadwert von der angegebenen Speicheradresse aus dem erweiterten Speicher. Dieser Wert kann mit XPOKE, XPOKEW, XPOKEQ in den Speicher geschrieben werden.

Der Bereich unterhalb Lomem im erweiterten Speicher kann systembedingt leider nicht ausgelesen werden.

Beispiel:
```
>XPOKE 64000,255:PRINT XPEEK(64000)
255
Ready
>XPOKEW 64000,65535:PRINT XPEEKW(64000)
65535
Ready
>XPOKEQ 64000,4294967295:PRINT XPEEKQ(64000)
4294967295
Ready
>
```

XPEEK$
Funktion

Syntax:

<String>=XPEEK$(<Adresse>,<Anzahl>)

Beschreibung:

Liest einen Datenblock, mit der angegebenen Größe, von der angegebenen Speicheradresse aus dem erweiterten Speicher und liefert diesen als String zurück. Strings kann man mit XPOKES in den Speicher geschrieben werden.

Beispiel:

```
>XPOKES 64000,"BasicBeetle":PRINT XPEEK$(64000,11)
BasicBeetle
Ready
>
```

XPOKE / XPOKEW / XPOKEQ
Befehl

Syntax:

XPOKE <Adresse>,<Bytewert>
XPOKEW <Adresse>,<Wortwert>
XPOKEQ <Adresse>,<Quadwert>

Beschreibung:

Schreibt ein Byte, Wort, Quadwert an die angegebene Speicheradresse im erweiterten Speicher. Dieser Wert kann mit XPEEK, XPEEKW, XPEEKQ zurück gelesen werden.

Der Bereich unterhalb Lomem im erweiterten Speicher kann systembedingt leider nicht beschrieben werden.

Beispiel:

```
>XPOKE 64000,255:PRINT XPEEK(64000)
255
Ready
>XPOKEW 64000,65535:PRINT XPEEKW(64000)
65535
Ready
>XPOKEQ 64000,4294967295:PRINT XPEEKQ(64000)
4294967295
Ready
>
```

XPOKES

Befehl

Syntax:

XPOKES <Adresse>,<String>

Beschreibung:

Schreibt den Inhalt eines Strings an die angegebene Speicheradresse in den erweiterten Speicher. Strings kann man mit XPEEK$ aus den Speicher ausgelesen werden.

Beispiel:

```
>XPOKES 64000,"BasicBeetle":PRINT XPEEK$(64000,11)
BasicBeetle
Ready
>
```

Fehler 1: Syntax error

Diese Fehlermeldung ist wohl die häufigste Fehlermeldung in der ganzen Computer-Programmierer-Welt. Er wird immer dann ausgegeben, wenn der Computer irgendetwas gar nicht versteht.

```
>PRIT "Hallo"

Error 1: Syntax error
Ready
>
```

Fehler 2: Subscript out of range

Wird diese Meldung ausgegeben, wurde irgendein Wertbereich überschritten.

```
>PRINT CHR$(300)

Error 2: Subscript out of range
Ready
>
```

Fehler 3: Parameter missing

Der BasicBeetle erwartet für den Befehl noch weitere Parameter. In der angegeben Form, kann der Computer den Befehl nicht ausführen.

```
>PRINT STRING$(5)

Error 3: Parameter missing
Ready
>
```

Fehler 4: Unexpected end of line

Beim Ausführen des Befehls ist der Computer auf das Zeilenende gestoßen, obwohl der Befehl noch nicht vollständig ausgeführt wurde. Oft wurde dann eine schließende Klammer einer Funktion vergessen.

```
>PRINT CHR$(65

Error 4: Unexpected end of line
Ready
>
```

Fehler 5: Type missmatch

Der Computer findet einen falschen Datentyp als Parameter vor. Oft wird die Reihenfolge der Parameter verwechselt. Hierzu am besten noch einmal in der Befehlsreferenz nachschauen.

```
>PRINT CHR$("A")

Error 5: Type missmatch
Ready
>
```

Fehler 6: Empty string

Der Befehl oder die Funktion erwartet einen String in dem mindestens 1 Zeichen enthalten sind. Leere Strings kann der Befehl nicht verarbeiten.

```
>PRINT ASC("")

Error 6: Empty string
Ready
>
```

Fehler 7: Line not found

Es wurde versuchst eine Programmzeile aufzurufen, welche nicht vorhanden ist.

```
>GOTO 100

Error 7: Line not found
Ready
>
```

Fehler 8: String to long

Bei einer Stringoperation entsteht ein Ergebnisstring, welcher mehr wie 255 Zeichen enthält. Strings mit mehr als 255 Zeichen, kann der BasicBeetle nicht verarbeiten.

```
>PRINT STRING$(200,"#")+STRING$(200,"*")

Error 8: String to long
Ready
>
```

Fehler 9: Variable not defined

Es wurde versucht eine Variable zu verwenden, welche nicht definiert ist.

```
>INC COUNTER

Error 9: Variable not defined
Ready
>
```

Fehler 10: Stack error

Der BasicBeetle kann nur eine bestimmte Anzahl von geschachtelten Schleifen und/oder Unterprogrammen verarbeiten. Wurde diese Anzahl überschritten, erfolgt diese Fehlermeldung. Eine häufige Ursache ist der fehlende RETURN-Befehl bei nach GOSUB-Aufrufen.

```
>10 GOSUB 10
>RUN

Error 10: Stack error in 10
Ready
>
```

Fehler 11: No SELECT

Es soll eine Tabellenabfrage durchgeführt werden, ohne dass vorher das Abfrageelement festgelegt wurde.

```
>10 CASE 13
>RUN

Error 11: No SELECT in 10
Ready
>
```

Fehler 12: No more DATA

Der READ-Befehl findet keine Daten mehr. Entweder wurde das auslesen der DATA-Zeilen nicht rechtzeitig beendet oder es wurde der RESTORE-Befehl vergessen.

```
>READ WERT

Error 12: No more DATA
Ready
>
```

Fehler 13: Direct in program

Ein Befehl, welcher nur für den Direktmodus erlaubt ist, wurde im Programm verwendet.

```
>10 NEW
>RUN

Error 13: Direct in program in 10
Ready
>
```

Fehler 14: Device error

Ein Zugriff auf eine Systemerweiterung oder eines Speichermoduls ist fehlgeschlagen. Tritt dieser Fehler bei einem Zugriff auf Speichermodulen statt, bitte überprüfen ob sich ein Speichermodul im Slot befindet.

```
>CAT

Error 14: Device error
Ready
>
```

Fehler 15: File not found

Es wurde versucht ein Programm zu laden oder eine Datendatei zu öffnen, welche auf dem aktuellen Speichermodul nicht vorhanden ist.

```
>LOAD "NOTTHERE"

Error 15: File not found
Ready
>
```

Fehler 16: Device full

Das Speichern eines Programms oder das ablegen weiterer Daten auf dem aktuellen Speichermodul ist nicht möglich, da die Kapazität erschöpft ist..

```
>SAVE "TESTPROG"

Error 16: Device full
Ready
>
```

Fehler 17: Error while printing

Während des Druckens ist ein Problem aufgetreten. Wurde der Drucker angeschlossen? Ist dieser Online?

```
>PRINTER=1:PRINT "Test":PRINTER=0

Error 17: Error while printing
Ready
>
```

Fehler 18: End of file

Beim Lesen von Daten aus einer Datendatei wurde festgestellt, dass das Dateiende erreicht wurde.

```
>GET WERT

Error 18: End of file
Ready
>
```

Fehler 19: File already open

Es ist nur erlaubt, 1 schreibende und 1 lesende Datendatei gleichzeitig zu öffnen. Soll eine 2. Datendatei geöffnet werden, erfolgt diese Meldung.

```
>OPENR "DATEI1":OPENR "DATEI2"

Error 19: File already open
Ready
>
```

Fehler 20: File not open

Man Will Daten aus einer Datendatei lesen oder in eine Datei schreiben, welche aber nicht geöffnet wurde.

```
>GET WERT

Error 20: File not open
Ready
>
```

Fehler 21: File already exists

Es wurde bei einer Dateioperation festgestellt, dass eine Zieldatei bereits existiert.

```
>COPYFILE "TESTPROG",3

Error 21: File already exists
Ready
>
```

Fehler 22: Paper out

Das Papier im Drucker ist ausgegangen. Papier nachlegen und den entsprechenden Druckbefehl wiederholen.

```
>PRINTER=1:PRINT "Test":PRINTER=0

Error 17: Paper out
Ready
>
```

Fehler 23: Memory Full

Der Arbeitsspeicher ist erschöpft oder es wurde versucht, den Arbeitsspeicher weiter zu beschränken als dieser zur Zeit schon in Verwendung ist. Daher konnte der Befehl nicht ausgeführt werden.

```
>DIM WERT(40000)

Error 23: Memory Full
Ready
>
```

Fehler 24: Only in run

Es wurde versucht, einen Befehl im Direktmodus zu geben, welcher nur innerhalb eines Programms erlaubt ist.

```
>CHAIN "MAIN.CHN"

Error 24: Only in run
Ready
>
```