Benutzerhandbuch

für

Version 2.0

Reinhard Zielinski
Berlin, März 2013

Inhaltsverzeichnis

Kapitel:

Vorwort

Installation von Trios-Basic

Der Arbeitsbildschirm

Die Schnellzugriffstasten

- F1 HELP-Die integrierte Hilfefunktion
- F2 LOAD-Programme laden
- F3 SAVE-Programme speichern
- F4 DIR-DIRECTORY anzeigen
- F5 RUN-Programm starten
- F6List-Programmlisting anzeigen
- F7 EDIT – Programmzeile editieren
- F8 TRON – Debugmodus einschalten / F9 TROFF -Debugmodus ausschalten
- F10 RECLAIM-Programm wiederherstellen
- F12 Basic beenden

Der Kommandomodus

Der Programmmodus

Das Demo-Programm laden

Fehlermeldungen

Variablen

Strings

Arrays

String-Arrays

Befehle zur Steuerung des Programmablaufs

- IF...THEN...ELSE, ON ..GOTO, GOSUB, FOR ..NEXT, RUN, END ,PAUSE

Datums- und Zeit-Funktionen
STIME, SDATE, SDOW, GTIME, GDATE, TIME, TIMER
Systembefehle
BYE, COGS, DUMP, FREE, NEW, LIST, CLEAR, TRON,
TROFF, EDIT, VER, RECLAIM
Logische Operatoren
NOT, AND, OR
Ein- und Ausgabebefehle
PEEK, POKE, INPUT, INKEY, REM
Befehle zur Benutzung der Maus
MOUSE, MBOUND, MX, MY, MZ, MB
Bildschirmbefehle / Funktionen
PRINT, CLS, HOME, POS, TAB, COL, PLOT, SCRDN, SCRUP, CROFF,
CRON, TWIN, WIN,TBAR, MBOX, FRAME, WSET, SCROLL, BUTTON,
DBUTT, BOX, GETX, GETY
Stringfunktionen
STR$, COMP$, LEN, CHR$, ASC, VAL, STRING$
Tile-Grafik
TLOAD, STILE, TPIC, TILE, FONT, MAP
Mathematische Funktionen
FN, RND
Funktionen der seriellen Schnittstelle
COM, COMG, COMR
SID-Soundbefehle
VOL, NT, NTOFF, PLAY, ADSR, WAVE, FLT, FMASK, RGMOD, CUT, RES,
PWM, SYNC, GDMP, BEEP
Dateifunktionen
OPEN, CLOSE, READ, WRITE, LOAD, SAVE, FILE, DIR, DEL, REN,
CHDIR, GFILE, GATTR, MKDIR, BLOAD
Welche Dateien wurden mit TRIOS-BASIC mitgeliefert
Befehlsübersicht von TRIOS-BASIC
Die Speicheraufteilung im e-RAM

Vorwort

Willkommen zu Trios-Basic 2.0 und willkommen auch zum spannenden Abenteuer Hive-Computer. Was ist denn am Hive-Computer spannend?, wird sich der Leser jetzt fragen. Ganz leicht ist die Frage nicht zu beantworten, da man schon etwas Enthusiasmus für Computer und Controllertechnik mitbringen muss um das zu verstehen, was den Hive so anders, als andere Projekte macht.
Ich stieß auf den Propeller-Chip (eines von 3 Herzen, welche im Hive schlagen) vor ca. 5-6 Jahren. Mit seinen Möglichkeiten, auf kleinsten Raum einen vollwertigen mit VGA oder TV-Ausgang, Tastatur und Maus-Port sowie Soundausgang ausgestatteten Kleincomputer zu realisieren, war er für mich ein bisher einzigartiger Chip, welchen ich mir mal genauer anschauen musste. Also Chip und externe Komponenten bestellt, Platine gezeichnet und geätzt und ran ans Löten.
Mich überraschte die simple Außenbeschaltung und das für Hobbybastler gute Handling des Chips als 40 Pin Dip-Gehäusevariante. Alles funktionierte auf Anhieb und die Testprogramme von Parallax ließen das Potential dieses Winzlings erahnen. Einzig die, mir völlig neue Programmiersprache Spin, war mir etwas suspekt. Obwohl ich einige Programmiersprachen schon durch hatte, (Basic,Pascal,C, Profan, Visual-Basic) wurde ich nicht so richtig warm mit diesem neuen Dialekt. Da zu dieser Zeit auch die deutsche Bastelwelt noch so gut wie nichts von diesem Chip gehört hatte, war eine Unterstützung für Probleme oder die Beantwortung von Fragen so gut wie ausgeschlossen, weshalb ich diesen coolen Controller wieder zu den Akten legte und ich mich mit dem AVR-Controller anfreundete.
Einige Jahre vergingen und wie es bei so vielen Bastelfreunden sicher auch üblich ist, durchwühlte ich meine Bastelkiste, weil ich irgend etwas suchte.
Nun fiel mir meine kleine Proto-Platine wieder in die Hände. Zuerst wollte ich sie wieder in die Kiste zurücktun, aber ich entschied einen neuen Versuch zu starten, diesen Controller zu bändigen. Hinreichend Erfahrung mit AVR-Controllern im Gepäck, war ich optimistisch, jetzt erfolgreicher zu sein.
Erste Anlaufstelle – Internet! Jetzt war der Bekanntheitsgrad des Propeller schon um einiges höher als damals, also las ich mich ein wenig ein, klickte auf diverse Links und stieß auf die Hive-Project-Seite.
Nun musste ich nur noch herausbekommen, ob ich „reif für den Hive?“ war. Zeilen kurz überflogen und die Antwort lautete „JAAAA“ auf jeden Fall.
Ich bestellte die Platine und die benötigten Bauteile und fing an, meinen Hive aufzubauen. Etwas skeptisch war ich ja schon, ob alles funktioniert und die serielle Schnittstelle brachte mich auch fast an den Rand des Wahnsinns, bis sie funktionierte. Aber dank der Projekt-Seite (dieses Problem hatten auch andere schon) fand ich den Fehler dann doch und mein Hive erblickte das Licht der Welt.
Nun musste ich mich ja wieder mit der damals für mich so seltsamen Programmiersprache SPIN auseinander setzen. Nun fiel der Groschen dank meiner Erfahrungen in der AVR-Programmierung und ich fing an, die SPIN-Programme zu verstehen. Jetzt entzündete das Feuer der Begeisterung, wenn ein selbst

geschriebenes Programm genau das tut, was man beabsichtigt hat.
Aber was ist denn nun am Hive anders?
Nun, ein Board, auf dem 3 Propeller-Chips mit je 8 Kernen zusammenarbeiten, jeder seine spezielle Funktion unabhängig vom anderen ausübt und dazu noch leicht aufzubauen und zu programmieren ist, macht den Hive zu einem Selbstbauprojekt der Extraklasse mit dem Lernfaktor 300 im Gegensatz zu anderen Bastelprojekten.
Um auch den Neulingen im Umgang mit dem Propeller-Chip einen leichten Einstieg ohne die Probleme ,die ich damals hatte, zu ermöglichen, ist das TRIOS-BASIC-Projekt entstanden.
Gerade Programmierneulinge sollen mit TBasic in die Programmierung eingeführt werden aber auch die alten Hasen werden viel Freude am Retro-Feeling einer Basic-Programmierung im Stile des C64 oder KC85 haben.
Ich wünsche euch viel Freude und Begeisterungs-Feuer beim Umgang mit eurem Hive.

Euer Zille9

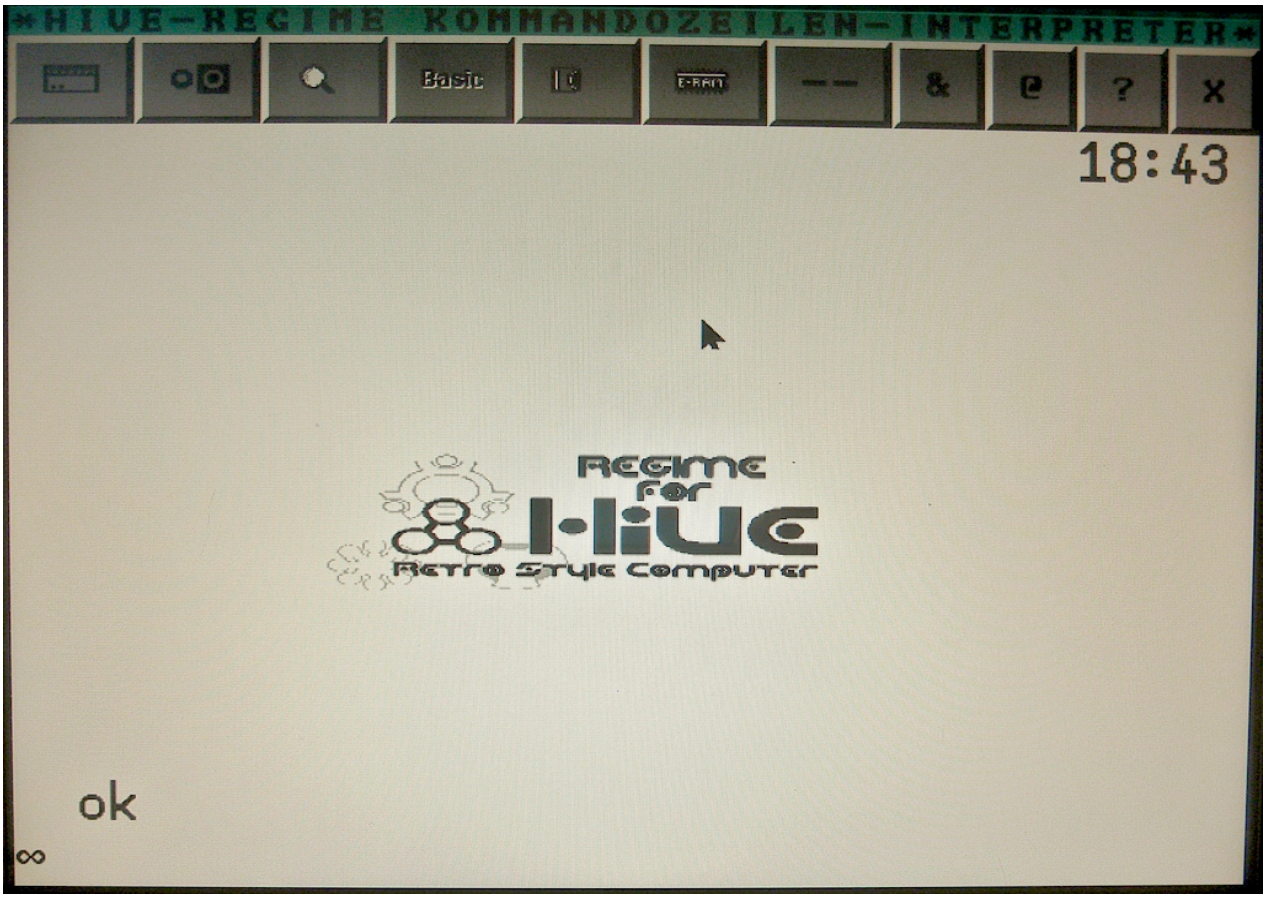
PS.:
Bei Fragen, Hilfestellungen, Kritik und was euch sonst noch einfällt, erreicht Ihr mich im Hive-Forum auf „hive-project.de“

HINWEIS:

Ich bin kein Schriftsteller, also seht es mir nach, wenn die Struktur dieses Handbuches oder der allgemeine Satzbau nicht so professionell ist, wie einige (gerade Programmierer) es gewohnt sind. Dieses Handbuch ist gerade für die Beginner unter uns gedacht und bewusst (versucht) einfach gehalten auch was die Programmbeispiele angeht.

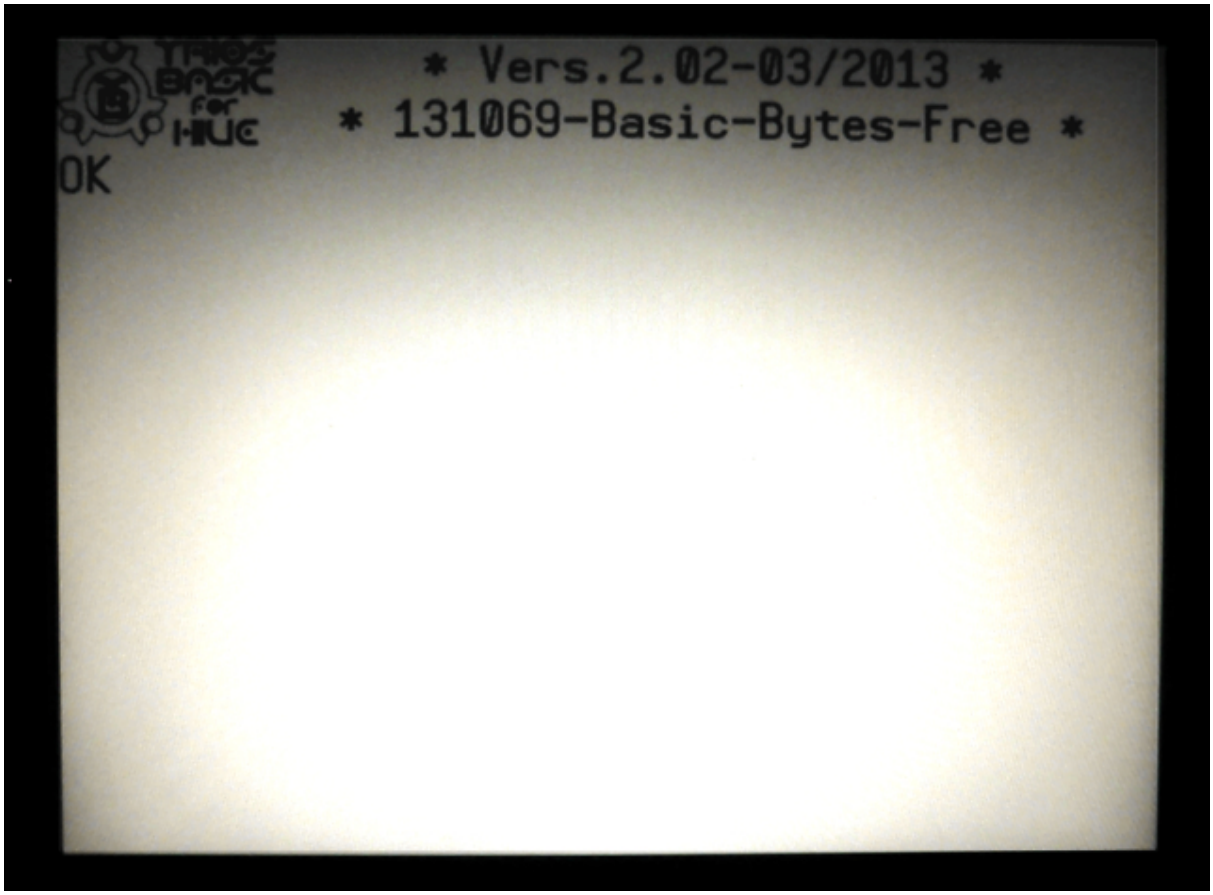
Installation von TRIOS-BASIC

Die Installation von Trios-Basic gestaltet sich sehr einfach. Entpacke das Paket „Basic.zip“ auf eine SD-Karte. Die Dateien Bel.sys und reg.sys sind angepasste Versionen und werden zum Betrieb von TRIOS-BASIC benötigt, deshalb müssen sie ebenfalls auf die SD-Card kopiert werden.
Starte Deinen Hive und gib in der Regime-Kommando-Zeile cd Basic ein. Betätige ENTER. Gib nun Basic ein und betätige wieder ENTER.
Alternativ kannst du auch mit der Maus auf den Knopf mit der Aufschrift BASIC in der oberen Befehlsleiste klicken.
Nun sollte nach einer kurzen Ladezeit der im nächsten Abschnitt gezeigte Bildschirm zu sehen sein.



Angepasste Version des Regime-Kommandozeileninterpreters

Der Arbeitsbildschirm



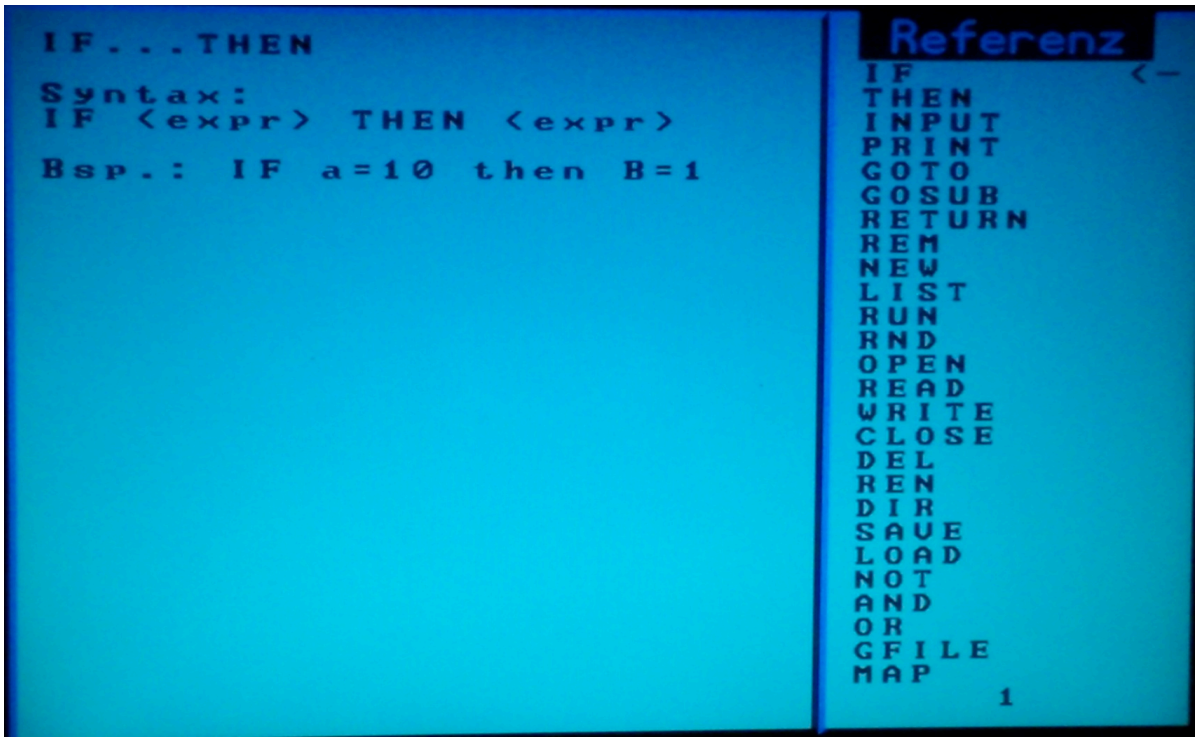
Nun sind wir schon für erste Schritte in der Programmierung des Hive-Computers bereit.

Die Schnellzugriffstasten

Die Schnellzugriffstasten (oder auch Funktionstasten genannt), wurden eingerichtet um häufig verwendete Funktionen im „schnellen Zugriff“ zu haben und so die Arbeit mit TRIOS-Basic flüssiger zu gestalten.

F1 HELP-Die integrierte Hilfefunktion

Mit Druck auf diese Taste wird die integrierte Kurzhilfe gestartet. In ihr werden alle implementierten Befehle kurz beschrieben und im einen oder anderen Fall mit Beispielen untermalt. Diese Hilfe ist aber nur zum kurzen Nachschlagen des Syntax-Aufbaus und der Wirkung, welche der jeweilige Befehl erzielt, gedacht. Für detaillierte Erläuterungen reichen weder der Platz im Propeller, noch der verfügbare Bildschirm aus, weswegen dieses Handbuch entstanden ist.
Zum Arbeiten reicht diese Hilfe jedoch mehr als aus, spart sie doch ein ständiges Nachschlagen im Handbuch. Probiert sie einfach mal aus und entscheidet selbst über Sinn und Zweck dieser Hilfe.
Navigiert wird mit den Tasten Hoch/Runter für die Auswahl des Befehls,welcher angezeigt werden soll. Die Tasten Bildhoch/Bildrunter rufen eine neue Seite (1-5) mit Befehlen auf. Die Taste ESC beendet die Hilfefunktion ,löscht den Bildschirm und kehrt zum Trios-Basic zurück.



F2 Load-Programme laden
F3 SAVE-Programme speichern

Die LOAD-Taste wird im einfachsten Fall ohne weiteren Parameter verwendet. Diese Schnellladefunktion ist gerade beim Experimentieren gut geeignet um ein mit F3 (ohne Parameter) gesichertes Programm als Arbeitskopie schnell wieder in den Speicher zu laden. Man erspart sich die Eingabe eines Dateinamens und das Programm startet automatisch.
F3 ohne Parameter sichert das aktuell im Speicher befindliche Programm als BAS.TMP auf der SD-Card.
Man kann also mit F3 und ENTER sein momentanes Programm sofort speichern und mit F2 und ENTER wieder laden und starten. So kann man seine Programmier-fortschritte sichern und laden ohne jedes mal einen Dateinamen zu vergeben und auf der SD-Card unnötig Unordnung zu stiften.

Die Befehle LOAD und SAVE werden zu einem späteren Zeitpunkt noch genauer erklärt. An dieser Stelle sollte diese Erläuterung reichen.

F4 DIR-DIRECTORY anzeigen

Mit dieser Taste wird das momentan gewählte Verzeichnis auf der SD-Card und die darin befindlichen Dateien angezeigt in der Form:

Dateiname Grösse Tag Monat Jahr der Erstellung

Da es vorkommen kann, das mehr Dateien auf der SD-Card vorhanden sind als der Bildschirm auf einmal darstellen kann, stoppt die Ausgabe nach 9 Dateien und wartet auf einen Tastendruck. Mit der Taste ESC wird die Ausgabe abgebrochen, jede andere Taste wiederholt die Ausgabe, bis alle Dateien angezeigt wurden und zeigt am Ende die Anzahl der gefundenen Dateien an.

F5 RUN-Programm starten

Diese Taste startet ein im Speicher befindliches Programm sofort. Befindet sich kein Programm im Speicher, erscheint das OK und der Cursor blinkt eine Zeile tiefer.

F6 LIST-Programmlisting anzeigen

Die Taste F6 listet das aktuell im Speicher befindliche Programm auf dem Bildschirm aus. Nach 10 Zeilen stoppt die Ausgabe und der Cursor blinkt schneller. Taste ESC beendet die Ausgabe, jede andere Taste setzt die Ausgabe fort, bis alle Zeilen angezeigt wurden.

F7 EDIT-Programmzeile editieren

Taste F7 ruft den Befehl EDIT auf. Gibst Du nur die Programmzeile ein, welche Du editieren willst und drücke ENTER.
Nähere Erklärungen findest Du im Abschnitt „Systembefehle – EDIT“.

F8 TRON-Debugmodus einschalten F9 TROFF-Debugmodus ausschalten.

Diese beiden Tasten schalten den Debugmodus ein bzw. aus. Im eingeschalteten Zustand wird bei Ausführung eines Programms, die gerade vom Interpreter bearbeitete Zeile angezeigt. Dies ist speziell bei der Fehlersuche sehr hilfreich. **Diese beiden Befehle sind nur über die Funktionstasten F8 und F9 aufrufbar.**

F10 RECLAIM-Programm wiederherstellen

Wer kennt das Problem nicht? Man hat ein geniales Programm geschrieben und will es testen, ein Fehler im Programm sorgt dafür, das der Computer nicht mehr auf Eingaben reagiert und der einzige Weg, um ihn wieder zu beleben ist der Reset-Taster. Suuuuper Programm weg und zig graue Haare mehr auf dem Haupt.
Das gehört mit TRIOS-BASIC nun zur Vergangenheit. Da mein lichtes Haupt diese schmerzhafte Erfahrung nur allzu oft gemacht hat, habe ich die Funktion RECLAIM in's Leben gerufen. Dieser Befehl stellt ein im Speicher befindliches Programm wieder her und man kann unbeeindruckt weiterarbeiten.

HINWEIS:
Einzige Voraussetzung für die erfolgreiche Wiederherstellung ist, das der Hive zwischenzeitlich nicht ausgeschaltet wurde oder andere Programme den Inhalt des E-

RAM's verändert haben.**Deshalb ist es auch wichtig, die im TRIOS-BASIC mitgelieferte reg.sys zu verwenden.** Die Original-Reg.sys benutzt die RAM-DISC-Funktion, welche den E-Ram überschreibt.
Es wird der gesamte Basic-Speicher durchsucht. Befindet sich kein Programm im Speicher, wird dies mit der Meldung NO PROGRAMM IN MEMORY quittiert.
Dieser Befehl ist nur über die Funktionstaste F10 erreichbar!

F12 Basic beenden

Mit ihr wird TRIOS-Basic beendet und kehrt zurück zu REGIME. Dabei erscheint das Wort BYE... auf dem Bildschirm.

Der Kommandomodus

Nach dem Start von TRIOS-Basic befindet sich der Hive-Computer im sogenannten Kommandomodus, zu erkennen am blinkenden Cursor, welcher auf Eingaben wartet. Im Kommandomodus können schon alle Befehle direkt ausgeführt werden ähnlich einem Taschenrechner, weswegen dieser Modus auch Taschenrechner-Modus genannt wird.
Gib zum Beispiel : Print 123+67 ein und betätige die Enter-Taste.
Die Operation wird sofort ausgeführt und der Hive gibt das Ergebnis 190 aus.
Dies funktioniert mit allen vier Grundrechenarten (+,-,*,/).
Aber auch Textausgaben sind direkt ausführbar.
Gib jetzt Print „Hallo HIVE“ ein, wobei die Anführungszeichen mit eingegeben werden müssen, da sie dem Basic-Interpreter sagen, das eine Zeichenkette verarbeitet werden soll.
Brav gibt Dein Hive diese Zeichenkette (ohne Anführungszeichen) aus.
Gut, diese Beispiele sind nicht besonders spektakulär aber folgendes Beispiel ist vielleicht doch schon etwas aufregender.
Gib folgendes in der Kommandozeile ein:

CD“dmp“
PLAY“Comic.dmp“
Wenn Du keinen Fehler gemacht hast, solltest Du jetzt aus den angeschlossenen Lautsprechern (bei Aktivboxen einschalten nicht vergessen!) Musik mit nach C64 klingendem Sound hören.
Um die Musikausgabe zu beenden gib

Play0
ein(nach dem Play eine Null eingeben, kein O).

Was haben wir gerade gemacht?

Der Befehl CD“DMP“
veranlasst Trios-Basic dazu das Unterverzeichnis DMP auf der SD-Karte zu öffnen (hier sind alle Musikdateien abgelegt).

Play“Comic.dmp“
Wird jeder sicher schon erraten haben, spielt die Musik-Datei Comic.dmp ab.

Play0
stoppt den Player.
Das war sicherlich schon beeindruckender und zeigt, wie einfach die Programmierung in TRIOS-Basic ist.
Ich hoffe, jetzt ist Dein Entdeckerdrang geweckt worden und du probierst noch

andere Befehle aus (siehe Befehlsübersicht).

HINWEIS:
Nicht alle Befehle erzeugen eine sofort sichtbare Wirkung, manche sind nur in Verbindung mit anderen Befehlen sinnvoll, manche hängen von anderen Befehlen direkt ab und sind allein für sich unsinnig.

Der Programmmodus

Der Programmmodus ist der Modus in dem sich der Interpreter nach dem Start eines Programms befindet, innerhalb dessen die Abarbeitung von Anweisungen innerhalb des gestarteten Basic-Programms stattfindet.
Diese Anweisungen dienen der eigentlichen Problemlösung und werden nacheinander vom Basic Interpreter analysiert und direkt ausgeführt.
Damit eine Programmzeile vom Basic-Interpreter fehlerfrei abgearbeitet werden kann, sind bestimmte Restriktionen im Aufbau einer Basic-Zeile einzuhalten.

1. Basic-Zeilen beginnen immer mit der Zeilennummer, diese muss einen Wert zwischen 1 und 65535 haben
2. Es kann niemals zwei Zeilen mit der gleichen Zeilennummer geben. Wird eine schon vorhandene Zeilennummer vergeben, wird die schon vorhandene Zeile überschrieben.
3. Mehrere Befehle in einer Zeile werden durch einen Doppelpunkt voneinander getrennt. Ausnahmen bilden z.Bsp. IF Then Anweisungen.
4. Befehle müssen nicht zwingend groß geschrieben werden da der Interpreter sie automatisch in Großschreibung umwandelt.
5. Die Länge einer Basic-Zeile darf 85 Zeichen nicht überschreiten. Sollte es nötig werden (z.Bsp. Bei Abhängigkeitsabfragen) mehr als 80 Zeichen zu nutzen, verwende die Gosub-Funktion (bei umfangreichen Vergleichen oder Abhängigkeiten sollten Gosub-Unterprogramme Verwendung finden, das erhöht die Lesbarkeit des Programms).

Hier noch ein paar Tipps:

1. Gib pro Zeile möglichst nur einen Befehl ein, das erleichtert Korrekturen (man muss eine superlange Zeile bei eventuellen Änderungen nicht noch einmal eingeben.) und der Interpreter kommt nicht so schnell durcheinander.
2. Wähle die Schrittweite zwischen den Zeilennummer nicht zu klein und nicht zu groß. Bewährt hat sich ein Abstand von 10 (10,20,30 usw.)
3. Benutze sooft wie möglich (nötig) die REM-Funktion für erläuternde Kommentare, das erleichtert das Lesen eines Programms ungemein und lässt auch andere User Deine Programme verstehen.
4. Gosub-Unterprogramme stets ans Ende des Basic-Programms legen, das erleichtert die Übersicht und lässt sich beim Programmablauf auch besser handhaben (sonst muss man ständig mit GOTO-Anweisungen die Unterprogramme überspringen und zurückspringen ->unübersichtlich)
5. Benutze nach jeder Änderung die Schnellspeicherfunktion (F3 und Enter), um Deine Programmier-Fortschritte vor Verlust zu sichern.

Das DEMO-Programm laden

Jetzt wollen wir aber endlich loslegen, um das Potential von TRIOS-Basic kennenzulernen.
Auf Deiner SD-Card befindet sich ein Demo-Programm, welches Dir einen kleinen Querschnitt von den Möglichkeiten des Basic-Interpreters zeigen wird.
Drücke die Taste F2 und vervollständige die Ladeanweisung mit „demo.bbs“,1 und drücke die Enter-Taste.
Nach einer kurzen Ladezeit startet das Demo-Programm automatisch.
Schau es Dir an und folge den Anweisungen.

Du kannst ein laufendes Programm jeder Zeit mit der Taste ESC abbrechen.
Der Hive quittiert dies mit einem System-Beep und dem Hinweis : „Break in Line:“ gefolgt von der Zeilennummer, in welcher der Interpreter zum Zeitpunkt des Abbruchs gerade war.

Fehlermeldungen

Natürlich kann es sein, das sich im laufe der Programmierarbeit, Fehler einschleichen. Dies können simple Schreibfehler , Dateifehler oder auch logische Fehler sein. Dies ist nicht weiter schlimm, jeder Fehler wird vom Interpreter abgefangen und mit einer entsprechenden Meldung quittiert welche auf die Art des Fehlers hinweist.
Ein Festfahren oder Aufhängen sollte also nicht vorkommen.
Sollte es aber doch einmal passieren, das der Hive auf Eingaben nicht mehr reagiert , keine Panik. Reset-Taste drücken, um den Hive neu zu starten und mit F2 und Enter die hoffentlich angelegte Sicherungskopie des Programms laden und eventuelle Fehler suchen.

HINWEIS:
Erfahrungsgemäß ist die letzte Sicherung immer schon zu lange her oder man hat im Eifer des Gefechts vergessen eine Sicherung anzulegen.
In diesem Fall ist normalerweise der Nervenzusammenbruch nahe.
Jeder kennt diesen Moment der totalen Verzweiflung ABER im Trios-Basic gibt es für diese Fälle die erhoffte ERLÖSUNG.
Die Funktionstaste F10 **RECLAIM** sorgt dafür, das das wertvolle Gedankengut wieder herstellbar ist. **Einzige Vorraussetzung dafür ist, das der Hive zwischenzeitlich nicht ausgeschalten wurde und keine anderen Programme den Inhalt des externen Ram's verändert haben**.

Variablen

Variablen sind in der Mathematik wie auch in der Programmierwelt Platzhalter für veränderliche Zahlenwerte. Das praktische an ihnen, ist die Verwendung in Abfragen oder Formeln ohne das man deren genauen Wert kennen muss bzw. dieser sich jeder Zeit ändern kann und dadurch die Formeln bzw. Abfragen allgemeingültig werden und mehrfach verwendet werden können.

Beispiel: a+b=c

Diese simple Formel bleibt gültig, egal welchen Wert a oder b haben.
Weist man nun a und b Werten zu, wird die Formel konkret und man erhält das Ergebnis c.

In TRIOS-Basic werden zwei Arten von Variablen unterschieden. Dies sind

1. numerische Variablen (zu deutsch Zahlen)
2. Zeichenketten-oder String-Variablen (oder einfach gesagt Text)

Diese beiden Arten von Variablen unterscheiden sich nicht nur inhaltlich, sondern auch optisch voneinander.

Mit der Anweisung : a=145 wird der numerischen Variablen a, der Wert 145 zugewiesen. Gibt man nun PRINT a ein und betätigt die Entertaste, erscheint die Zahl 145 auf dem Bildschirm. Das funktioniert mit jeder anderen Variablen im Bereich a-z, wobei es keinen Unterschied macht, ob der Variablenname Groß- und klein geschrieben wird.

Weise nun der Variablen b einen Wert zu und gib die im oberen Beispiel gezeigte Formel ein. Überprüfe das Ergebnis mit PRINT c.

HINWEIS:

1. Variablen werden mit Buchstaben von a-z bezeichnet
2. Ob Groß- oder kleingeschrieben wird, spielt dabei keine Rolle
3. Wird einer Variablen mehrfach ein Wert zugewiesen, entspricht der Inhalt der Variablen immer der letzten Zuweisung.

Strings

Kommen wir nun zur zweiten Variablenart, den Strings- oder Zeichenketten-Variablen.
Diese funktionieren in der Zuweisung genauso wie die numerischen Variablen, ihnen wird aber zur Kennzeichnung für den Interpreter eine # (Raute) vorangestellt.

Beispiel: #a="Hallo Welt"

Ob die Zuweisung erfolgreich war kann mit PRINT #a überprüft werden.
Das Raute-Zeichen teilt dem Interpreter mit, das es sich um eine Stringvariable handelt. Nach dem Gleichheitszeichen folgt die Zeichenkette, welche zugewiesen werden soll. Diese muss in Anführungszeichen stehen, wie jeder Text, der im TRIOS-Basic verarbeitet werden soll. Nur so kann der Interpreter zwischen Zeichenketten und Zahlen unterscheiden.
Da man Text nicht zusammenrechnen kann funktionieren hier natürlich die Grundrechenarten nicht. Die nachträgliche Manipulation von Strings erfolgt mit speziellen Befehlen (siehe dazu auch den Abschnitt – Zeichenkettenfunktionen).

HINWEIS:
- Stringvariablen haben den Bezeichner #a-#z (Groß- oder kleingeschrieben) Zeichenketten welche Strings zugewiesen werden ,müssen in Anführungszeichen stehen
- Wird einer Stringvariablen mehrfach ein Wert zugewiesen, entspricht der Inhalt der Variablen immer der letzten Zuweisung.
- Strings werden mit speziellen Befehlen manipuliert, eine Bearbeitung mit mathematischen Funktionen ist nicht erlaubt.
- Die Länge eines Strings darf bis zu 33 Zeichen betragen, siehe hierzu den Abschnitt „String-Arrays"

Arrays - dimensionieren mit DIM

Mit Hilfe eines Arrays (Feldes) können die Daten eines üblicherweise einheitlichen Datentyps so im Speicher eines Computers abgelegt werden, dass ein Zugriff auf die Daten über einen Index möglich wird (Quelle Wikipedia).
Im TRIOS-Basic ist die Verwendung von bis zu dreidimensionalen Arrays möglich. Arrays werden bei numerischen und String-Variablen unterstützt. Dabei wird die Variablenzuweisung durch den Index erweitert.

Beispiel:

A(1)=345

In diesem Beispiel wird der Variablen A mit dem Index 1 der Wert 345 zugewiesen. Die Standardindexweite ist auf 10 eingestellt, das heisst der Index von (in diesem Fall) A kann zwischen 0 und 10 betragen. Werden größere Indexwerte benötigt, so ist das Feld mit der Anweisung DIM A(Indexwert) entsprechend zu dimensionieren. Bei eindimensionalen Feldern (so wie in diesem Fall) darf der Indexwert Werte von 0 bis 511 annehmen.
Das heisst A kann 512 verschiedene Werte enthalten, die über den Indexwert zugewiesen oder abgerufen werden können, wobei A(0)=123 der Standardzuweisung A=123 entspricht.
Werden zwei-oder dreidimensionale Felder benötigt, verringert sich der Indexwert je Dimension entsprechend (die Gesamtfeldgröße darf 512 Elemente nicht überschreiten). Das bedeutet, ein zweidimensionales Feld ist maximal 23x22 Felder groß (DIM A(22,21) da ein Feld immer bei 0 beginnt) und ein dreidimensionales Feld kann nur 8x8x8 Felder groß sein (DIM A(7,7,7)).
Eine Überschreitung dieser Werte wird mit einer Fehlermeldung quittiert.

Durch die Verwendung mehrdimensionaler Felder wird die Verwendung von nur 26 Variablen (a-z=26 verschiedene Variablen) aufgebrochen und es sind nun 26*512=13312 verschiedene Zuweisungen möglich.

String-Arrays

Für die Verwendung von String-arrays gilt das bisher gesagte entsprechend.
Die Standarddimension beträgt ebenfalls die Weite 10 (#a(0)...#a(10)).
Da bei Stringvariablen aber von der Zeichenanzahl viel mehr Daten verarbeitet werden müssen, gibt es im Gegensatz zu den numerischen Variablen einige Einschränkungen zu beachten.

HINWEIS:
Werden Strings in String-Arrays verarbeitet, darf die Länge eines Strings maximal 33 Zeichen betragen. **Dies wird vom Interpreter gnadenlos durchgesetzt. Alle Strings, die diese Beschränkung nicht einhalten, werden abgeschnitten. Nur so kann eine Überschneidung der Felder verhindert werden.**
Ist der String länger geschieht folgendes:

Bsp.:
```
10 #a(0)="Dies ist eine sehr lange Zeile, viel zu lang für ein String-Array"
20 PRINT #a
```

kleiner Hinweis: Die Zuweisung #a(0)="Zeichenkette" entspricht der Standardzuweisung #a="Zeichenkette"

Auf dem Bildschirm erscheint:
Dies ist eine sehr lange Zeile,

Wo ist der Rest?

Ganz einfach, im Speicher ist für jeden String-Array ein Bereich von 33 Zeichen reserviert. Wird diese Länge überschritten, schneidet der Basic-Interpreter den Rest des Strings ab um Überschneidungen in folgende Feldbereich zu verhindern.

Also, werden Strings in Arrays verwendet, unbedingt auf die Längenbegrenzung von 33 Zeichen pro Array-Element achten.

Befehle zur Steuerung des Programmablaufes

(ON).. GOTO	Berechnete und unbedingte Sprünge
(ON) … GOSUB	Berechnete und unbedingte Unterprogramm-Aufrufe
IF … THEN … ELSE	Bedingungsabfrage
FOR … NEXT	Wiederholungsanweisung
RUN	Programmstart
PAUSE	Programm anhalten
END	Programm beenden

Unbedingter Sprung

Syntax: GOTO *Zeilennummer*

Der Befehl GOTO dient dazu die Programmabarbeitung an einer anderen Stelle im Programm fortzuführen. *Zeilennummer* ist dabei die Nummer der Basic-Zeile, an der die Programmabarbeitung weiter gehen soll.

Bsp.: 10 PRINT“Hier startet das Programm“
20 PRINT“Ab hier geht es weiter“
30 GOTO 20 :REM Sprung nach Zeile 20

Das Beispiel gibt zuerst den Text „Hier startet das Programm“ auf dem Bildschirm aus und danach den Text „Ab hier geht es weiter“ in einer Endlosschleife, da mit der GOTO Anweisung immer wieder in die Zeile 20 gesprungen wird. Das Programm muss mit der ESC-Taste unterbrochen werden.

Berechneter Sprung

Syntax: ON *Ausdruck* **GOTO** *Zeilennummer*

Ausdruck stellt einen numerischen Wert zwischen 1 und der Anzahl der nach GOTO angegebenen Zeilennummern dar. Der Wert 0 führt dazu, das die ON GOTO Anweisung ignoriert (übersprungen) wird. Ist der Wert von *Ausdruck* größer als die Anzahl der Zeilennummern nach der GOTO Anweisung, wird die letzte Zeilennummer als Sprunganweisung benutzt.

Bsp.:

```
10 CLS:REM Bildschirm löschen
20 PRINT"****Auswahlmenue****"
30 PRINT"* 1.Dateneingabe          *"
40 PRINT"* 2.Datenausgabe          *"
50 PRINT"* 3.Berechnung            *"
60 PRINT"* 4.Programmende          *"
70 INPUT"Geben Sie die Zahl des Menuepunktes ein";A
60 IF A>4 THEN GOTO 70
80 ON A GOTO 100,200,300,400
100 REM Programmteil Dateneingabe
  :
  :
  :
190 GOTO 10
200 REM Programmteil Datenausgabe
  :
  :
  :
290 GOTO 10
300 REM Programmteil Berechnung
  :
  :
  :
390 GOTO 10
400 REM Programmende
410 END
```

Im Beispiel erscheint ein Programmauswahlmenue auf dem Bildschirm, bei dem der Benutzer unter Angabe der Funktionsnummer, die einzelnen Programmteile aufrufen kann.

Unbedingter und Berechneter Unterprogrammaufruf

Syntax: ON *Ausdruck* **GOSUB** *Zeilennummer* **RETURN**

Anders als bei der GOTO-Anweisung dient der GOSUB-Befehl dazu, in Unterprogrammteile zu verzweigen und nach Abarbeitung dieser mittels des Befehls RETURN wieder an den Punkt nach der GOSUB Anweisung zurückzukehren.
Hat Ausdruck den Wert 0, wird die gesamte ON GOSUB Anweisung ignoriert (übersprungen) und die Programmabarbeitung wird nach dieser fortgeführt.

Bsp.:

```
10 CLS:REM Bildschirm löschen
20 PRINT“****Auswahlmenue****“
30 PRINT“* 1.Dateneingabe          *“
40 PRINT“* 2.Datenausgabe          *“
50 PRINT“* 3.Berechnung            *“
60 PRINT“* 4.Programmende          *“
70 INPUT“Geben Sie die Zahl des Menuepunktes ein“;A
80 IF A=4 THEN END: REM Programmende
90 ON A GOSUB 100,200,300
60 GOTO 10
100 REM Programmteil Dateneingabe
  :
  :
  :
190 RETURN
200 REM Programmteil Datenausgabe
  :
  :
  :
290 RETURN
300 REM Programmteil Berechnung
  :
  :
  :
390 RETURN
  :
  :
1000 REM Nächster Programmteil
```

Das Beispiel ist dem vorhergehenden sehr ähnlich, nur wurden hier statt GOTO, GOSUB Anweisungen benutzt.

Bedingungsabfragen

Syntax: IF *Bedingung* **THEN** *Anweisung* **ELSE** *Ausweichanweisung*

Bedingungsabfragen sind die in der Programmierwelt wohl am häufigsten verwendeten Befehle zur Programmsteuerung. Mit ihnen ist es erst möglich, den Programmablauf den Anforderungen der zu erfüllenden Aufgaben entsprechend zu steuern.
Bedingung stellt die zu erfüllende Bedingung für die nach THEN angegebene Anweisung dar. Ist die Bedingung nicht erfüllt kann mit dem Befehl ELSE eine Ausweichanweisung definiert werden. *Anweisung* und *Ausweichanweisung* kann eine mathematische Operation, eine Sprunganweisung, eine Bilschirmausgabeanweisung oder ähnliches sein.

Bsp.:

```
10 B=MF INT(RND(5))
20 CLS
30 INPUT“Gib eine Zahl ein“;A
40 IF A=B THEN PRINT“Richtig geraten“
50 ELSE PRINT“Falsch geraten“
60 PAUSE 2000:GOTO 20
```

In diesem Beispiel wird der Variablen B ein Zufallswert zwischen 0 und 5 zugewiesen. Diese Zahl sollst Du erraten. Ist der eingegebene Wert für A richtig, erfolgt in Zeile 40 nach der Überprüfung die Ausgabe „Richtig geraten!“.
Gibst Du den falschen Wert ein, wird in Zeile 50 die ELSE-Ausweichanweisung ausgeführt und entsprechend „Falsch geraten!“ auf dem Bildschirm ausgegeben.

Programmstart

Syntax: RUN

Die Anweisung RUN startet das im Speicher befindliche Basic-Programm ab der ersten Programmzeile. Alternativ kann auch die Taste F5 gedrückt werden.
Programmzeilen dürfen mit 1 bis 65534 nummeriert werden.

Programmende

Syntax: END
Der Befehl END beendet die Programmausführung und Trios-BASIC kehrt in den Kommandomodus zurück.

Programmunterbrechung für eine bestimmte Zeit

Manchmal ist es nötig oder wünschenswert, den Programmablauf für eine bestimmte Zeit zu unterbrechen (z.Bsp.:für Texthinweise, welche für ein paar Sekunden zu sehen sein sollen o.ä.)
Für diesen Fall gibt es folgenden Befehl

Syntax: PAUSE <*Zeitwert*>

Wobei Zeitwert die Dauer der Programmunterbrechung in Millisekunden darstellt.

Bsp.: PAUSE 500

Der Programmablauf wird für 500 Millisekunden angehalten (½ Sekunde).

FOR...NEXT Wiederholungsschleifen

Syntax: FOR <*Variable*>=*Wert* **TO** *Wert2* **<*STEP* *Schrittweite*>**

For Next Schleifen werden benutzt um bestimmte Vorgänge eine festgelegte Anzahl oft zu wiederholen. Dabei ist es möglich den Start und Endwert sowie die Schrittweite der Zählschleife festzulegen. Der Befehl Next markiert das Ende der Schleife und darf nicht vergessen werden. Alle Programmzeilen zwischen FOR und NEXT werden in die Zählschleife einbezogen.

Bsp.:

```
10 CLS
20 FOR i=1 TO 10 STEP 2
30 PRINT i
40 NEXT i
```

Ausgabe:

```
1
3
5
7
9
```

Im obigen Beispiel wird eine Zählschleife von 1 bis 10 erstellt, welche eine Schrittweite von 2 haben soll. Nun wird mit 1 beginnend hochgezählt. Durch die Schrittweitenfestlegung wird bei jedem Durchgang nicht um eins sondern um 2 erhöht bis die Zahl 10 erreicht ist. Deshalb erfolgt die Ausgabe auch entsprechend.

Datums- und Zeit-Funktionen

Stellen der internen Uhr

Syntax: STIME *hh:mm:ss*

hh=Stunden mm=Minuten ss=Sekunden

Bsp.: STIME 23:45:00

Stellt die Uhr auf die Uhrzeit 23:45:00 (**Doppelpunkte nicht vergessen**).

Einstellen des Datums

Syntax: SDATE *tt.mm.yyyy*

tt=Tag mm=Monat yyyy=Jahr

Bsp.: SDATE 23.4.2013

Stellt das Datum 23.April 2013 ein.

Einstellen des aktuellen Wochentages

Syntax: SDOW <*Tag*>

Tag repräsentiert eine Zahl zwischen 1 und 7 für den jeweiligen Wochentag (z.Bsp. 1 für Montag und 7 für Sonntag)

Bsp.:SDOW 2 setzt den Wochentag Dienstag

Abfragen der aktuellen Uhrzeit

Syntax: GTIME *(Teil)*

Mit dem Befehl GTIME kann die aktuelle Uhrzeit als einzelne Parameter abgefragt werden, wobei „Teil“ eine Zahl zwischen 1 und 3 darstellt, mit welcher folgende Zeitwerte abgefragt werden können.

GTIME(1)-gibt die aktuelle Stunde zurück
GTIME(2)-Rückgabe der aktuellen Minute
GTIME(3)-Sekundenwert wird zurückgegeben

Bsp.:

```
10 CROFF:CLS
20 PRINT“Jede Minute ertoent ein Piepton!“
30 a=GTIME(3)
40 POS 10,4:PRINT GTIME(1);“:“;GTIME(2);“:“;a;
50 IF a=0 THEN BEEP 88
60 GOTO 30
```

In obigen Programm wird in Zeile 30 die Variable a mit dem aktuellen Sekundenwert geladen. In Zeile 40 wird die aktuelle Uhrzeit ausgegeben und in Zeile 50 wird überprüft, ob der Wert von a gleich null ist (Minute beginnt). Wenn ja erfolgt die Ausgabe eines akustischen Signals.
Durch die Anweisung GOTO 30 (in Zeile 60) wird eine Schleife gebildet, welche bei Erreichen von Zeile 60 wieder zurück in Zeile 30 springt und die Überprüfung beginnt von neuem.

Abfragen des aktuellen Datums

Syntax: GDATE *(Teil)*

GDATE wird dafür benutzt, das aktuelle Datum als einzelne Parameter abzufragen, wobei „Teil“ eine Zahl von 1 bis 4 darstellt, welche folgende Parameter übergibt.

GDATE(1)-Rückgabe des aktuellen Tagesdatums (z.Bsp. Der 24. eines Monats)
GDATE(2)-Rückgabe des aktuellen Monats (von 1-12)
GDATE(3)-Das Jahr wird zurückgegeben (z.Bsp.:2013)
GDATE(4)-Rückgabe des Tages der Woche (1-7 für Montag-Sonntag)

Bsp.:

```
10 CLS
20 a=GDATE(4)
30 ON a GOSUB 100,110,120,130,140,150,160
40 PRINT“Heute ist “;#a;“ , der „;GDATE(1);“.“;GDATE(2);“.“;GDATE(3)
50 END
100 #a=“Montag“
105 RETURN
110 #a=“Dienstag“
115 RETURN
120 #a=“Mittwoch“
125 RETURN
130 #a=“Donnerstag“
135 RETURN
140 #a=“Freitag“
145 RETURN
150 #a=“Samstag“
155 RETURN
160 #a=“Sonntag“
165 RETURN
```

Zeitanzeige

Syntax: TIME *x,y*

Gibt die aktuelle Uhrzeit im Format HH:MM:SS an Position x,y aus.
Diese Funktion ist praktisch, wenn im laufenden Programm eine Uhr angezeigt werden soll. In einer Schleife fungiert sie als Digitaluhr

Bsp.:

```
10 CROFF:CLS
20 TIME 12,0
30 GOTO 20
```

Die Verwendung von Timer

Syntax: TIMER <*Option*>

In TRIOS-BASIC sind für zeitabhängige Funktionen 8 unabhängige Timer programmierbar welche mit Werten gesetzt und ausgelesen werden können um Aktionen zu starten.
Dazu gibt es 3 Optionen

TIMER S*(ID,wert)*-setzt einen Timer mit der *ID* mit dem Zählwert *wert*, wobei *ID* die Nummer des Timers ist (1-8), der verwendet wird und *wert* entspricht der Zeit in 100ms (der Wert 10 entspricht 1 Sekunde)

TIMER C*(ID)*

Abfrage ob der Timer mit der Nummer *ID*(1-8) abgelaufen ist.
Rückgabewert 1 bedeutet Timer abgelaufen, Rückgabe 0, Timer läuft noch.

TIMER R*(ID)*

Abfrage des Zählwertes von Timer mit der Nummer *ID*(1-8) .

Bsp.:

```
5 CROFF:CLS
10 TIMER S(1,50)
20 TIMER S(2,60)
30 TIMER S(3,20)
40 TIMER S(4,90)
50 TIMER S(5,130)
60 FOR I=1 TO 5
70 IF TIMER C(I)=1 THEN GOSUB 150
80 NEXT I
100 GOTO 60
150 POS 5,I*2
200 PRINT“Timer „;I;“ ist abgelaufen“
210 RETURN
```

In diesem Beispiel werden 5 Timer mit jeweils unterschiedlichen Zählwerten gesetzt (Zeile 10 bis 50). In der FOR-NEXT Schleife werden nacheinander alle fünf Timer auf „abgelaufen“ überprüft und in Zeile 150 bis 210 jeweils angezeigt.
Überprüfe selbst die Abfolge der Anzeige mit den gesetzten Werten.
Zuerst wird Timer 3 dann 1, 2, 4 und zuletzt Timer 5 als abgelaufen notiert.

Systembefehle

Basic beenden

Syntax: BYE

Der Befehl BYE beendet die aktuelle Basic-Sitzung und kehrt zu REGIME zurück.

HINWEIS:
Sichere ein eventuell im Speicher befindliches Programm bevor Du diesen Befehl verwendest. Am einfachsten mit der Taste F3 und ENTER.

Verwendete Cogs anzeigen

Syntax: COGS *<wert>*

Zeigt die Anzahl der verwendeten Cogs an von

COGS 1 – Administra
COGS 2 – Bellatrix
COGS 3 – Regnatix

COGS ist die Bezeichnung der Prozessorkerne im Propellerchip. Jeder Propellerchip hat 8 Kerne(Cogs).

Der HEX-Speichermonitor

Syntax: DUMP *Adresse,Zeilen,Ort*

Mit dem Befehl DUMP können Speicherbereiche der Regnatix-Speichers oder des externen RAM's angezeigt werden. Dies ist sehr hilfreich bei Arbeiten mit Peek und Poke – Befehlen zur Manipulation von Speicherstellen. Die ist nur eine Anzeige. Speichermanipulationen sind nur mit Hilfe von Poke-Anweisungen möglich.

Adresse gibt den Bereich im Speicher an, bei dem die Anzeige beginnen soll. Die Adressangabe kann dezimal oder hexadezimal erfolgen.

Zeilen sagt dem Interpreter die Anzahl der anzuzeigenden Zeilen an, wobei nach 10

ausgegebenen Zeilen pausiert und auf Tastendruck gewartet wird. Mit ESC wird die Ausgabe abgebrochen.
Ort gibt den Speicherbereich an, der ausgegeben werden soll

0 – Regnatix RAM
1 – externer RAM

Bsp.: DUMP $200,20,1

Gibt 20 Zeilen des externen RAMs ab Adresse hexadezimal $200 (dezimal 512) aus.

Anzeige des freien Basic-Speichers

Syntax: FREE

Zeigt die Größe des noch verfügbaren Basic-Speichers an

Bsp.: PRINT FREE

Basic-Speicher Löschen

Syntax: NEW

Löscht den gesamten Basic-Programmspeicher.

HINWEIS:
Im Gegensatz zu vielen anderen Basic-Interpretern, die nur den Speicherzeiger auf null setzen, löscht TRIOS-BASIC den Speicher wirklich endgültig. Alle Speicherzellen werden mit Nullen überschrieben, also diesen Befehl mit Bedacht verwenden

Programmlisting ausgeben

Syntax: LIST *Startzeile,Endzeile*

Der Befehl LIST ohne Parameter listet das gesamte im Speicher befindliche Programm auf dem Bildschirm auf.

Wird nur Startzeile als Parameter übergeben, wird nur diese Zeile ausgegeben.
Wird Startzeile und Endzeile übergeben, listet der Interpreter alle in diesem Bereich befindlichen Programmzeilen auf.
Existiert keine Zeile mit den in den Parametern angegebenen Zeilennummern, erfolgt auch keine Ausgabe.
Die Ausgabe wird nach jeweils 10 Zeilen pausiert und kann mit Tastendruck fortgeführt werden. Taste ESC beendet die Ausgabe.

Variablenspeicher löschen

Syntax: CLEAR

CLEAR löscht den Inhalt aller Variablen und Strings, den Speicher für DIR-Befehl und den MAP-Speicher sowie die Array-Dimensionen.

Zeileneditierung

Syntax: EDIT *Zeilennummer*

Mit EDIT ist es möglich nachträglich, schon vorhandene Zeilen zu verändern (zu editieren). *Zeilennummer* gibt die zu editierende Zeile an. Folgende Steuertasten werden bei der Editierung unterstützt:

Cursortasten rechts, links	bewegen des Cursors innerhalb der Zeile.
Cursortaste hoch, Ende-Taste	bewegt den Cursor an das Zeilenende
Cursortaste runter, Pos1-Taste	bewegt den Cursor an den Zeilenanfang
Backspace-Taste	Löscht das Zeichen links vom Cursor.

HINWEIS:
Ist die Editierung abgeschlossen, muss der Cursor ans Ende der Zeile gesetzt werden, da die Cursorposition die aktuelle Zeilenlänge markiert. Dafür können die Pfeiltasten (rechts, links) oder die Ende-Taste verwendet werden.Es wird kein Einfügen von Text unterstützt, d.h. es wird immer überschrieben.
Wird in der editierten Zeile die Zeilennummer geändert, legt der Interpreter eine neue Zeile mit dem aktuellen Inhalt an(Kopie).
Denke daran, das schon existierende Zeilen der gleichen Zeilennummer überschrieben werden.

TRIOS-BASIC-Version anzeigen

Syntax: VER

VER kann nur in Verbindung mit dem PRINT-Befehl verwendet werden und zeigt die aktuelle TRIOS-BASIC-Version an.

Bsp.:PRINT VER

Logische Operatoren

Ein logischer Operator ist eine Funktion, die einen Wahrheitswert liefert. Im TRIOS-BASIC gibt es zwei Arten von logischen Operatoren. Dies sind einseitige (NOT) und zweiseitige (AND, OR) Operatoren.
NOT ist ein einseitiger Operator und bezieht sich nur auf den Operanden, der nach ihm steht .

Bsp.:

```
10 a=10:b=20
20 IF NOT a=b THEN GOTO 200
100 PRINT“A und B sind gleich“
110 END
200 PRINT“A ist nicht B“
```

Der AND Operator verknüpft zwei Operanden miteinander und gibt nur dann WAHR zurück, wenn beide Operanden WAHR sind.
Ändere den Wert von a auf a=20 in Zeile 10 und überprüfe die Reaktion.

Bsp.:

```
10 a=10:b=20
20 IF a=10 AND b=20 THEN GOTO 200
100 PRINT“Werte für a und b sind nicht WAHR“
110 END
200 PRINT“Die Werte für a und b sind WAHR“
```

Der OR Operator verknüpft ebenfalls zwei Operanden miteinander und gibt WAHR zurück, wenn mindestens ein Operand WAHR ist.
Ändere die Werte für a und b in Zeile 10 und überprüfe die Reaktion.

Bsp.:

```
10 a=10:b=20
20 IF a>5 OR b<20 THEN GOTO 200
100 Print“Bedingung nicht WAHR“
110 END
200 PRINT“Bedingung WAHR“
```

Ändere den Wert von a auf 4 in Zeile 10 und überprüfe die Reaktion.

Ein- und Ausgabebefehle

Speicherzellen lesen

Syntax: PEEK *Adresse*

PEEK ist ein Speicherzellen-Lesebefehl. Mit ihm kann der Inhalt einzelner Speicherzellen im E-Ram gelesen werden. Da es sich um einen Byte-Lesebefehl handelt liegt der Wertebereich zwischen 0 und 255.

Bsp.:
```
10 a=PEEK $4000
20 PRINT a
```

Im Beispiel wird der Inhalt der Speicherzelle $4000 (dezimal 16384) gelesen und in die Variable a geladen. In Zeile 20 wird der gelesene Inhalt ausgegeben.

Speicherzellen schreiben

Syntax: POKE *Adresse,Wert*

POKE ist das Gegenstück zu PEEK und beschreibt die Speicherzelle an der *Adresse* Mit dem übergebenen *Wert*. Dieser darf im Bereich von 0 bis 255 liegen.

Bsp.:
```
10 a=23
20 POKE $80000,a
30 PRINT PEEK $80000
```

Im Beispiel wird an Adresse $80000 im E-Ram der Wert 23 geschrieben. Zur Überprüfung, ob die Operation erfolgreich war, wird in Zeile 30 die Speicherzelle $80000 wieder ausgelesen und auf dem Bildschirm, deren Inhalt angezeigt (es sollte der Wert 23 erscheinen).

Werteeingabe über die Tastatur

Syntax: INPUT *"Eingabetext";String oder Variable,Variable...*

Mit dem Befehl INPUT ist es möglich diverse Dateneingaben zur Weiterverarbeitung einzugeben, dabei werden beide Variablentypen (numerische und Stringvariablen) unterstützt. Zur Erklärung kann ein Eingabeprompt-Text, der auf die Art der Eingabe hinweist übergeben werden. Danach folgen die einzelnen Variablentypen, die mit

Werten gefüllt werden sollen.

Bsp.:
```
10 CLS
20 INPUT“Geben Sie Ihren Namen und Alter ein:“;#a,a
30 PRINT“Sie heissen „;#a;“ und sind „;a;“ Jahre alt!“
```

HINWEIS:
Stringvariablen können nur einzeln mit INPUT übergeben werden und müssen als erste Variable angegeben werden (INPUT „Name,Alter?:“;#a,a).
Numerische Variablen können durch Komma getrennt eingegeben werden (INPUT“Test“;a,b,c,d,e).
Der Eingabepromt-Text ist nicht zwingend erforderlich und kann auch weggelassen werden (INPUT #a,a).

Kommentare im Programmtext

Syntax: REM *Kommentar*

Der REM-Befehl bietet die Möglichkeit, im Programmtext erklärende Kommentare zu setzen. Sie werden vom Interpreter bei der Programmabarbeitung ignoriert und dienen dazu ein Programm für andere leserlicher zu gestalten. Zudem ist es durch sie auch für den Programmierer leichter sich wieder in seinem Werk zurecht zu finden, wenn er es einige Zeit nicht bearbeitet hat.

HINWEIS:
Mache so oft wie möglich von diesem Befehl Gebrauch, er erleichtert die Arbeit und auch die spätere Editierung ungemein und hilft die Übersicht über große Programme zu wahren.

Tastaturabfragen

Syntax: INKEY

Diese Funktion liefert Informationen über die Betätigung der Tastatur. Der Rückgabewert entspricht dem ASCII-Code der zuletzt gedrückten Taste, wird keine Taste betätigt, ist der Rückgabewert Null. Im Gegensatz zur Input-Anweisung wird hier nicht auf die Betätigung der Tastatur gewartet. Vielmehr muss die Abfrage in

einer Schleife erfolgen um den Tastenwert des Bedieners einzufangen. Dies bedeutet, das Tastaturabfragen möglich sind, ohne den Programmablauf zu unterbrechen und die zyklische Abfrage der Tastatur dazu zu benutzen, das Programm durch die Abfragen entsprechend zu steuern.

Bsp.:
```
10 CLS
20 a=INKEY:IF a=0 THEN GOTO 20
30 PRINT CHR$(a),a,
40 GOTO 20
```

Im Beispiel wird die Tastatur in Zeile 20 in einer Schleife abgefragt, die erst beendet wird, wenn eine Taste betätigt wurde.
In Zeile 30 wird das dem Eingabewert a entsprechende Zeichen und der zugehörige ASCII-Zahlenwert ausgegeben. Im folgendem Beispiel wollen wir das Programm entsprechend der gedrückten Tasten steuern.

Bsp.:
```
10 CLS
20 a=INKEY:IF a=0 THEN GOTO 20
30 IF a=101 THEN GOTO 100
40 IF a=97 THEN GOTO 200
50 GOTO 10
100 PRINT“Du hast das e gedrückt“
110 GOTO 10
200 PRINT“Nun wurde das a betätigt“
210 GOTO 20
```

In den Zeilen 30 und 40 wird der Inhalt der Variablen a mit den Vorgabewerten 101 und 97 verglichen (101=ASCII-Wert Taste e, 97=ASCII-Wert Taste a) und entsprechend in unterschiedliche Programmteile verzweigt.
Experimentiere mit den Werten etwas herum und überprüfe die Reaktionen.

HINWEIS:
INKEY muss in einer Schleife abgefragt werden. Wird keine Taste betätigt, ist der Rückgabewert Null. Bei Betätigung entspricht der Rückgabewert dem ASCII-Wert der betätigten Taste.

Befehle zur Benutzung der Maus

Im Gegensatz zu den meisten Basic-Interpretern (nicht nur bei den Interpretern für den Propeller Chip), ist es im TRIOS-BASIC möglich Interaktionen im Programm mit Hilfe der am HIVE angeschlossenen Maus durchzuführen.
Hierfür wurden die im Folgenden beschriebenen Befehle integriert.

Mausbedienung ein/ ausschalten

Syntax: MOUSE *on,color*

Um überhaupt mit der Maus arbeiten zu können, müssen wir dem Interpreter mitteilen, das wir dies tun wollen. Dafür gibt es diesen Befehl. Er muss zwingend vor der ersten Maus-Interaktion im Programm stehen, da ansonsten alle Maus-Befehle vom Interpreter ignoriert werden.
On bezeichnet einen Wert zwischen 0 und 1 wobei die Null für ausgeschalteten und eins für eingeschalteten Maus-Pfeil steht. Mit dem Parameter *color* wird die Farbe des Maus-Pfeils gesetzt und kann Werte zwischen 0 und 255 annehmen.

Bsp.:
```
10 CLS
20 MOUSE 1,30
30 END
```

Im Beispiel wird in Zeile 20 der Maus-Pfeil eingeschaltet und erhält die Farbe hellblau. Wie Du sicher bemerkt haben wirst, ist der Maus-Pfeil auch dann noch sichtbar, wenn das Programm längst beendet ist. Dies liegt darin begründet, das der Maus-Pfeil Bestandteil des Grafiktreibers in Bellatrix ist und vollkommen unabhängig vom Interpreter arbeitet. Du solltest also im Programm bei Beendigung der Mausbedienung auch den Maus-Pfeil wieder abschalten (und zwar mit MOUSE 0,0 wobei der Farbwert hier keine Rolle spielt aber als erforderlicher Parameter angegeben werden muss).

HINWEIS:
Der Maus-Pfeil arbeitet unabhängig vom Interpreter und muss bei Beendigung des Programms wieder mit MOUSE 0,0 ausgeschaltet werden.

Mausbereich definieren

Syntax: MBOUND *x,y,xx,yy*

In vielen Fällen ist es wünschenswert den Bereich, indem sich der Maus-Pfeil bewegen darf einzugrenzen. Hierfür ist dieser Befehl gedacht. Mit ihm ist es möglich, den Maus-Bereich Pixelgenau einzugrenzen.
Das heisst, das als Koordinatenangaben nicht Zeilen und Spaltenzahl angegeben werden, sondern die tatsächliche physikalische Auflösung des TRIOS-BASIC-Bildschirms.

Bsp.: MBOUND 0,0,639,479

… legt den gesamten Bildschirmbereich als aktive Fläche fest, wohingegen der Befehl....

Bsp.:MBOUND 0,470, 639,479

… nur die unterste Zeile des Bildschirms als aktive Fläche zulässt (z.Bsp.für Spiele)

HINWEIS:
Dieser Befehl wirkt nur, wenn zuvor der Maus-Pfeil aktiviert wurde.

Mauskoordinaten abfragen

Syntax: MX, MY, MZ

Dies sind die drei Befehle, um die Position des Maus-Pfeils und des Scrollrades (wenn vorhanden) als Funktionswerte zurückgibt.

Bsp.:
```
10 CROFF:CLS
20 MOUSE 1,1
30 x=MX:y=MY:z=MZ
40 POS 0,0:PRINT x,y,z
50 GOTO 30
```

MX und MY liefern Rückgabewerte, die der Bildschirmposition in Form von Zeilen und Spaltennummern (x=0-39, y=0-29) entsprechen. MZ stellt die Scrollradposition dar und kann auch negative Werte annehmen.

Maustasten abfragen

Syntax: MB *taste*

Kommen wir nun zum wichtigsten Maus-Befehl, der Abfrage der Maus-Tasten. Mit dem Befehl MB und dem Parameter *taste* wird erst die eigentliche Interaktion per Maus-Bedienung möglich. *Taste* stellt dabei einen Wert zwischen 0 und 4 dar und hat folgende Bedeutung

0=linke Maustaste
1=rechte Maustaste
2=mittlere oder Schroll-Rad-Taste
3=linke seitliche Taste
4=rechte seitliche Taste

Der Rückgabewert bei gedrückter Taste ist 255, wird keine Taste gedrückt wird 0 zurückgegeben. Bei der Verwendung von Buttons im Programm (siehe Abschnitt Buttons) erfolgt die Rückgabe der entsprechenden Nummer des Buttons, allerdings nur bei anklicken des Buttons mit der linken Maus-Taste.

Bsp.:

```
10 CROFF:CLS:MOUSE 1,1
20 POS 10,10:PRINT "l","m","r",
30 POS 10,4:PRINT "l=linke Maustaste"
40 POS 10,6:PRINT "r=rechte Maustaste"
50 POS 10,8:PRINT "m=mittlere Maustaste"
60 a=MB 0:b=MB 1:c=MB 2
70 POS 10,14:PRINT a,c,b,
80 IF a=255 THEN COL 0,30,40
90 IF b=255 THEN COL 0,40,50
100 IF c=255 THEN COL 0,50,60
110 GOTO 60
```

Probiere das Beispiel aus und beobachte die Wirkung der Maus-Tasten-Abfrage.
Jeder Maus-Taste ist eine andere Hintergrundfarbe zugeordnet. Diese ändert sich, sobald eine der Tasten gedrückt wird.
Das Beispiel entspricht der Datei „MOUSBUTT.BAS“ auf der SD-Card.

Bildschirmbefehle / Funktionen

Einfache Bildschirmausgabe

Syntax: PRINT *Ausdruck*

Der PRINT-Befehl ist der wichtigste Befehl überhaupt, erlaubt er doch erst die Kommunikation zwischen Mensch und Computer.
Er ist im Kommandomodus genauso nutzbar, wie im Programmmodus. *Ausdruck* kann ein Funktionswert, ein berechneter Rückgabewert oder eine Zeichenkette (String) sein.
Im Gegensatz zu Funktionswerteausgaben müssen Zeichenketten in Anführungszeichen stehen.

Bsp.:
```
10 CLS
20 PRINT“Dies ist eine Zeichenkette“
30 PRINT 34.5+67.9
40 #a=“Dies ist ein String“
50 PRINT #a
```

Im Beispiel wird in Zeile 20 eine Zeichenkette und in Zeile 30 ein Funktionswert ausgegeben. In Zeile 50 wird ebenfalls eine Zeichenkette in Form einer Stringvariablen ausgegeben.

Es ist auch möglich mehrere Variablen oder numerische Werte getrennt durch Komma oder Semikolon darzustellen wobei das Komma den Interpreter veranlasst, den nächsten Wert eine Tabulatorposition weiter rechts auszugeben wohingegen ein Semikolon die Werte oder Zeichen ohne Freizeichen direkt hintereinander zu schreiben.

Bsp.:
```
PRINT „5+8=“;5+8,“Tolles Ergebnis“
```

Im Beispiel wird zuerst die Zeichenkette „5+8=“ dargestellt und durch das folgende Semikolon, das Ergebnis direkt darauffolgend gedruckt. Das nun folgende Komma sorgt dafür, das die nächste Zeichenkette („Tolles Ergebnis“) um eine Tabulatorposition (5 Zeichen) nach rechts versetzt ausgegeben wird.
Experimentiere ruhig etwas mit dem Print-Befehl herum und beobachte die Wirkung der einzelnen Darstellungsvarianten.

Bildschirm bzw. Bildbereich löschen

Syntax: CLS *<x,y,xx,yy>*

CLS (Clear Screen) ohne Parameter löscht den gesamten Bildschirm, des aktiven Fensters (Window) und setzt den Cursor (falls er nicht ausgeschalten wurde) an die linke obere Ecke.
Mit der Parameterangabe *x,y,xx,yy* ist es möglich nur bestimmte Bildschirmbereiche zu löschen.

Bsp.:

```
10 CLS
20 FOR I=1 TO 600
30 PRINT „A“;
40 NEXT I
50 CLS 6,6,30,20
```

Dieses kleine Beispiel füllt den Bildschirm mit dem Buchstaben A und löscht dann nur einen Teil des Bildschirms.

HINWEIS:
X darf Werte von 0 bis 39 und Y von 0 bis 29 annehmen.

Cursor zurücksetzen

Syntax: HOME

HOME hat eine ähnliche Wirkung wie CLS nur wird hier nur der Cursor an die linke obere Bildschirmecke gesetzt. Der Bildschirminhalt bleibt erhalten.

Cursor an eine bestimmte Bildschirmposition setzen

Syntax: POS *x,y*

POS setzt den Cursor an die Position, die mit den Parametern *x* und *y* angegeben wurden. X darf Werte von 0 bis 39 und y 0 bis 29 annehmen.

HINWEIS:
POS bezieht sich immer auf das aktuell mit WSET gesetzte Window (siehe WIN bzw. WSET).

Bildschirmausgabe mit Hilfe der TAB-Funktion

Syntax: TAB*(wert)*

Die Tab-Funktion kann benutzt werden um eine Struktur in der Ausgabe der Daten zu erhalten. Diese ist programmiertechnisch einfacher als die Ausgabepositionen per POS-Befehl zu erzeugen.
Dieser Befehl ist nur in Verbindung mit PRINT sinnvoll.

Bsp.:
```
10 CLS
20 a=12.8:b=17.3:c=22.9:d=55.2
30 PRINT TAB(4);a;TAB(10);b;TAB(20);c;TAB(28);d
```

Ausgabe:
```
   12.8  17.3       22.9       55.2
```

Der Unterschied zur Verwendung des Komma's als Tabulator ist die Möglichkeit, die Abstände zwischen den Werten flexibel zu gestalten.

Bildschirmfarben ändern

Syntax: COL *Vordergr , Hintergr , Cursor*

TRIOS-BASIC arbeitet mit einem Grafiktreiber, der 64 Farben unterstützt. Dies erlaubt eine sehr flexible farbliche Gestaltung des Bildschirms.
Vordergr – definiert die Vordergrund oder auch Schriftfarbe
Hintergr - setzt dementsprechend die Hintergrundfarbe
Cursor - bestimmt, in welcher Farbe der blinkende Cursor erscheint

Die einzelnen Farbwerte dürfen sich im Bereich von 0 bis 255 bewegen.
Hier eine kleine Farbtabelle als Anhaltspunkt für die wichtigsten Farben.

Farbtabelle:

0 - Schwarz	4 - Dunkelblau	9 - Blau	16-Dunkelgrün	20-Dunkel-Türkis
31-Hell-Blau	32-Grün	40-Blaugrün	48-Hell-Grün	60-Türkis
64-Dunkelrot	68-Dunkel-Lila	84-Dunkelgrau	128-Rot	136-Lila
168- Grau	192-Rosa	204-Hell-Lila	230-Orange	146-Dunkel-Orange
240-Gelb	252-Hell-Grau	255-Weiss		

Bildschirmpunkt setzen

Syntax: PLOT *Farbe , x , y*

Plot setzt einen Bildschirmpunkt mit der als Parameter übergebenen *Farbe(0-255)* an die Spaltenposition *x (0-39)* in Zeile *y(0-29)*.
Nun ja, Punkt ist vielleicht etwas untertrieben. Da es sich beim Grafiktreiber um einen Tile-basierten Treiber handelt, ist die kleinste darstellbare Grafikgrösse natürlich ein Tile und das besteht aus 16x16 Bildpunkten. Als Punkt, zugegeben etwas groß aber der Propeller hat bei einer Auflösung von 640x480 Bildpunkten einfach zuwenig Speicher, um jeden einzelnen Pixel adressieren zu können.

Bsp.:
```
10 CLS
20 FOR I=10 TO 20
30 FOR A=10 TO 20
40 PLOT 44,I,A
50 NEXT A
60 NEXT I
```

Das Beispiel plottet ein Quadrat auf den Bildschirm.

Bildschirmbereiche nach unten scrollen

Syntax: SCRDN *lines, farbe, x , y ,xx ,yy ,rate*

SCRDN scrollt den Bildschirm um die Anzahl der mit *lines* angegebenen Zeilen von Position x,y bis xx,yy nach unten.Die gescrollten Zeilen werden in der übergebenen Farbe dargestellt.

Bsp.:
```
10 CLS
20 FOR I=1 TO 10
30 a=RND(255)
40 SCRDN 1,a,2,11,35,24,0
50 NEXT I
```

Im Beispiel wird der Bereich von Spalte 2 Zeile 11 bis Spalte 35 Zeile 24 in einer durch Zufallsfunktion ermittelten Farbe nach unten gescrollt.
Der Wert *rate* bestimmt die Geschwindigkeit der Scrollfunktion und kann Werte zwischen 0 und 255 haben, wobei 0 die schnellste und 255 die langsamste Scrollrate ist.

Bildschirmbereiche nach oben scrollen

Syntax: SCRUP *lines, farbe, x , y ,xx ,yy ,rate*

SCRUP macht genau das Gegenteil wie SCRDN und scrollt den Bildschirm nach oben. Die Parameter entsprechen denen der SCRDN-Funktion.
Interessant wird das ganze, wenn man beide Scroll-Funktionen kombiniert, wie im folgenden Beispiel.

Bsp.:

```
10 CLS
20 FOR I=1 TO 10
30 a=RND(255)
40 SCRDN 1,a,2,11,35,24,1
50 SCRUP 1,a,2,2,35,11,1
60 NEXT I
```

Wir haben das SCRDN-Beispiel nun erweitert und den SCRUP-Befehl eingefügt. Das Ergebnis ist ein nach oben und unten scrollender Bildschirm.

Bildschirmcursor ausschalten / einschalten

Syntax: CROFF

CROFF schaltet den Cursor unsichtbar. Dies ist besonders bei Textausgaben oder bei der Darstellung von Tilegrafiken hilfreich. In solchen Fällen ist ein irgendwo blinkender Cursor eher lästig.

Syntax: CRON

Mit CRON kann der Cursor bei Bedarf wieder sichtbar gemacht werden. Dies ist besonders bei der Zeileneditierung nützlich.

Ein Titel-Window erstellen

Syntax: TWIN *Nr, Vordergr, Hintergr, Cursor, x, y, xx, yy, Mode,"Titeltext"*

Kommen wir zu den Fensterfunktionen, die das TRIOS-BASIC erst so richtig interessant machen. Durch den neuen Grafiktreiber wird die ganze Sache auch etwas bunter im Gegensatz zur Vorgängerversion.
Folgende Parameter müssen übergeben werden.

Nr – Fensternummer (gültig sind Werte zwischen 1 und 7)
Vordergr – Schriftfarbe des Fensters (gilt nur für dieses Fenster)
Hintergr – Hintergrundfarbe des Fensters (gilt nur für dieses Fenster)
Cursor - Cursorfarbe (ebenfalls nur gültig für dieses Fenster)
x,y,xx,yy – Position des Fensters von x,y bis xx,yy
Mode - Verwendung mit Tile-Font (0) oder Systemfont(1)
Titeltext – Text der in der Titelzeile stehen soll

Einmal definierte Fenster sind solange in ihren Dimensionen gültig (auch wenn es nicht mehr sichtbar ist) bis es neu definiert wird oder Du ein neues Programm lädst.
Ein Umschalten der Fenster erfolgt mit dem Befehl WSET Nr (siehe dort).
Das Hauptfenster hat immer die Nummer 0 und kann nicht geändert werden. Selbst definieren kannst Du die Fenster 1 bis 7.
Mit dem Parameter *Mode* wird bestimmt, mit welchem Font der Titeltext dargestellt wird. 1 entspricht dem Systemfont (dies ist die Standard Schriftart). Mit 0 wird für den Titeltext ein mit TLOAD geladener und mit STILE aktivierter Tile-Font verwendet, was der optischen Gestaltung noch mehr Möglichkeiten bietet.

Bsp.:
```
10 CLS
20 TLOAD 1,"font1.dat",16,11
30 STILE 1
40 TWIN 1,50,44,0,0,5,14,29,1,"Window1"
50 TWIN 1,90,30,0,15,5,39,29,0,"Window2"
```

Fenster 1 erscheint links auf dem Bildschirm mit dem Systemfont und rechts erscheint Fenster 2 mit einem Tile-Font.

HINWEIS:
Der Text der Titelzeile wird automatisch vom Interpreter auf die Fensterbreite gekürzt, sollte er zu lang sein. Beachte also, das Deine Titeltexte auch in die Fenster passen.

Window erstellen

Syntax: WIN *Nr, Vordergr, Hintergr, Cursor, x, y, xx, yy,mode*

WIN erstellt ein Window (Fenster) mit der Nummer *Nr* und den Farben *Vordergr,Hintergr,Cursor* an Position *x,y* bis *xx,yy*

Nr – Fensternummer (gültig sind Werte zwischen 1 und 7)
Vordergr – Schriftfarbe des Fensters (gilt nur für dieses Fenster 0-255)
Hintergr – Hintergrundfarbe des Fensters (gilt nur für dieses Fenster 0-255)
Cursor - Cursorfarbe (ebenfalls nur gültig für dieses Fenster 0-255)
x,y,xx,yy – Position des Fensters von x,y bis xx,yy (x,xx 0-39; y,yy 0-29)
mode - Rahmen 0=ohne Rahmen 1=mit Rahmen

Einmal definierte Fenster sind solange in ihren Dimensionen gültig (auch wenn es nicht mehr sichtbar ist) bis es neu definiert wird oder Du ein neues Programm lädst.
Ein Umschalten der Fenster erfolgt mit dem Befehl WSET Nr (siehe dort).
Das Hauptfenster hat immer die Nummer 0 und kann nicht geändert werden. Selbst definieren kannst Du die Fenster 1 bis 7.

Bsp.:
```
10 CLS
20 WIN 1,0,30,0,5,5,30,25,1
```

Erstellt ein Fenster mit hellblauem Hintergrund an Position 5,5 bis 30,25 mit einem Rahmen.

Titel-Bar erstellen

Syntax: TBAR *orand,Hinterg,urand,Vordergr,modus,,,Text“*

TBAR erstellt eine Titelleiste am oberen Bildschirmrand mit folgenden Parametern:
orand - obere Randfarbe (0-255)
Hintergr - Hintergrundfarbe (0-255)
Urand - untere Randfarbe (0-255)
Vordergr – Schriftfarbe (0-255)
Modus - Schriftart (1-Systemfont , 0- Tilefont)

Bsp.:
```
10 CLS
20 TLOAD 1,“Font5.dat“,16,11
```

30 STILE 1
40 TBAR 0,20,40,0,1,“TITELTEXT IN EINER TBAR“

Dieses Beispiel erstellt eine TBAR mit der Tile-Font-Datei „Font5.dat“

Message-Box erstellen

Syntax: MBOX *orand,Hinterg,urand,x,y,modus,,,Titel“,“Nachricht“*

Mit MBOX ist es möglich kleine Hinweisboxen zu generieren.

Parameter:
orand - obere Randfarbe (0-255)
Hintergr - Hintergrundfarbe (0-255)
urand - untere Randfarbe (0-255)
x,y - Position (x=0-39 ; y=0-29)
modus - Schriftart (0=Tile-Font, 1=Systemfont)
Titel - Titeltext der MBOX
Nachricht- Nachrichtentext

HINWEIS:
Es kann nur ein einzeiliger Nachrichtentext ausgegeben werden.
Achte darauf, das der Titeltext nicht länger als die Nachricht ist, sonst wird der Text abgeschnitten.

Bsp.:
MBOX 0,30,40,10,10,1,“Hilfe“,“Hilfe ich hab eine MBOX“

Frame(Rahmen) erstellen

Syntax: FRAME *orand,Rahmen,urand,x,y,xx,yy*

FRAME generiert einen 3D-Rahmen mit den folgenden Parametern:

orand - obere Randfarbe (0-255)
Rahmen – eigentliche Rahmenfarbe (0-255)
urand - untere Randfarbe (0-255)
x,y,xx,yy- Position und Größe von x,y bis xx,yy (x von 0-39, y von 0-29)

Bsp.:
```
FRAME 30,0,40,10,10,35,25
```

Zeichnet einen schwarzen Rahmen mit einer blauen oberen Randfarbe und einer türkisen unteren Randfarbe an die Position x=10 y=10 bis x=35 y=25

Aktives Window(Fenster) wechseln

Syntax: WSET *Wnr*

Um mit mehreren Fenstern zu arbeiten ist es nötig, zwischen diesen wechseln zu können. Dafür gibt es den Befehl WSET. Unter Angabe der Fensternummer kannst Du nun einfach zwischen den Fenstern umschalten.

Bsp.:
```
10 CLS:b=1
20 WIN 1,0,30,0,0,0,39,10,1
30 WIN 2,0,40,0,0,11,39,19,1
40 WIN 3,0,50,0,0,20,39,29,1
50 WSET b
60 a=INKEY:IF a=0 THEN GOTO 50
70 IF a=9 THEN b=b+1
80 IF b=4 THEN b=1
90 WSET b
100 GOTO 50
```

In diesem Beispiel werden 3 Windows untereinander dargestellt. Das aktive Window, ist das oberste. Durch Drücken der Tabulatortaste (Die Tab-Taste hat den Wert 9 und befindet sich links neben dem Q), wird in das darunter liegende Fenster gewechselt. Ein weiterer Druck auf die Tab-Taste macht das unterste Fenster zum aktiven Window. Wird nochmal die Taste TAB betätigt, landet der Cursor wieder im oberen Fenster.

Eine Textzeile horizontal scrollen

Syntax: SCROLL *„Scrolltext“,rate,Vordergr,Hintergr,x,y,xx*

Mit dem Befehl SCROLL ist es möglich eine Textzeile auf dem Bildschirm von rechts nach links scrollen zu lassen. Folgende Parameter müssen dabei übergeben werden:

Scrolltext - Der Text der gescrollt werden soll
rate - Geschwindigkeit, mit der gescrollt wird (0-255)
Vordergr - Textfarbe des Scrolltextes (0-255)
Hintergr - Hintergrundfarbe des Scrolltextes (0-255)
x,y - Position bis zu der gescrollt werden soll (x von 0-39, y von 0-29)
xx - Startpunkt des Scrollbereiches (0-39)

Die Werte der Scrollrate dürfen im Bereich von 0-255 liegen wobei 0 die schnellste und 255 die langsamste Scrollrate darstellt.
Praktikable Werte sind im Bereich von 3 bis ca.15.

Bsp.:
SCROLL „Testtext“,5,0,30,0,10,39

HINWEIS:
Während die Scrollfunktion aktiv ist, wird der Interpreter blockiert. Wähle also die Scrollrate nicht zu langsam um die Programmabarbeitung nicht zu lange zu unterbrechen.

Einen Befehlsknopf erstellen

Syntax: BUTTON *ID,Farbe,x,y,mode,“Text“*

Befehlsknöpfe machen die Bedienung mit der Maus erst wirklich sinnvoll, deshalb wurde der Befehl BUTTON ins TRIOS-BASIC integriert.
Durch anklicken mit der Maus ist eine Verzweigung in diverse Unterprogrammteile möglich und schafft so die Interaktion mit dem Hive ähnlich wie am PC zu gestalten.
Folgende Parameter werden dazu benötigt.

ID - Nummer des Buttons (Diese Nummer wird beim Anklicken mit der Maus zurückgegeben und identifiziert den gewählten Button.
Farbe -bestimmt die Farbe des Buttons
x,y -Position des Buttons auf dem Bildschirm(x=0-39, y=0-29)
mode – legt fest, ob der Text mit Systemfont (1) oder mit Tile-Font (0) dargestellt wird.

Bsp.:
```
10 CLS:CROFF:MOUSE 1,1
20 BUTTON 1,30,10,10,0,“Testknopf“
30 a=MB 0
40 IF a=1 THEN PRINT a;
50 GOTO 30
```

In diesem Beispiel wird an Position Zeile 10 Spalte 10 ein blauer Button mit dem Text:Testknopf mit Tile-Font-Schriftart erstellt. Wenn Du nun mit dem Mauspfeil auf den Button klickst (linke Maustaste) wird die Nummer des Buttons ausgegeben (in diesem Fall eine 1).

HINWEIS:
Die Größe des Buttons richtet sich nach der Textlänge, also immer darauf achten, was als Text im Button stehen soll. Stringvariablen können ebenfalls als Button-Beschriftung dienen.

Einen Befehlsknopf löschen

Syntax: DBUTT *ID*

DBUTT löscht einen zuvor erstellten Button mit der Nummer *ID* vom Bildschirm.

Bsp.: DBUTT 1

Löscht den im Beispiel BUTTON erzeugten Befehlsknopf vom Bildschirm.

Eine Box zeichnen

Syntax: BOX *Farbe,x,y,xx,yy,Schatten*

BOX erstellt ein Quadrat mit der *Farbe* an Position *x,y* bis *xx,yy* , mit (1) oder ohne *Schatten* (0) auf dem Bildschirm.
Farbe darf Werte im Bereich von 0-255 annehmen. Die Werte für *x* und *xx* dürfen im Bereich 0-39 und die *y* und *yy*-Werte im Bereich 0-29 liegen. Für *Schatten* sind nur Werte von 0(ohne)-1(mit Schatten) erlaubt.

Bsp.:BOX 50,3,5,20,15,1

Erstellt eine hellgrüne Box an Position 3,5 bis 20,15 mit Schatten.

Cursorposition abfragen

Syntax: GETX

Gibt als Funktionswert die Spaltenposition des Cursors zurück.

Syntax: GETY

Gibt als Funktionswert die Zeilenposition des Cursors zurück.

Bsp.:

```
10 CROFF:CLS:POS 0,2
20 a=INKEY:IF a=0 THEN GOTO 20
30 PRINT CHR$(a);
40 x=GETX
50 y=GETY
60 POS 0,30:PRINT x,y
70 POS x,y
100 GOTO 20
```

Im Beispiel wird jede Tasteneingabe auf dem Bildschirm ausgegeben. In Zeile 0 wird die jeweilige Cursorposition angezeigt.

Stringfunktionen

Die Funktionen left ,mid, right

Syntax: STR$ <*Option*>

In vielen Basic-Varianten hat dieser Befehl eine ganz andere Aufgabe als im TRIOS-BASIC (wandelt normalerweise eine Zahl in einen String um, das geht bei TRIOS-BASIC einfacher).

Hier ist er der Marker für die Standard-Stringfunktionen left, mid und right.

Syntax: STR$ *L(String,Anzahl)*

STR$ L extrahiert einen Teilstring von links aus *String* mit der Länge *Anzahl*

Bsp.:

#a="Dies ist der erste String"

PRINT STR$ L(#a,8)

Ausgabe:

Dies ist

Syntax: STR$ *M(String,Start,Anzahl)*

Mit diesem Befehl wird ein Teilstring ab der Position *Start* mit der Länge *Anzahl* aus *String* extrahiert.

Bsp.:

#a="Dies ist der erste String"

PRINT STR$ M(#a,10,16)

Ausgabe:

der erste String

Syntax: STR$ *R(String,Anzahl)*

Nun wird ein Teilstring von rechts mit der Länge *Anzahl* aus *String* extrahiert.

#a="Dies ist der erste String"
PRINT STR$ R(#a,6)
Ausgabe:
String

Strings vergleichen

Syntax:COMP$ *(String1,String2)*

Um zwei Strings miteinander zu vergleichen, ist dieser Befehl gedacht.
Ist String1 und String2 identisch, wird -1 als Funktionswert zurückgegeben.
Sind beide Strings unterschiedlich erfolgt die Rückgabe von Null.

Bsp.:
#a="Dieser String ist kurz"
#b="dieser string ist kurz"
PRINT COMP$(#a,#b)

Ausgabe
0
...Da auch Groß- und Kleinschreibung unterschieden wird, ist die Ausgabe entsprechend zwei ungleicher Strings Null
Probiere folgendes:
#b=#a
PRINT COMP$(#a,#b)
Und die Ausgabe wird -1 sein.

Die Länge eines Strings ermitteln

Syntax: LEN *(String)*

Mit LEN kann die Länge einer Stringvariablen ermittelt werden.

Bsp.:

```
#a="alle Drohnen aufgepasst!"
PRINT LEN(#a)
```

Ausgabe:

```
24
```

Wandlung ASCII-Code nach Zeichen

Syntax: CHR$ *(ASCII-Code)*

CHR$ wandelt einen als ASCII-Code übergebenen Wert in ein entsprechendes Zeichen um.

Bsp.:

```
#a=CHR$(66)
PRINT #a
```

oder

```
PRINT CHR$(66)
```

Ausgabe

```
B
```

Wandlung Zeichen nach ASCII-Code

Syntax: ASC *(Zeichen)*
ASC wandelt ein Zeichen in den zugehörigen ASCII-Code.

Bsp.:
PRINT ASC(„A“)
oder
#a=“A“
PRINT ASC(#a)
oder
PRINT ASC(„Alles Gut“)

Ausgabe
65

Umwandeln eines Strings in eine Zahl

Syntax: VAL *(String)*

VAL wandelt eine als String übergebene Zahl in einen numerischen Wert um.

Bsp.:
#a=“34.98“
b=VAL(#a)
PRINT b
oder
b=VAL(„34.98“)
PRINT b

Ausgabe
34.98

HINWEIS:

Natürlich kann auch eine numerische Variable in einen String umgewandelt werden dazu wird einfach dem String die Zahl zugewiesen.

Bsp.: a=123.45

```
#a=a
PRINT #a
```

Ausgabe

123.45

Wiederholung einer Zeichenkette

Syntax: STRING$ *(Wiederholungen,String)*

STRING$ gibt die Zeichenkette *String* mit der Anzahl *Wiederholungen* auf dem Bildschirm aus.

Bsp.:

```
#a="Hive-Computer "
PRINT STRING$(10,#a)
```

oder

```
PRINT STRING$(10,"Hive-Computer ")
```

Ausgabe:

Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer Hive-Computer

Die Zeichenkette „Hive-Computer „ wird 10 mal auf dem Bildschirm ausgegeben.

Ganz anders verhält es sich bei folgendem Beispiel.

Bsp.:

```
#a="Hive-Computer "
#b=STRING$(10,#a)
PRINT #b
```

Ausgabe:

```
Hive-Computer Hive-Computer
```

Warum wird hier nur zweimal die die Zeichenkette ausgegeben?

Das liegt an der Längenbegrenzung für die Strings auf 33 Zeichen. Der Interpreter hat festgestellt, das die Zeichenkette nur zweimal in die Variable #b passt und hat den Rest ignoriert, da er nicht mehr hineinpasst. Dies ist wichtig zu wissen, wenn man mit Strings arbeitet. **Die maximale Länge eines Strings beträgt 33 Zeichen.**

Tile-Grafik-Befehle

Wie schon öfter erwähnt, unterstützt TRIOS-BASIC die Verwendung von Tile-Grafiken. Mit ihnen sind die gestalterischen Möglichkeiten um ein Vielfaches höher als in der Vorgängerversion. Tile-Grafiken können für die verschiedensten Aufgaben benutzt werden.

-als reine Grafikdateien (um z.Bsp. Bilder darzustellen)

- als Fontdateien (verwende für Deine Programme doch einfach eine andere Schriftart)
- als Spielegrafik (mit Tiles ist es möglich Maps für Spiele zu erstellen)
- uvm.

Tile-Grafik von SD-Card laden

Syntax: TLOAD *Nr,"Tiledatei",x-größe,y-größe*

Um mit eigenen Tile-Dateien zu arbeiten, müssen sie zuerst in den E-RAM geladen werden. Das macht TLOAD. Da es möglich ist, bis zu 14 Tilegrafiken in den E-RAM zu laden, ist es notwendig die einzelnen Dateien zu unterscheiden. Dies geschied mit

der Angabe von *Nr*. Danach wird der Name der Tile-Datei in Anführungszeichen übergeben. Damit der Interpreter die Datei auch findet, muss sie zwingend im Unterverzeichnis TILE vorhanden sein, welches sich im Ordner BASIC befindet. Abschließend werden noch die Tile-Datei-Dimensionen übergeben. Siehe dazu folgendes Beispiel.

Bsp.:

TLOAD 1,“Font1.dat“,16,11

Im Beispiel wird die sich auf der SD-Card befindliche *Tile-Datei* „Font1.dat“ mit der *x-Größe* 16 und der *y-Größe* 11 auf Position *Nr* 1 in den E-RAM geladen.

x- und y-Größe entsprechen der Anzahl der Tiles in x und y -Richtung.

Da alle mit „Font“ bezeichneten Dateien eine Größe von 256*176 Pixeln haben, entsprechen ihre Tile-Dimensionen immer 16 in x und 11 in y-Richtung

Um zu erfahren wie die Dimensionen der entsprechenden Grafiken ist, muss folgende Rechnung angestellt werden.

-jedes Tile besteht aus 16*16 Pixeln (die kleinste Grafikeinheit)

-Auflösung der Grafik in x-Richtung z.Bsp.:256

-macht in x-Richtung 256/16 Pixel je Tile =16 Tiles

-Auflösung in y-Richtung zBsp:176

-macht in y-Richtung 176/16 Pixel je Tile=11 Tiles

256*176 Pixel ist zugleich auch die maximal zulässige Tile-Datei-Größe.

Zwischengrößen sind möglich dürfen diese Grenze aber nicht überschreiten.

Bsp.: Tile-Grafik-Größe=128*128 -> erlaubt (128*128=16384<256*176=45056)

Tile-Grafik-Größe=256*256 ->nicht erlaubt weil 256*256=65536 > 45056)

HINWEIS:

Tile-Grafiken dürfen maximal 256*176=45056 Pixel Groß sein. Jede andere Größe innerhalb dieser Grenze ist erlaubt.

Tile-Dateien müssen immer im Unterverzeichnis „TILE“ im Ordner BASIC abgelegt werden.

Tile-Grafiken erstellt man am einfachsten mit einem Grafikprogramm, mit dem man Pixelgenau arbeiten kann (z.Bsp.Ulead IPhoto Impact o.ä.).Danach müssen die Grafiken für den Propeller konvertiert werden. Am besten eignet sich das Programm Prop2Bitmap.exe. Dieses Programm kann unter http://www.rayslogic.com/propeller/

Programming/2BitBitmap.htm heruntergeladen werden.

In diesem Programm werden die Tile-Datei-Größen auch angezeigt, ohne das man sie extra ausrechnen muss.

Tile-Datei zur Verwendung auswählen

Syntax: STILE *Nr*

Damit man nun mit der geladenen Datei arbeiten kann, muss sie zunächst aktiviert werden. Dies geschieht mit dem Befehl STILE mit der als *Nr* übergebenen Tile-Datei-Nr. Der Interpreter lädt nun die Tile-Daten aus dem E-RAM in einen Puffer in Bellatrix. Jetzt sind die Weichen gestellt und wir sind bereit für erste Experimente.

Tile-Datei als Grafik auf dem Bildschirm ausgeben

Syntax: TPIC *Farbe1,Farbe2,Farbe3,x,y*

Um zu sehen, wie unsere Tile-Datei aussieht, lassen wir sie jetzt auf dem Bildschirm erscheinen. Dazu wird der Befehl TPIC benutzt.

Folgende Parameter müssen übergeben werden.

Farbe1 - erste Tile-Farbe

Farbe2 - zweite Tile-Farbe

Farbe3 - dritte Tile-Farbe

x,y - Position auf dem Bildschirm

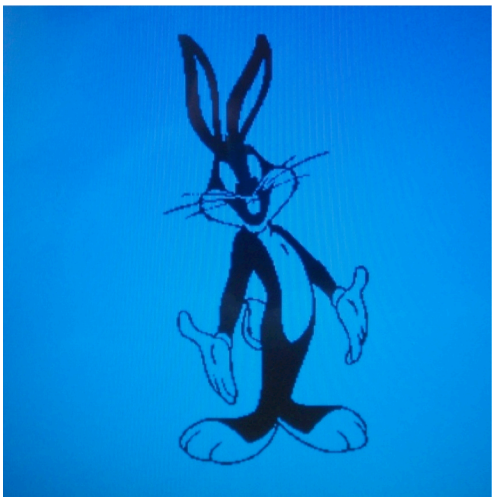
Die drei Tilefarben, sind die Farben, die bei der Erstellung der Dat-Datei mit Prop2Bitmap.exe gewählt wurden (meistens ist Farbe1 die Hintergrundfarbe und Farbe2 und 3 Vordergrundfarben).

Bsp.:

```
10 COL 0,30,0:CLS
20 TLOAD 1,“Bunny9.dat“,8,16
```

```
30 STILE 1
40 TPIC 30,0,0,10,12
```

Ausgabe



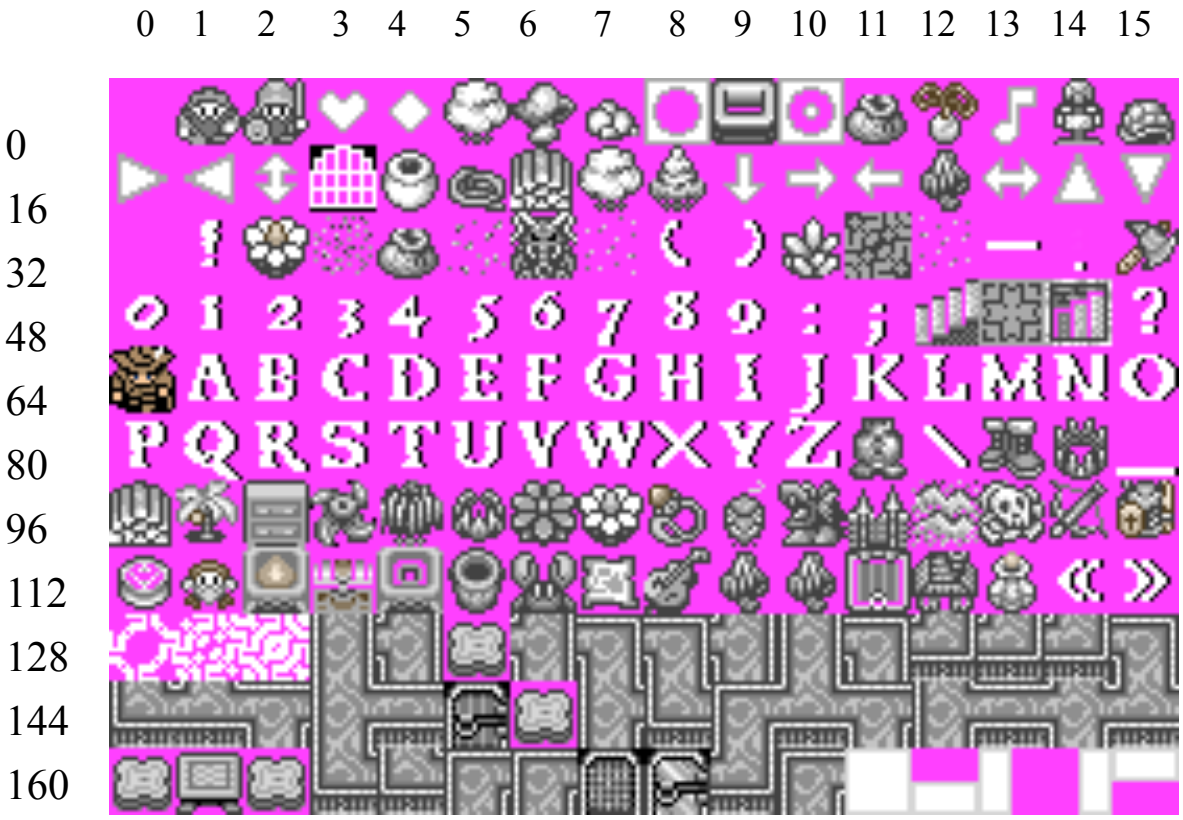
Einzelnes Tile auf dem Bildschirm ausgeben

Syntax: TILE *Tilenr, Farbe1, Farbe2, Farbe3, x, y*

Neben der Verwendung der Tile-Datei als reine Grafikdatei, ist es oft auch wünschenswert Tile-Dateien als Gestaltungsmittel einzusetzen um beispielsweise Menüs oder Startbildschirme zu erstellen. Dazu sind unter anderem, die als Font bezeichneten Tile-Dateien im Verzeichnis TILE gedacht. In ihnen sind neben einem Schriftsatz, etliche Grafiktiles enthalten, die für diese Zwecke brauchbar sind.

Um auf einzelne TILE-Stücke innerhalb dieser Dateien zugreifen zu können, ist der TILE-Befehl gedacht.

Im Folgendem ist die Datei Font1.dat dargestellt, um die Wirkungsweise dieses Befehls zu erläutern.



Jedes Tile stellt einen Block der Größe 16*16 Pixel dar. An den Randzahlen, ist der Prinzipielle Aufbau von Tile Dateien erkennbar. Nach jeweils 16 Pixeln beginnt ein anderes Grafiksymbol. Insgesamt besteht die Datei aus 176 Tile-Blöcken. Jeder einzelne Block kann nun mit dem Befehl TILE zur Ausgabe angewählt werden.

Im folgenden kleinen Beispiel geben wir die oberste Zeile der Tile-Datei aus.

Bsp.:

```
10 CLS
20 TLOAD 1,“Font1.dat“,16,11
30 STILE 1
40 FOR I=0 TO 15
50 TILE i,255,0,$80,10+i,10
60 NEXT i
```

Im nächsten Beispiel geben wir einzelne Tiles auf dem Bildschirm aus.

Bsp.:

```
10 CLS
20 TLOAD 1,“Font1.dat“,16,11
30 STILE 1
40 TILE 2,255,0,240,8,5
50 TILE 20,255,0,$80,10,10
60 TILE 161,$e6,0,255,12,6
```

Wie im zweiten Beispiel zu sehen, kann jedes TILE in der Farbgebung individuell gestaltet werden. Voraussetzung dafür ist die im Programm Prop2Bitmap definierte Farbanzahl (max 3).

Text mit Tile-Font anzeigen

Syntax: FONT *„Zeichenkette“, Farbe1, Farbe2, Farbe3, x, y*

Eine der grössten Vorteile von Tile-Dateien, ist die Möglichkeit verschiedenste Schriftarten in seinen Anwendungen zu verwenden. Je nach Art der Schrift, ändert sich dadurch das Aussehen und der Charme einer Anwendung grundlegend. Die individuelle Farbgebung der Schrift tut dazu ihr übriges. Um diese Funktion nutzen zu können, muss zuerst eine Tile-Datei mit Font-Tiles geladen (z.Bsp.TLOAD 1,“Font5.dat“,16,11) und aktiviert (STILE 1) werden.

Bsp.:

```
10 CLS
20 TLOAD 1,“Font5.dat“,16,11
30 STILE 1
40 #a= “DAS IST EIN TEXT MIT TILE-FONT“
50 FONT #a,$E6,0,255,10,10
```

oder

```
50 FONT“DAS IST EIN TEXT MIT TILE-FONT“,$E6,0,255,10,10
```

Wie im Beispiel zu sehen, kann *Zeichenkette* ein in Anführungszeichen stehender Text oder ein zuvor definierter String sein. Danach folgen die 3 Farbwerte, um die entsprechende Zeichenkette farblich zu gestalten. *X* und *y* geben wieder die Position auf dem Bildschirm an.

In diesem Fall wird der Text auf orangen Hintergrund in schwarz (diese Tile-Datei ermöglicht nur zwei Farben, die dritte wird ignoriert, muss aber als Parameter mit angegeben werden) ausgegeben.

HINWEIS:

Der Text wurde bewusst nur mit Großbuchstaben geschrieben, da die auf der SD-Card vorhandenen Dateien nur große Buchstaben enthalten um möglichst viele Grafiktiles mit unter zu bringen. Jeder kann sich aber seinen eigenen Schriftsatz bzw. seine eigene Tile-Datei erstellen und einbinden.

Wichtig dabei ist nur, das die Buchstaben auch an der für den ASCII-Code entsprechenden Stelle vorhanden sind, sonst funktioniert der Font-Befehl natürlich nicht korrekt (ein großes A hat den ASCII-Code 65 und muss dementsprechend an Position 65 in der Tile Datei stehen.

Selbst erstellte Tile-Dateien müssen zwingend ins Unterverzeichnis TILE im Ordner BASIC kopiert werden, um vom Interpreter auch gefunden zu werden.

MAP-Dateien erstellen, speichern, laden und anzeigen

Syntax: MAP <*Option*>

Der MAP-Befehl ist dafür gedacht, mit Tile-Grafik selbst gestaltete Bildschirmseiten zu speichern, zu laden und anzuzeigen. Momentan existiert noch kein Map-Designer (steht aber auf meinem Plan), weswegen sich die Erstellung eines mit Tile-Grafik gefüllten Bildschirms zur Zeit noch etwas aufwendiger gestaltet.

Die Funktionsweise des MAP-Befehls wird anhand eines Programmausschnittes, welches verwandt wurde, um den Bildschirm des DMP-Players zu erstellen, erläutert.

Bsp.:

```
10 COL 0,$e6,0
20 CROFF
```

```
30 CLS
40 TLOAD 1,"font1.dat",16,16
50 STILE 1
100 MAP w,0,0,39,29
120 TILE 166,255,$80,0,0,0
130 FOR i=1 TO 38
140 TILE 148,255,$80,0,i,0
150 NEXT i
160 TILE 143,255,$80,0,39,0
170 TILE 131,255,$80,0,0,1
180 TILE 131,255,$80,0,39,1
190 TILE 147,$56,$80,0,0,2
200 FOR i=1 TO 38
210 TILE 148,$56,$80,0,i,2
220 NEXT i
230 TILE 132,$56,$80,0,39,2
240 TILE 147,$56,$80,0,0,3
```

In der farbig markierten Zeile 100 wird dem Interpreter mitgeteilt, alle folgenden Tile-Befehle in den E-RAM zu schreiben.

Map-schreiben

Syntax: MAP W,x,y,xx,yy

MAP w fordert den Interpreter auf, alle nun folgenden Tile-Befehle in den E-RAM zu schreiben. Als Parameter werden die Bildschirmdimensionen x,y bis xx,yy übergeben. Jeder Tile-Befehl mit Tile-Nr, Farbwerte1-3 und Position x,y wird gespeichert. Dabei kann jedes Tile andere Farbwerte, Tile-Nr und x,y Positionen haben, es wird jedes einzelne Tile ins RAM geschrieben.

Bist Du mit dem Ergebnis zufrieden, müssen wir die Daten nur noch aus dem RAM auf SD-Card schreiben.

MAP-auf SD-Card sichern

Syntax: MAP S,“Dateiname“

Jetzt sichern wir die Daten auf SD-Card. Vergib eindeutige Dateinamen, um die MAP-Datei von anderen (zBsp. Basic-Dateien) unterscheiden zu können beispielsweise Mapxxx.dat oder Name.map.

MAP-von SD-Card laden

Syntax: MAP L,“Dateiname“

Mit diesem Befehl laden wir eine zuvor gesicherte Map-Datei wieder in den E-RAM.

MAP-auf dem Bildschirm anzeigen

Syntax: MAP D

Um die Map-Daten im E-RAM auf dem Bildschirm anzuzeigen, wird einfach MAP D eingegeben und das Werk sollte wieder zu sehen sein.

HINWEIS:

-Um Map-Daten zu erstellen, MAP W verwenden

-Danach mit MAP S die Daten sichern

-Laden der Daten erfolgt mit MAP L

-zu guter letzt MAP D zum darstellen der MAP-Datei

MAP D benötigt keine weiteren Parameter, da alle nötigen Informationen in der MAP-Datei enthalten sind (Dimensionen, x und y Positionen sowie Farbwerte).

Befinden sich keine Daten im RAM, hat der Befehl MAP D keine Wirkung.

Mathematische Funktionen

grundlegende Rechenoperationen

In dieser Version von TRIOS-BASIC wurde der Umfang der mathematischen Funktionen drastisch erweitert.

Im Gegensatz zu allen mir bekannten, auf Femto-Basic basierenden Basic-Varianten, beherrscht TRIOS-BASIC das Rechnen mit Gleitkomma-Zahlen und unterstützt wissenschaftliche Rechenoperationen. Zu den grundlegenden Rechenoperationen gehören die vier Grundrechenarten + - * /.

Bsp.: PRINT 3*4+2 -> Ausgabe 14

Bsp.: PRINT 5.3*7.8+3.2 -> Ausgabe 44.57

Wie in der Schule gelernt kommt Punktrechnung vor Strichrechnung d.h. es werden immer erst die Operationen Multiplikation und Division ausgeführt.

Auch im folgendem Beispiel ist das so.

Bsp.:PRINT 4+2*3 ->Ausgabe 10

Bsp.:PRINT 7.8+3.2*5.3 ->Ausgabe 24.76

Sollen Addition oder Subtraktion zuerst ausgeführt werden, so sind die entsprechenden Operationen in Klammern zu setzen.

Bsp.:PRINT 3*(4+2) ->Ausgabe 18

Bsp.:PRINT 5.3*(7.8+3.2) -> Ausgabe 58.3

Probiere auch andere Rechenoperationen aus und überprüfe das Ergebnis.

Als erweiterte Funktionen stehen im direkten Zugriff folgende mathematische Operationen zur Verfügung:

Wurzel einer Zahl

Syntax: *√(Zahlenwert)*

Um das Wurzelzeichen aufzurufen drücke bitte die Tastenkombination AltGr+W.
Die Klammer beschreibt die Weite der Wurzeloperation.

Bsp.: PRINT √(4+3.7) ->Ausgabe 2.77489

Potenz einer Zahl

Syntax: *Wert1 ^ Wert2*

Potenz einer Zahl heisst das die um Wert2 fache Multiplikation von Wert1.

Bsp.:PRINT 5^7 -> Ausgabe 78125.9

Modulo-Operation

Syntax: *Wert1 // Wert2*
Modulo berechnet den Rest der Division Wert1 geteilt durch Wert2

Bsp.: PRINT 10 // 8 -> Ausgabe 2

Absolutwert einer Zahl

Syntax: !Zahl

Absolutwert bedeutet der vorzeichenlose Anteil eines Rechenwertes.

Bsp.: PRINT !8-12 -> Ausgabe 4

Die Kreiszahl PI

Syntax: π

Das Symbol für PI befindet sich auf der Tastenkombination AltGr+P.

Bsp.: PRINT π -> Ausgabe 3.14159

Erweiterte mathematische Funktionen

Syntax: FN <*Funktion*>

In der folgenden Tabelle sind alle zur Zeit implementierten mathematischen Funktionen, die über den Befehl FN aufgerufen werden, enthalten.

SIN(X)	Sinus (X im Bogenmaß)
COS(X)	Cosinus (X im Bogenmaß)
TAN(X)	Tangens (X im Bogenmaß)
LN(X)	natürlicher Logarithmus (X>0)
E(X)	Exponentialfunktion e^x
FRC(X)	Fractionalwert (Nachkommawert von X)
INT(X)	Integerwert von X
LOG(X)	Logarithmus zur Basis 10
EXP(X)	Exponentialfunktion 10^x
ATN(X)	Arcustangens
FL(X)	Rückgabe des nächst kleineren Integerwertes
CL(X)	Rückgabe des nächst höheren Integerwertes
MIN(X1,X2)	Rückgabe des kleineren Wertes von X1 und X2
MAX(X1,X2)	Rückgabe des größeren Wertes von X1 und X2

Bsp.:

```
PRINT FN SIN(45)    ->  0.8509
PRINT FN COS(33)  ->  -0.0132714
PRINT FN TAN(76)  -> 0.686751
PRINT FN LN(2.45)  -> 0.896086
PRINT FN E(1.2)  -> 3.32013
PRINT FN FRC(47.234) -> 0.234
PRINT FN INT(56.34) -> 56
PRINT FN LOG(12.3) -> 1.08991
PRINT FN EXP(2) -> 100
PRINT FN ATN(23) -> 1.52735
PRINT FN FL(48.8) -> 48
PRINT FN CL(48.1) -> 49
PRINT FN MIN(23.4,67.8) -> 23.4
PRINT FN MAX(23.4,67.8) -> 67.8
```

Erzeugung von Zufallszahlen

Syntax: RND*(Wert)*

Mit RND wird eine Zufallszahl im Bereich von 0.000 bis *Wert* erzeugt.

Bsp.: PRINT RND(5) - erzeugt eine Zufallszahl zwischen 0.000 und 5.000

Der zurückgegebene Wert kann bis zu 3 Nachkommastellen haben (z.Bsp.:RND(4) - erzeugt 2.567). Sind ganze Zahlen erwünscht, so erweitere den Befehl mit der Integer-Funktion FN INT(x).

Bsp.:PRINT FN INT(RND(5)) -> Ausgabe eines zufälligen Integerwertes.

Funktionen der seriellen Schnittstelle

Ich habe lange überlegt, ob die Unterstützung von seriellen Schnittstellen-Funktionen überhaupt ins TRIOS-BASIC integriert werden. Mir fiel dann aber ein, das ich bei meinen früheren AVR-Projekten fast immer darauf angewiesen war, irgendwelche Informationen von außen in den AVR zu bekommen, um sie dann weiter zu verarbeiten. Außerdem macht es das HIVE-System interessanter, wenn es mit anderer Peripherie interagieren könnte. Um das ganze trotzdem so einfach, wie möglich zu halten wurden lediglich 3 Befehle ins Leben gerufen, die dies ermöglichen sollen.

Serielle Schnittstelle öffnen

Syntax: COM *on,baud*

COM öffnet oder schliesst die serielle Schnittstelle, je nachdem, was dem Parameter *on* übergeben wird *on* 1 bedeutet öffnen und *on* 0 schliessen.

Die Schnittstellenparameter, welche normalerweise angegeben werden müssen, sind hier schon fest vorgegeben. Lediglich die Baudrate wird mit *baud* angegeben.

Format = 8 Bit , no Parity

Bsp.: COM 1,57600 ...öffnet die serielle Schnittstelle mit einer Übertragungsrate von 57600 Baud

Dies genügt nun schon, um erste Versuche zu unternehmen.

Öffne ein Terminalprogramm auf Deinem Computer ,stelle die obigen Parameter ein und starte die Verbindung auf dem PC (**achte darauf, das die Dip-Schalter auf dem HIVE-Board auf Regnatix stehen!**).

Gib etwas auf der Hive-Tastatur ein. Wenn Du alles richtig gemacht hast, sollte der eingegebene Text auch im Terminal-Fenster zu sehen sein. Alles, was auf dem Bildschirm an Text ausgegeben wird, wird nun solange zur seriellen Schnittstelle gesendet, bis Du den Befehl COM 0 eingibst (schliesst die serielle Schnittstelle).

Das Senden von Daten funktioniert nun schon, fehlt nur noch der Datenempfang. Dafür sind zwei , von der Funktion fast identische Befehle vorgesehen.

Daten von der Seriellen Schnittstelle empfangen ohne zu warten

Syntax: COMG

COMG empfängt Daten von der seriellen Schnittstelle ohne auf ein Zeichen zu warten. Das heisst, dieser Befehl ist dazu gedacht, Schnittstellenabfragen zu realisieren, ohne den Programmfluss zu unterbrechen. Er muss also in einer Abfrageschleife verwendet werden. Folgendes Beispiel demonstriert dieses Vorgehen. Es wird in einer Schleife die serielle Schnittstelle abgefragt und parallel dazu die Uhrzeit am rechten oberen Bildschirmrand dargestellt. Öffne wieder Dein Terminalprogramm auf dem PC und verbinde es mit dem HIVE.

Nun werden alle Tastatureingaben auf Deiner PC-Tastatur auf dem HIVE-Bildschirm dargestellt, bis die Enter-Taste betätigt wird.

Bsp.:

```
5 CROFF
10 CLS
20 COM 1,57600
30 a=COMG
40 IF a>0 THEN PRINT CHR$ (a);
50 IF a=13 THEN END
60 x=GETX :y=GETY
70 TIME 30,0
80 POS x,y
100 GOTO 30
```

Manchmal ist es wünschenswert, das der Computer solange wartet, bis ein Zeichen von der seriellen Schnittstelle eintrifft. dafür ist der folgende Befehl gedacht.

Daten von der Seriellen Schnittstelle empfangen (auf Zeichen warten)

Syntax: COMR

COMR wartet auf Daten von der seriellen Schnittstelle. Der Programmfluss wird solange unterbrochen bis ein Zeichen erkannt wird.

Um den Unterschied zu demonstrieren, ändern wir unser vorheriges Beispiel entsprechend ab.

Bsp.:

```
5 CROFF
10 CLS
20 COM 1,57600
30 a=COMR
40 PRINT CHR$ (a);
50 IF a=13 THEN END
60 x=GETX :y=GETY
70 TIME 30,0
80 POS x,y
100 GOTO 30
```

Schon beim Start ist zu erkennen, das die Uhrzeit erst angezeigt wird, wenn ein Zeichen zur seriellen Schnittstelle gesandt wird. Das liegt daran, das die Ausgabe der Uhrzeit erst nach dem COMR-Befehl aufgerufen wird und demzufolge nur nach Tastendruck aktualisiert werden kann.

Also überlege, welchen Befehl Du verwenden musst, damit Dein Programm keine ungewollte Unterbrechung erfährt.

SID-Sound-Befehle

Als besonderes Highlight ist die SID-Sound-Emulation in TRIOS-BASIC zu betrachten. Sie ermöglicht echte Retro-Klänge ganz im Stile eines C64-Homecomputers der 90er Jahre.

Um allerdings dem Hive diese Klänge zu entlocken, ist ein wenig Programmierarbeit nötig, wird dann allerdings mit einer sehr realen SID-Emulation belohnt.

Die Emulation geschieht in stereo und verwendet 3 Soundkanäle. Alle Kanäle können gleichzeitig benutzt werden.

Die Kanäle sind separat programmierbar, einige Befehle allerdings beziehen sich auf alle Kanäle gleichzeitig, wie zum Beispiel der Befehl VOL.

Sound-Lautstärke einstellen

Syntax: VOL *wert*

Bevor mit der Programmierung der diversen Soundbefehle begonnen wird, ist es nötig die Lautstärke, mit der der Sound abgespielt werden soll, einzustellen.

Dies geschieht mit dem Befehl VOL gefolgt durch einen Wert zwischen 0 (stumm) bis 15 (ganz laut)

Bsp.: VOL 15 -> stellt die Lautstärke auf den Maximalwert (ganz laut)

Die Lautstärkeeinstellung bezieht sich auf alle Kanäle, ist also sozusagen die Gesamtlautstärke.

Wenn Du diesen Befehl eingegeben hast, tut sich offenbar noch nichts aber das ändert sich gleich.

Wellenform wählen

Syntax: WAVE *Kanal,WForm*

Als nächstes müssen wir die Wellenform, welche erklingen soll, wählen. Wellenform bezeichnet den Charakter des Klanges der erzeugt werden soll. Es stehen 4 Wellenformen zur Verfügung.

Dreieck = 1 (warmer Klang, wenig Oberwellen)

Sägezahn = 2 (scharfer Klang, viele Oberwellen)

Rechteck = 3 (hohler Klang, einige Oberwellen)

Rauschen= 4 (grollender bis zischender Klang, je nach Tonhöhe)

Wir wählen für unseren Versuch die Wellenform Dreieck

Bsp.: WAVE 0,1 ->wählt für Kanal 0 die Wellenform Dreieck

Als nächstes müssen wir noch den Lautstärkeverlauf oder in der Musiker-Fachsprache auch Lautstärke Hüllkurve des zu erzeugenden Klanges einstellen.

Lautstärke-Hüllkurve einstellen

Syntax: ADSR *Kanal,Attack,Decay,Sustain,Release*

ADSR steht für die zu übergebenen Parameter Attack, Decay Sustain, Release.

Diese haben folgende Bedeutung:

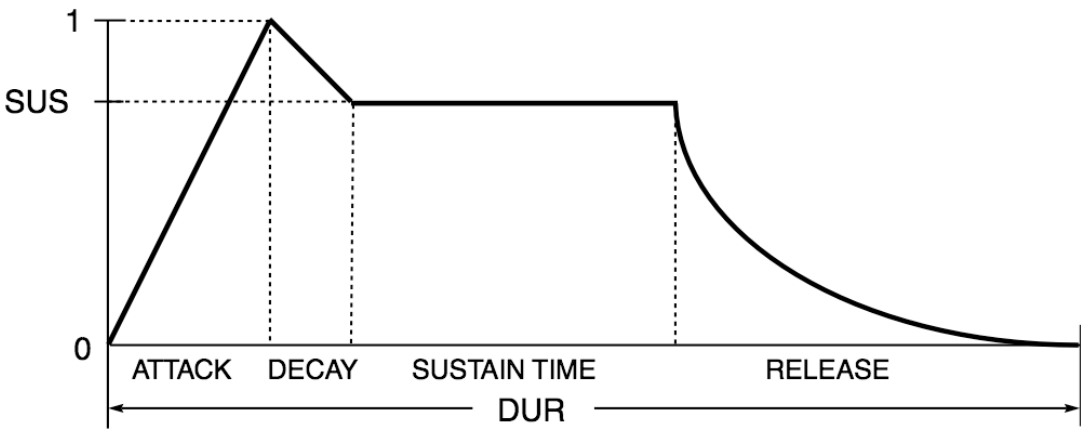
Attack - Lautstärke-Anstiegszeit

Decay - Zeit bis zum erreichen der Haltelautstärke (Sustainpegel)

Sustain - Pegel, der bis zum Loslassen einer Taste gehalten wird

Release - Abkling- oder Ausklingzeit des Klanges

Wie diese Begriffe zu verstehen sind wird in folgender Grafik dargestellt.



Bsp.: ADSR 0,0,10,30,10 ->ADSR-Einstellung für Kanal 0

Attackwert =0 -> harter Anschlag

Decay =10 ->kurzer Abfall zum Sustainpegel

Sustain = 30 ->lauter Pegel bis zum loslassen

Release = 10 -> kurze Ausklingzeit

Die Wertebereiche für diese Parameter liegen zwischen 0 und 255.

Nun wollen wir mal sehen, was wir getan haben und lassen einen Ton erklingen.

Note spielen

Syntax: NT *Kanal,Tonhöhe*

NT lässt für *Kanal* die übergebene *Tonhöhe* erklingen.

Die SID-Emulation kann Tonhöhen von 0-3,9kHz erzeugen. Da es für Melodien zu umständlich wäre, für jeden Ton die entsprechende Frequenz zu errechnen, macht dies der Interpreter für Dich.

Die zu übergebene Tonhöhe muss ein Ganzzahlwert im Bereich von 0-255, wobei jeder Wert einen Halbton darstellt. Als Orientierung: 60 ist zBsp. ein C, 61 ein Cis, 62 ein D usw.

Bsp.:NT 0,60 -> es erklingt ein C auf Kanal 0

Was ist geschehen? Der Ton hört gar nicht auf. Deswegen schnell zum nächsten Befehl. Die Note ausklingen lassen.

Note ausklingen lassen

Syntax: NTOFF *Kanal*

Lässt für den gewählten Kanal die Note ausklingen

Bsp.: NTOFF 0 -> Ton auf Kanal 0 ausklingen lassen.

Filter setzen

Syntax: FLT *lp,hp,bp*

Setzt einen von drei möglichen Filtertypen.

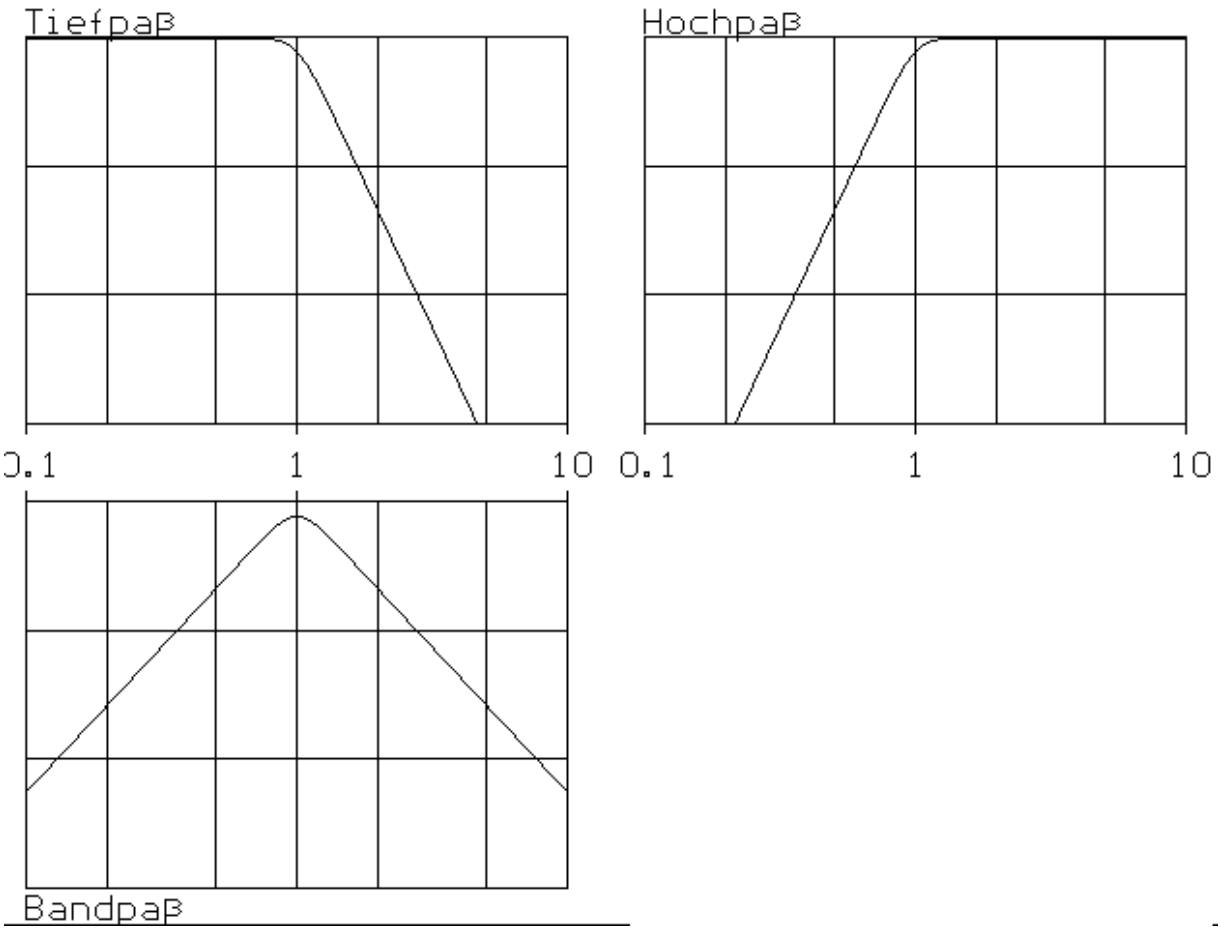
lp=Lowpass (Tiefpassfilter)

hp=Highpass (Hochpassfilter)

bp=Bandpass (Bandpassfilter)

Die Funktion der einzelnen Filtertypen ist sehr verschieden. Während der Tiefpassfilter die Tiefen Töne passieren lässt und hohe Töne sperrt, reagiert der Hochpassfilter genau entgegengesetzt.

Ein Bandpassfilter lässt nur einen definierbaren Bereich des Frequenzspektrums passieren. Folgende Grafik stellt die einzelnen Filtertypen in ihrer Funktion dar.



-1=Filter zuweisen, 0=keine Filterzuweisung.

Bsp.: FLT 0,0,-1 es wird der Bandpassfilter ausgewählt.

Filter zuweisen

Syntax: FMASK *chan1,chan2,chan3*

Weist den gewählten Filtertyp den Kanälen zu, die ihn verwenden sollen.
-1 Filter zuweisen, 0=keine Filterzuweisung

Bsp.: FMASK 0,0,-1 weist Kanal 3 den gewählten Filtertyp zu.

Filter-Cutoff-Einstellung setzen

Syntax: CUT *wert*

Setzt die Cutoff-Frequenz des ausgewählten Filters.
Werte im Bereich von 0-1100 werden verarbeitet.

Bsp.: CUT 870

Filter-Resonanzfrequenz setzen

Syntax: RES *wert*

Setzt den Resonanzwert des Filters, bei der der Klang am wenigsten vom Filter blockiert wird. Gültige Werte liegen im Bereich von 0-15.

Ringmodulator verwenden

Syntax: RGMOD *chan1,chan2,chan3*

Mit dem Ringmodulator ist es möglich, die Kurvenform eines Klanges von der Kurvenform eines zweiten Klanges zu modulieren. Damit sind sehr ungewöhnliche Klangvariationen möglich.

-1 setzt für den entsprechenden Kanal, die Modulation

0 schaltet die Modulation aus

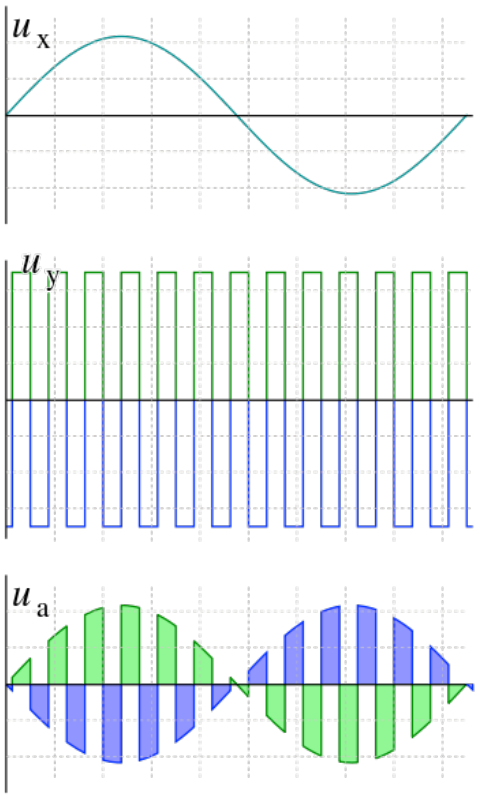
chan1=moduliert chan2

chan2=moduliert chan3

chan3=moduliert chan1

Bsp.:RGMOD 0,-1,0 -> chan2 moduliert chan3

Wie die Ringmodulation zu verstehen ist, zeigt folgende Grafik.



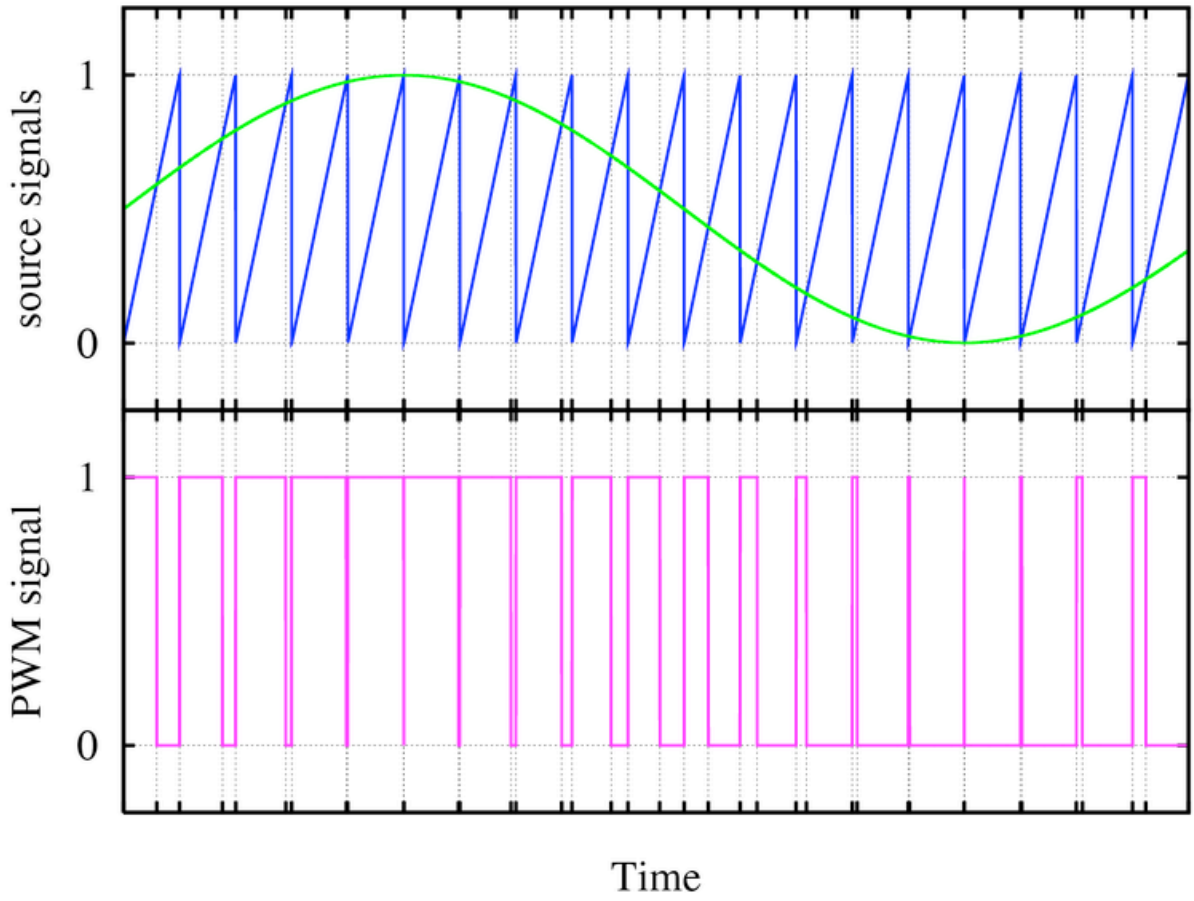
Pulsweitenmodulation

Syntax: PWM *chan,wert*

Ermöglicht die Pulsweitenmodulation von *chan* mit dem Parameter *wert.*
Diese Modulationsart ist nur auf Wave 3 anwendbar (Rechteck).

Bsp.: PWM 2,1024

Wie die PWM-Funktion agiert, ist in folgender Grafik zu sehen.



Synchronisation zweier Kanäle

Syntax: SYNC *chan1,chan2,chan3*

Synchronisiert die Wellenformen zweier Kanäle, ebenfalls eine Möglichkeit einer ungewöhnlichen Klangverformung.

-1 schaltet die Synchronisation ein

0 schaltet die Synchronisation aus

chan1=Kanal 1 synchronisiert Kanal 2

chan2=Kanal 2 synchronisiert Kanal 3

chan3=Kanal 3 synchronisiert Kanal 1

Bsp.: SYNC 0,-1,0 -> Kanal 2 synchronisiert Kanal 3

SID-DMP-Datei abspielen

Syntax: PLAY „dmpdatei“ *{Option}*

Wird der Dateiname einer SID-Dmp-Datei angegeben, spielt TRIOS-BASIC diese ab.

Option kann folgendes sein:

1=Player pausieren

0=Player stoppen

Bsp.:

PLAY“Axe.dmp“ -> Dmp-Datei abspielen

PLAY 1 -> pausiert den Player PLAY 1 -> Player spielt weiter

PLAY 0 -> Player stoppen

HINWEIS:

Die mitgelieferten Sound-Dateien befinden sich im Unterverzeichnis „DMP“.

Um sie abzuspielen, wechsle erst mit CHDIR“DMP“ in dieses Verzeichnis.

ACHTUNG:

Wenn eine SID-DMP-Datei abgespielt wird, ist der Zugriff auf die SD-Card blockiert. Jeder Zugriff auf die SD-Card mit Dateibefehlen unterbricht sofort die Wiedergabe.

Player-Position abfragen

Syntax: GDMP

Fragt die aktuelle Position des Players innerhalb einer SID-DMP-Datei ab.

Bsp.: a=GDMP

GDMP ist beim Start des Players am grössten und verringert seinen Wert bis 0, was dem Ende der SID-DMP-Datei entspricht.

System-Beep

Syntax: BEEP *{wert}*

Beep ohne Parameter lässt den System-Beep erklingen, um zum Beispiel Fehleingaben akustisch zu untermalen.

Beep mit dem Parameter, *wert* von 0-255 kann als kleines Soundmodul genutzt werden.

Bsp.: BEEP 88

Dateifunktionen

Datei öffnen

Syntax: OPEN *„Dateiname“,Option*

Öffnet eine Datei mit dem Namen Dateiname mit folgenden Zugriffsoptionen.

R=lesen

W=schreiben

A=anhängen

Bsp.: OPEN“Test.dat“,R -> öffnet die Datei Test.dat zum schreiben

Werte aus Datei lesen

Syntax: READ *Var,Var,Var...*

Liest einen oder mehrere Werte aus der mit OPEN“Name“,R geöffneten Datei und übergibt sie den angegebenen Variablen.

Bsp.:

```
10 OPEN“Test.dat“,R
20 READ a,b,c
30 CLOSE
40 Print a,b,c
```

Werte in Datei schreiben

Syntax: WRITE *„String“|Var,Var,Var...*

Schreibt die übergebenen Werte in eine geöffnete Datei.

Bsp.:

```
10 OPEN“Test.dat“,W
20 a=12:b=33:c=67
30 WRITE a,b,c
40 CLOSE
```

Geöffnete Datei schliessen

Syntax: CLOSE

Schließt eine geöffnete Datei.

Datei auf der SD-Card löschen

Syntax: DEL*“Dateiname“*

Löscht eine, auf der SD-Card befindliche Datei. Wird die Datei nicht gefunden oder ist die Datei schreibgeschützt, erfolgt eine Fehlermeldung.

Bsp.: DEL“Test.dat“ -> Die Datei „Test.dat“ wird gelöscht

Datei auf der SD-Card umbenennen

Syntax: REN *“Dateiname alt“,“Dateiname neu“*

Es wird die Datei mit dem Namen *Dateiname alt* umbenannt in *Dateiname neu*.

Die Umbenennung ist nur erfolgreich, wenn die Datei existiert und nicht schreibgeschützt ist.

Bsp.: REN „Test.dat“,“Fest.bat“

Verzeichnis wechseln

Syntax: CHDIR *“Name“*

Wechselt in das Verzeichnis *Name*.

CHDIR “.“ wechselt wieder eine Ebene nach oben.

CHDIR “..“ wechselt ins Root-Verzeichnis.

Bsp.: CHDIR “DMP“ -> es wird in das Verzeichnis DMP gewechselt.

Leider kann der Befehl nicht CD heissen, da er intern mit der Hexadezimalausgabe kollidiert.

Verzeichnis anzeigen

Syntax: DIR *“FILTER“,mode* **oder DIR** *x,y,Zeilen,mode*

Es wird das aktuelle Verzeichnis entweder mit Filter oder mit Parameterübergabe angezeigt. Gleichzeitig wird die Dateiliste im E-RAM gespeichert.

Anzeige mit Filter:

Filter kann benutzt werden, um Dateien mit bestimmten Dateiendungen anzuzeigen.

z.Bsp.: DIR “BAS“,1 -> zeigt alle Dateien mit der Dateierweiterung BAS an.

mode hat folgende Bedeutung

0=unsichtbar, dies kann benutzt werden, um Dateilisten einzuladen, ohne das Programmbild zu stören.

1=einfache Ausgabe der Dateiliste (es werden nur die Dateinamen anzeigt)

2=erweiterte Ausgabe (es werden zusätzlich Größe und Erstellungsdatum angezeigt)

Anzeige mit Parameterübergabe:

Die zweite Möglichkeit der Verwendung des DIR-Befehls ist die Anzeige mit Darstellungsparametern. Damit ist es möglich die Dateilisten-Ausgabe den eigenen Bedürfnissen bei der Darstellung anzupassen.

Bsp.:DIR 2,3,10,2 -> Dateiliste wird in erweiterter Darstellung an Position x=2 und y=3 mit 10 Zeilen ausgegeben.

HINWEIS:

Die DIR-Dateiliste wird im E-RAM gespeichert (allerdings nur die Dateinamen).

Eine Weiterverarbeitung kann mit dem Befehl GFILE erfolgen.

Mit der Taste F6 wird die Dateiliste mit den aktuellen Einstellungen dargestellt.

Basic-Programm speichern

Syntax: SAVE *“Name.Bas“,mode*

Speichert das aktuelle Programm auf SD-Karte.

Mit *mode* wird die Art der Speicherung festgelegt. Wird *mode* weggelassen, erfolgt die Speicherung Zeile für Zeile als Textdatei. Eine spätere Editierung am PC ist möglich.

Wird als mode eine 1 eingegeben, wird das Programm als Binärdatei gespeichert. Eine spätere Editierung am PC ist nicht mehr möglich, dafür geht das Speichern

schneller, muss aber beim Laden ebenfalls mit dem mode 1 geladen werden.

Bsp.:SAVE“Test.Bas“ -> Speichert das Programm als Textfile

SAVE“Test.BBS“,1 ->Speichert das Programm als Binärdatei

HINWEIS:

Ich habe bewusst im zweiten Beispiel eine andere Dateierweiterung benutzt, da es übersichtlicher ist, beide Dateitypen unterscheiden zu können, da sie auch unterschiedlich geladen werden müssen. Welchen Dateizusatz Du verwendest, ist Dir überlassen (ob BBS oder DAT oder was weiss ich, ist egal).

Wird nur SAVE eingegeben (oder F3 gedrückt) und die Enter-Taste betätigt, legt der Interpreter eine Schnellsicherung unter dem Namen BAS.TMP im Root-Verzeichnis an. Diese wird automatisch als Binärdatei gesichert.

Dateien, die mit dem mode 1 gesichert wurden, sind am PC nicht mehr editierbar.

Programm laden

Syntax: LOAD *“Name.Bas“,mode*

Lädt eine zuvor gesicherte Datei in den Arbeitsspeicher von TRIOS-BASIC.

Wird nur der Dateiname angegeben, erwartet der Interpreter eine als Textdatei gesicherte Datei (Save „name.bas“).

Mit LOAD“name.bbs“,1 wird eine zuvor gesicherte Binärdatei geladen und automatisch gestartet.

Bsp.: LOAD“Test.Bas“ -> lädt die Datei als Textdatei

LOAD“Test.BBS“,1 ->lädt die Datei als Binärdatei und startet automatisch

LOAD ohne die Angabe eines Dateinamens, lädt die Sicherungsdatei BAS.TMP und startet sie automatisch.

Alternativ kann auch die Taste F2 benutzt werden.

HINWEIS:

Der Unterschied der zwei möglichen LOAD und SAVE-Kammandos ist die Art der Bearbeitung, entweder als Textdatei dafür langsamer aber am PC editierbar oder schneller aber nicht mehr am PC editierbar.

Auf dem HIVE sind beide Varianten, weiterhin editierbar.

In jedem Fall sollte man die Dateien entsprechend benennen, um die Art der Speicherung optisch auseinanderhalten zu können, wie, ist Dir überlassen.

Zeichen aus einer Datei lesen oder in eine Datei schreiben

Syntax: FILE

File ist die , im Gegensatz zu READ und WRITE, einfachere Form fortlaufend in eine Datei zu schreiben oder aus dieser zu lesen.

Bsp.:

```
10 OPEN "color.bas",r
15 b=GATTR (0)
20 FOR i=1 TO b
30 a=FILE
40 PRINT CHR$ (a);
50 NEXT i
60 CLOSE
```

Dieses Beispiel liest alle Zeichen aus der Datei „COLOR.BAS“ und zeigt sie auf dem Bildschirm an.

Im folgendem Beispiel werden in die Datei „Color.TXT“ Zahlenwerte im Bereich 33 bis 256 in Binärform geschrieben.

Bsp.:

```
10 OPEN "color.txt",w
20 FOR i=33 TO 256
30 FILE =i
50 NEXT i
60 CLOSE
```

Zeichen aus einer Datei lesen oder in eine Datei schreiben

Syntax: GFILE *{Nr}*

GFILE ohne Parameter gibt die Anzahl der mit dem DIR-Befehl gefundenen Dateieinträge zurück.

GFILE *Nr* übergibt den Dateinamen der mit Nr gekennzeichneten Datei an die Stringvariable #Z.

Bsp.:

```
10 DIR“BAS“,0
20 a=GFILE
30 GFILE 2
40 PRINT #Z
50 PRINT“Anzahl der gefundenen Einträge:“;a
```

im Beispiel wird die Dateiliste mit dem Filter BAS in den E-RAM geschrieben (alle Basic-Dateien). In Zeile 20 wird die Anzahl der gefundenen Dateien an die Variable a übergeben. In Zeile 30 Wird der zweite gefundene Dateieintrag an die Stringvariable #Z übergeben (dies geschieht intern). In Zeile 40 wird der zweite Dateiname ausgegeben und in Zeile 50 die Gesamtanzahl der gefundenen Dateien.

HINWEIS:

Die Stringvariable #Z wird intern vom Interpreter benutzt, um Zugriff auf den mit GFILE Nr übergebenen Dateinamen zu erhalten. Das musst Du bei der Programmierung mit Stringvariablen beachten um ein ungewolltes Überschreiben von #Z zu verhindern.

Abfragen von Dateiattributen

Syntax: GATTR *(wert)*

Oft sind Informationen über die Größe einer Datei ,deren Erstellungs- oder

Änderungsdatum o.ä. nötig. Dafür wurde GATTR geschaffen.

Wert bestimmt, welche Informationen abgefragt werden sollen.

0 = Dateigröße
1 = Erstellungsdatum - Tag
2 = Erstellungsdatum - Monat
3 = Erstellungsdatum - Jahr
4 = Erstellungsdatum - Sekunden
5 = Erstellungsdatum - Minuten
6 = Erstellungsdatum - Stunden
7 = Zugriffsdatum - Tag
8 = Zugriffsdatum - Monat
9 = Zugriffsdatum - Jahr
10 = Änderungsdatum - Tag
11 = Änderungsdatum - Monat
12 = Änderungsdatum - Jahr
13 = Änderungsdatum - Sekunden
14 = Änderungsdatum - Minuten
15 = Änderungsdatum - Stunden
16 = Read-Only-Bit
17 = Hidden-Bit
18 = System-Bit
19 = Directory
20 = Archiv-Bit

Bsp.:

```
10 OPEN“COLOR.BAS“,R
20 A=GATTR(0)
30 PRINT A
```

Das Beispiel fragt die Dateigröße von “COLOR.BAS“ ab und gibt sie auf dem Bildschirm aus.

Verzeichnis erstellen

Syntax: MKDIR *“NAME“*

Erstellt ein Verzeichnis *NAME* auf der SD-Karte.

Binärdateien laden

Syntax: BLAOD *„Name“*

Mit diesem Befehl ist es möglich eine externe Binärdatei zu laden, um zum Beispiel ein neues, für den Hive erstelltes Programm zu starten, ohne extra zur REGIME-Kommandozeile zurückzukehren.

Wird das ausgeführte Programm wieder verlassen, landest Du allerdings wieder in REGIME, da dieser, das von Regnatix verwendete Startprogramm ist. Alle die die Flash-Basic-Version haben kehren wieder zurück zum Basic-Interpreter.

Ein eventuell im Speicher des E-Ram‘s befindliches Programm sollte sich nach der Rückkehr mit F10(Reclaim) wieder herstellen lassen können. Voraussetzung hierfür ist, das der E-Ram nicht durch andere Vorgänge verändert oder Dein Hive nicht zwischenzeitlich ausgeschaltet wurde.

Bsp.:BLOAD“Testdatei.bin“

HINWEIS:

Bei Eingabe des Dateinamens, spielt Groß-oder Kleinschreibung keine Rolle. Allerdings muss die Dateierweiterung mit angegeben werden.

Welche Dateien wurden mit TRIOS-BASIC mitgeliefert?

Im ROOT-Verzeichnis

-Bel.Sys	Grafiktreiber
-Reg.Sys	Kommandozeilen-Interpreter REGIME (angepasste Version)
-Regfont.dat	Zeichensatz für REGIME
-Regime.txt	Hilfedatei zu REGIME
-Bas.Tmp	Basic-Sicherungsdatei

Im Verzeichnis BASIC

-Baller.Bas	kleines Ballerspiel
-Baller.Bbs	gleiche Datei in Binärform
-Beep.Bas	Demo zur Funktion Beep
-Buttons.Bas	Demo zur Button-Funktion
-Buttons.Bbs	gleiche Datei im Binärformat
-Climber.Bas	ein Level des Spiels portiert vom KC87 (leider sehr langsam)
-Climber.Bbs	gleiche Datei in Binärform
-Color.Bas	Demo mit Farben
-Colorbox.Bas	Demo mit dem Box, Frame und Window-Befehl
-Demo.Bas	Demo vieler TRIOS-BASIC-Funktionen
-Dmpplay.Bas	Dmp-Player mit Maus-Bedienung
-Dmpplay.bbs	gleiche Datei in Binärform
-Fonts.Bas	Kleine Spielerei mit Tile-Fonts
-Kreis.Bas	farbige Kreise zeichnen

-Kreis.Bbs	gleiche Datei in Binärform
-Lmap.Bas	Laden einer Map-Datei
-Map.Bas	Demo Map-Datei erstellen
-Map.Bbs	gleiche Datei in Binärform
-Map.Dat	die mit Map.Bas erstellte Map-Datei
-Mappl.Bas	Programm, mit dem der Bildschirm für den Dmp-Player ertellt wurde.
-Mappl.Dat	Map-Datei für den Dmp-Player
-Mousbutt.Bas	Maus-Button-Demo
-Mouse.Bas	Demo zur Mausabfrage
-Piano.Bas	Kleines Piano-Programm mit SID-Sound (über die Tastatur)
-Tile.Bas	Tile-Grafiken anzeigen
-Tile.Bbs	gleiche Datei in Binärform
-Pong64.Bas	ein hüpfender Ball
-Pong64.bbs	gleiche Datei in Binärform
-Timer.Bas	Timer-Demo
-Errors.Txt	Fehler-Text von TRIOS-BASIC (nicht löschen!!!)
-Bas64.bel	Grafiktreiber für TRIOS-BASIC
-Bas.Adm	Administra-Treiber
-Help.Bin	Hilfedatei (aufrufbar mit F1)
-BASIC.Bin	der Basic-Interpreter
CombinedWaveforms.Bin	Systemdatei für Sid-Sound-Emulator

Die orange unterlegten Dateien sind Systemdateien, die zum Betrieb von TRIOS-BASIC notwendig sind und dürfen nicht entfernt werden.

Das Verzeichnis Help

Beinhaltet alle für die Hilfefunktion notwendigen Dateien und dürfen nicht gelöscht oder verschoben werden.

Das Verzeichnis TILE

Hier befinden sich die Tile-Grafik-Dateien. Tile-Dateien, die TRIOS-BASIC verwenden soll, müssen sich in diesem Ordner befinden.

Name	Funktion	TLOAD-Parameter
Bunny.dat	Tile-Grafik	11,9
Bunny0.dat	Tile-Grafik	12,9
Bunny2.dat	Tile-Grafik	11,8
Bunny3.dat	Tile-Grafik	8,14
Bunny4.dat	Tile-Grafik	9,13
Bunny5.dat	Tile-Grafik	13,8
Bunny6.dat	Tile-Grafik	8,14
Bunny7.dat	Tile-Grafik	10,11
Bunny8.dat	Tile-Grafik	9,14
Bunny9.dat	Tile-Grafik	8,16
Bunny10.dat	Tile-Grafik	7,16
Bunny11.dat	Tile-Grafik	8,14
Bunny12.dat	Tile-Grafik	10,13
Bunny13.dat	Tile-Grafik	10,12
Bunny14.dat	Tile-Grafik	8,13
Bunny15.dat	Tile-Grafik	11,11
Bunny16.dat	Tile-Grafik	9,14
Bunnya.dat	Tile-Grafik	13,10
Font1.dat	Tile-Font	16,11
Font1.dat	Tile-Font	16,11
Font2.dat	Tile-Font	16,11
Font3.dat	Tile-Font	16,11
Font4.dat	Tile-Font	16,11

Name	Funktion	TLOAD-Parameter
Font5.dat	Tile-Font	16,11
Font6.dat	Tile-Font	16,11
Font7.dat	Tile-Font	16,11
Font8.dat	Tile-Font	16,11
Font9.dat	Tile-Font	16,11
Fontkc.dat	Tile-Font	16,11
Fonts.dat	Tile-Font	16,11
Fontts.dat	Tile-Font	16,11
Man.dat	Mouse-Tile	1,1
Mouse1.dat	Mouse-Tile	1,1
Mouse2.dat	Mouse-Tile	1,1
Mouse3.dat	Mouse-Tile	1,1
Rechner.dat	Tile-Font	16,11
Sysfont.dat	Tile-Font	16,11
Sysfont1.dat	Tile-Font	16,11
Sysfont2.dat	Tile-Font	16,11
sysfontb.dat	Tile-Font	16,11
tsfont.dat	Tile-Font	16,11

Das Verzeichnis DMP

Hier befinden sich alle SID-Sounddateien.

Axe.dmp
Aztec.dmp
Batman.dmp
Blue.dmp
Bob.dmp
Boulder.dmp
BOZ.DMP
burner.dmp
Cobra.dmp
Comic.dmp
Commando.dmp
Dan.dmp
Dragons.dmp
Eye.dmp
Funkrock.dmp
Ghosts.dmp
Hammer.dmp
IceAge.dmp
Lamebada.dmp
Monday.dmp
Oxygen_4.dmp
Oxyron.dmp
Reggae.dmp
S_O_S.dmp

Befehlsübersicht von TRIOS-BASIC 2.0

Vorweg gleich der Hinweis, diese Befehlsübersicht spiegelt nur den aktuellen Versionsstand von TRIOS-Basic wieder und kann sich jeder Zeit ändern.

Programmablaufbefehle:

IF...THEN...ELSE
(ON)..GOTO
(ON)..GOSUB...RETURN
RUN
END
PAUSE

Datums-und Zeitfunktionen

STIME
SDATE
GTIME
GDATE
SDOW
TIME
TIMER

Ein-Ausgabebefehle

PEEK
POKE
INPUT
REM
INKEY

Systembefehle

BYE
COGS
DUMP
FREE
NEW
LIST
CLEAR
TRON *
TROFF *
EDIT
VER
RECLAIM *

Logische Operatoren

NOT
AND
OR

Mouse-Befehle

MX
MY
MZ
MB
MOUSE
MBOUND

Dateifunktionen
OPEN
READ
WRITE
CLOSE
DEL
REN
CHDIR
DIR
SAVE
LOAD
FILE
GFILE
GATTR
MKDIR
BLOAD

SID-Soundbefehle
NT
NTOFF
VOL
PLAY
ADSR
WAVE
FLT
FMASK
RGMOD
CUT
RES
PWM
SYNC
GDMP
BEEP

Bildschirmbefehle / Funktionen
PRINT
CLS
HOME
POS
TAB
COL
PLOT
SCRDN
SCRUP
CROFF
CRON
TWIN
WIN
TBAR
MBOX
FRAME
WSET
SCROLL
BUTTON
DBUTT
BOX
GETX
GETY

Stringfunktionen
STR$
COMP$
LEN
CHR$
ASC
VAL
STRING$

Schleifen-Befehle

FOR...TO...STEP...NEXT

Grafik-Tile-Befehle

TLOAD
TILE
STILE
TPIC
FONT
MAP

Mathematische Funktionen

RND
FN

Befehle der seriellen Schnittstelle

COM
COMR
COMG

Felder

DIM

*Befehl nur über Funktionstaste aufrufbar!

Die Speicheraufteilung im e-RAM

In folgender Grafik ist die zur Zeit gültige Aufteilung des externen Ram's dargestellt. Diese entspricht, wie die Basic-Befehlsübersicht dem momentanen Versionsstand und kann sich in künftigen Versionen ändern.

RAM-Bank1

Basic Programm-Speicher	$00000 : $1FFFF	128kb	
Bearbeitungs-Speicher	$20000 : $3FFFF	128kb	
Tilespeicher 14x11kB 256x176 Pixel	$40000 : $667FF	154kB	14x11kb
Systemfont	$66800-$693FF	16kb	
Mouse-Tile	$69400-$6943F	64bytes	
Dir-Speicher	$69440-$6AFFF	7103Bytes	
Arrayspeicher A(7,7,7)-Z(7,7,7)	$6B000-$77FFF	52kB	
Shadow-BS-Speicher	$78000-$784CF	1200 Bytes	
FREI	$784B0-$7EFFF	28kB	
Error-Texte	$7F000-$7FAFF	2815Bytes	ca.2,7kB
DIM-Speicher	$7FB00-$7FBFF, $7FC00-$7FCFF		512Bytes
Basic-System-Flags	$7FF00-$7FFFF	255 Bytes	

RAM-Bank2

Map-Speicher	$80000 : $8FFFF	64kB	
Stringarray Speicher #A(7,7,7)-#Z(7,7,7)	$90000 : $FE7FF	442kB	
FREI	$FE800-$FFEAF	5888 Bytes	
TRIOS-System-Flags	$FFF00-$FFFFF	255 Bytes	

Hilfreich ist diese Übersicht für das allgemeine Verständnis der internen Datenverarbeitung von TRIOS-Basic sowie für die direkte Manipulation aus Programmen heraus (z.Bsp. Bei Peek und Poke Anweisungen).
Die mit ***FREI*** bezeichneten Ram-Bereiche werden von TRIOS-Basic nicht benutzt und stehen dem Anwender zur Verfügung.

HINWEIS:

Schreibe nicht unüberlegt in, von TRIOS-Basic verwendete Speicherbereiche. Dadurch können unvorhersehbare Systemzustände entstehen und ein Datenverlust ist nicht auszuschliessen.

Ich hoffe, dieses Handbuch hat Dich in die grundlegenden Funktionen von TRIOS-BASIC 2.0 eingeführt und Lust, eigene Programme auf dem HIVE zu erstellen gemacht. Fragen beantworte ich gern im Forum (hive-project.de). Jedes Basic-Programm, was Du mit uns teilen möchtest, kannst Du im Forum zur Verfügung stellen, um auch anderen Usern den Umgang mit TRIOS-BASIC schmackhaft zu machen und die Weiterentwicklung zu unterstützen.

Ich wünsche allen Basic- und Hive-Freunden viel Spaß und viel Entwicklerdrang mit dieser Programmiersprache.

Drohne Zille9

Widerstand ist zwecklos!