Inhaltsverzeichnis

Kapitel:

Vorwort

Willkommen zu Trios-Basic 2 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

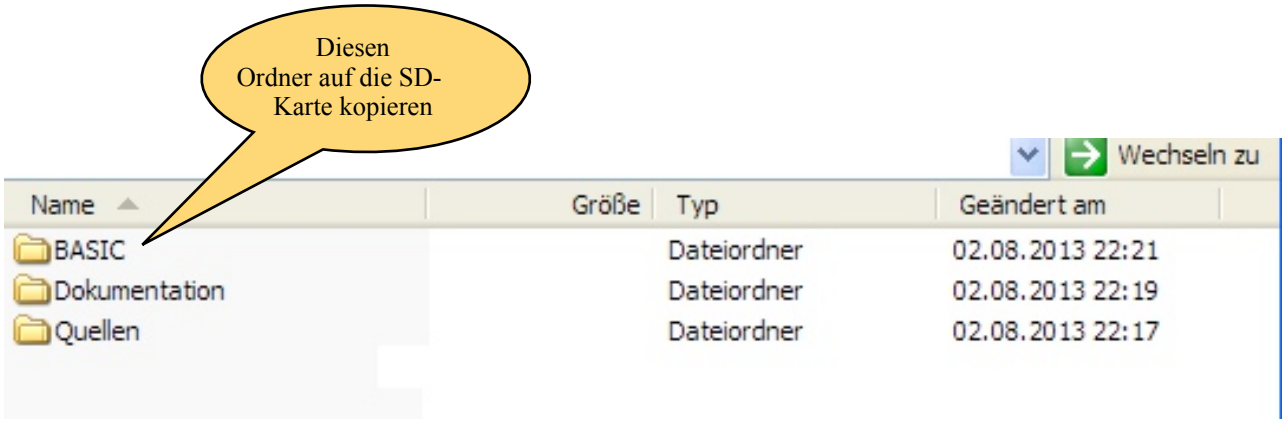
PS.:
Bei Fragen, Hilfestellungen, Kritik und was euch sonst noch einfällt, erreicht Ihr mich im Hive-Forum auf „hive-project.de“

HINWEIS:

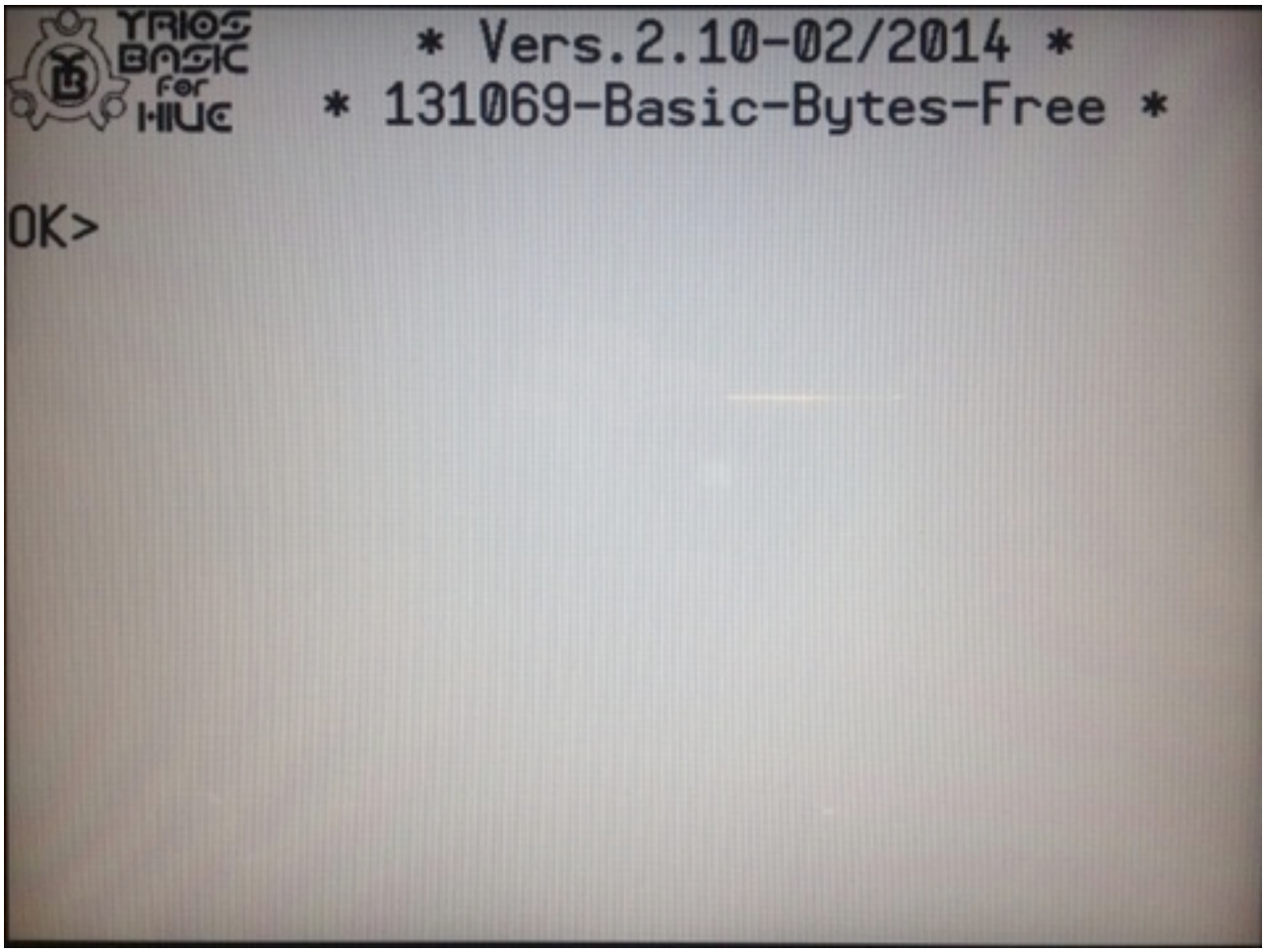
Ich bin kein Schriftsteller, also seht es mir nach, wenn die Struktur dieses Handbuches oder der allgemeine Satzbau nicht so professionell ist, wie einige (gerade Programmierer) es gewohnt sind. Dieses Handbuch ist gerade für die Beginner unter uns gedacht und bewusst (versucht) einfach gehalten auch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

Installation von TRIOS-BASIC

Die Installation von Trios-Basic gestaltet sich sehr einfach. Entpacke die gezippte Paketdatei in ein Verzeichnis auf Deinem PC und kopiere den Ordner „BASIC“ auf eine SD-Karte. Es wird vorausgesetzt, das die SD-Karte startfähig ist (es befinden sich die Dateien Reg.sys und Bel.sys im Hauptverzeichnis).
Starte Deinen Hive und gib in der Regime-Kommando-Zeile cd Basic ein. Betätige ENTER. Gib nun Basic ein und betätige wieder ENTER.
Nun sollte nach einer kurzen Ladezeit der im nächsten Abschnitt gezeigte Bildschirm zu sehen sein.



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 das Wort HELP auf dem Bildschirm ausgegeben. Dieser Befehl hat mehrere mögliche Funktionen, die im Folgenden erläutert werden. Drückst Du nur die Entertaste, so werden alle im TRIOS-BASIC enthaltenen Befehle auf dem Bildschirm ausgegeben.Dies dient der allgemeinen Übersicht.

HELP
Listet alle verfügbaren Befehle auf dem Bildschirm auf

HELP A
listet alle mit dem Buchstaben A beginnenden Befehle auf

HELP PRINT
zeigt den Hilfetext zum Befehl PRINT auf dem Bildschirm an

Diese integrierte Hilfefunktion sollte Dir den ein oder anderen Griff zum Handbuch ersparen und ich hoffe, das sie für Dich ein nützliches Hilfsmittel bei der Programmierung sein wird.

F2 Load-Programme laden
F3 SAVE-Programme speichern

Die LOAD-und SAVE-Taste erspart die Eingabe des gleichlautenden Befehls und muß nur noch um den Dateinamen der gewünschten Datei ergänzt werden.

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. Gib 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.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 TRIOS oder PLEXUS. 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:

CHDIR“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 CHDIR“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. speichere so oft wie möglich,um Deine Programmier-Fortschritte vor Verlust zu sichern.

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, 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.

Bsp.:

```
DIM #A(30,1),C(88),G(12,3)
```

Dieses Beispiel dimensioniert die Stringvariable #A und die beiden numerischen Variablen C und G.

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 Feldbereiche 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,tag*

tt=Tag mm=Monat yyyy=Jahr tag=Tageszahl (1-7)
Tag repräsentiert eine Zahl zwischen 1 und 7 für den jeweiligen Wochentag (z.Bsp. 1 für Montag und 7 für Sonntag)

Bsp.: SDATE 23,4,2013,2

Stellt das Datum 23.April 2013 ein und setzt den Wochentag 2 (Dienstag).
Als Trennung zwischen den Werten ist ein Komma, kein Punkt zu verwenden!

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 und 4 Counter programmierbar welche mit Werten gesetzt und ausgelesen werden können um Aktionen zu starten. Dabei zählen Timer 1-8 rückwärts und Timer 9-12 vorwärts.
3 Optionen sind verfügbar.

TIMER S*(ID,wert)*-setzt einen Timer mit der *ID* mit dem Zählwert *wert*, wobei *ID* die Nummer des Timers ist (1-12), 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*(1-12) .

Bsp.:

```
5 CROFF:CLS
10 TIMER S(1,500)
20 TIMER S(2,600)
30 TIMER S(3,200)
40 TIMER S(4,900)
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 1 dann 4, 5, 3 und zuletzt Timer 2 als abgelaufen notiert.

HINWEIS: Der Befehl TIMER C(*ID*) ist nur für Timer 1-8 sinnvoll , da hier auf Timerablauf (=0) abgefragt wird.

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.

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.

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*

Ab Version 2.3 erfolgt die Ausgabe des Programmlistings mit Syntaxhervorhebung . Dies hat den Vorteil, das ein Programm leichter zu lesen ist, da Befehle, Variablen, Zahlenwerte usw. jeweils in anderer Farbe dargestellt werden. Ich hätte es selbst kaum geglaubt, aber dieses Feature erleichtert die Fehlersuche erheblich und wertet TRIOS-Basic ungemein auf. Nun aber zur Funktionsweise.
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	bewegt den Cursor an das Zeilenende
Cursortaste runter	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) 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.
Der Editor wird solange ausgeführt, bis entweder die ESC-Taste gedrückt wurde oder die letzte Basiczeile erreicht ist.

Anzeige des freien Basic-Speichers

Syntax: FREE

Der Befehl FREE zeigt den noch verfügbaren Arbeitsspeicher von Trios-Basic in Bytes an.

Bsp.:PRINT FREE

Anzeige der aktuellen Trios-Basic-Version

Syntax : VER

Bsp.:PRINT VER

Es wird die Versionsnummer des verwendeten Trios-Basic zurückgegeben.

Neunummerierung von Basic-Zeilen

Syntax: RENUM *Startzeile,Endzeile,neue Startzeile,Schrittweite*

Neu ab Version 2.1 ist die Möglichkeit, Programmzeilen neu zu nummerieren. Oft wünscht man sich, zusätzliche Zeilen ins Programm aufzunehmen, nur ist kein Platz mehr zwischen zwei Zeilennummern vorhanden, um dies zu tun. Hier greift dieser Befehl. Mit ihm ist es möglich den Abstand zwischen zwei Programmzeilen-Nummern wieder zu vergrößern, oder ,falls gewünscht zu verkleinern. Das hat jedoch auch Grenzen. RENUM ändert nur die Zeilennummern, nicht jedoch die bei GOTO oder GOSUB angegebenen Sprungadressen. Diese müssen hinterher per

Hand angepasst werden!
Es gibt zwei Varianten der Handhabung.

1.RENUM ohne Parameter nummeriert das gesamte Basic-Programm beginnend mit der kleinsten Zeilennummer. Dabei wird Das Programm beginnend mit der Zeilennummer 10 und einer Schrittweite von 10 neu durchnummeriert.

2.RENUM mit Parameterübergabe, dabei haben die Parameter folgende Bedeutung

Startzeile	-Zeilennummer, bei der die Neunummerierung beginnen soll
Endzeile	-Zeilennummer, bis zu der neu nummeriert werden soll
neue Startzeile	-neue erste Zeilennummer
Schrittweite	-Abstand zwischen den neu nummerierten Zeilen

Bsp.:RENUM 10,80,5,5

Nummeriert das Programm ab Zeile 10 bis 80 beginnend mit 5 und einer Schrittweite von 5 neu.

Bsp.: RENUM

Nummeriert das gesamte Programm. Das Programm erhält als erste Zeilennummer die 10 und nummeriert alle folgenden Zeilen mit einer Schrittweite von 10 neu.

HINWEIS:
Sprungadressen von GOTO und GOSUB müssen per Hand angepasst werden!
Das Neunummerieren ändert nicht die Reihenfolge der Basiczeilen, also achte darauf, das die Zeilennummern in logischer Folge vergeben werden! (keine größeren vor kleineren Zeilennummern)

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,Typ*

PEEK ist ein Speicherzellen-Lesebefehl. Mit ihm kann der Inhalt von Speicherzellen im E-Ram gelesen werden. Der Wertetyp wird mit dem Parameter „Typ“ angegeben wobei 1=Byte, 2=Word und 4=Long bedeutet.

Bsp.:
```
10 a=PEEK $4000,1
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.

HINWEIS: Es werden nur Integerwerte gelesen

Speicherzellen schreiben

Syntax: POKE *Adresse,Wert,Typ*

POKE ist das Gegenstück zu PEEK und beschreibt die Speicherzelle an der *Adresse* Mit dem übergebenen *Wert*. *Typ* bestimmt, ob *Wert* als Byte, Word oder Long gespeichert wird (Byte=1,Word=2,Long=4).

Bsp.:
```
10 a=230344
20 POKE $80000,a,4
30 PRINT PEEK $80000,4
```

Im Beispiel wird an Adresse $80000 im E-Ram der Wert 230344 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 230344 erscheinen).

HINWEIS:Es dürfen nur Integerwerte in den Ram gepoked werden!

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:
Numerische oder String-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: MGET*(N)*

Mit diesem Befehl kann die Position der Maus in X- und Y Richtung sowie die Position des Scrollrades abgefragt werden. Dabei darf *N* einen Wert von 1 bis 3 annehmen.

Für N bedeuten die Werte:

1 X-Position
2 Y-Position
3 Z-Position (Scrollradposition)

Bsp.:
```
10 CROFF:CLS
20 MOUSE 1,1
30 x=MGET(1):y=MGET(2):z=MGET(3)
40 POS 0,0:PRINT x,y,z;“ „
50 GOTO 30
```

MGET(1) und MGET(2) liefern Rückgabewerte, die der Bildschirmposition in Form von Zeilen und Spaltennummern (x=0-39, y=0-29) entsprechen. MGET(3) 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.

Ausgabe von Numerischen Werten in Hexadezimaler oder Binärer Form

Es ist manchmal nützlich oder nötig Zahlenwerte auch in der hexadezimalen oder binären Form darzustellen. Um dies zu ermöglichen, wurden folgende zwei Befehle in TRIOS-Basic aufgenommen.

Darstellen von hexadezimalen Zahlen

Syntax:PRINT HEX*(wert)*

Bsp.:PRINT HEX(93)

Ausgabe:5D

Ausgeben eines einzelnen Zeichens an x-y-Position

Syntax:PUT*(Char,x,y)*

Bsp.:PUT(65,10,5)

Gibt das ASCII-Zeichen 65 (ein großes A) an der Position Spalte 10 Zeile 5 auf dem Bildschirm aus.

Darstellen von binären Zahlen

Syntax:PRINT BIN*(wert)*

Bsp.:PRINT BIN(93)

Ausgabe:01011101

HINWEIS:
Die Verwendung der Befehle HEX und BIN ist nur in Verbindung mit dem PRINT-Befehl erlaubt, da es sich um reine Ausgabebefehle handelt. Eine Werte- oder Variablenzuweisung ist nicht zulässig.
Zur Ausgabe von Variablenwerten sind diese Funktionen aber erlaubt und auch gedacht.

Bildschirm löschen

Syntax: CLS

CLS (Clear Screen) löscht den gesamten Bildschirm, des aktiven Fensters (Window) und setzt den Cursor (falls er nicht ausgeschalten wurde) an die linke obere Ecke.

Bsp.:
```
10 CLS
20 FOR I=1 TO 600
30 PRINT „A“;
40 NEXT I
50 CLS
```

Dieses kleine Beispiel füllt den Bildschirm mit dem Buchstaben A und löscht dann den Bildschirm.

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.

Die Window-Funktion

Die Window-Funktion hat ab Basic 2.1 die umfangreichste Veränderung erfahren. Alle bisherigen Befehle der Vorgängerversion zur Erstellung der verschiedenen Fenstertypen (MBOX,TWIN,TBAR)wurden entfernt und in die WIN-Funktion integriert. Die Optik erinnert stark an Plexus, kann aber vom Nutzer auch verändert werden, aber dazu später. Insgesamt sind 7 unabhängige Fenster gleichzeitig definierbar. Der WIN-Befehl hat verschiedene Command-Chars, welche folgend erläutert werden.

Ein Window erstellen

Syntax:

WIN C*(Nr, Vordergr, Hintergr, Cursor, x, y, xx, yy, typ, mode,"Titeltext")*

WIN C (CREATE) erstellt ein Window (Fenster) mit der Nummer *Nr* und den Farben *Vordergr,Hintergr,Cursor* an Position *x,y* bis *xx,yy* vom Typ *typ* mit *mode*=1 oder ohne Schatten *mode*=0 mit dem Titel *„Titeltext"*

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)
typ	Fenstertyp, momentan werden 8 verschiedene Fenstertypen unterstützt
mode	Rahmen 0=ohne Schatten 1=mit Schatten
Titeltext	Hier kann der Titeltext eines Fensters angegeben werden

Fenstertypen:

1	Fenster ohne Rahmen
2	Fenster mit Rahmen
3	Fenster ohne Rahmen, mit Titelzeile
4	Fenster mit Rahmen und Titelzeile
5	Fenster mit Rahmen und Scrollleiste rechts, ohne Titelzeile
6	Fenster mit Rahmen,Scrollleiste und Titelzeile
7	Fenster mit Rahmen,Titelleiste und Statusleiste
8	Fenster mit Rahmen, Titelleiste,Statusleiste und Scrollleiste

Einmal definierte Fenster sind solange in ihren Dimensionen gültig (auch wenn es nicht mehr sichtbar ist) bis es neu definiert oder durch WIN R(Nr) gelöscht wird.
Ein Umschalten der Fenster erfolgt mit dem Befehl WSET Nr (siehe dort).
Das Hauptfenster hat immer die Nummer 0 und kann nicht geändert werden.

Bsp.:
```
10 CLS
20 WIN C(1,0,30,0,5,5,30,25,1,1,“„)
```

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

Den Fenstertitel setzen oder ändern

SYNTAX: WIN T*(Nr,“Titeltext“)*

Soll der Titel eines Fensters geändert oder erst später gesetzt werden, ist dieser Befehl zu verwenden. Er hat natürlich nur auf Fenstertypen eine Wirkung, die auch eine Titelzeile besitzen (Typ 3,4,6,7,8).

Bsp.:
```
10 CLS
20 WIN C(1,0,30,0,5,5,30,25,4,1,“„)
30 WIN T(1,“Mein Fenster“)
```

Erstellt ein Fenster mit hellblauem Hintergrund an Position 5,5 bis 30,25 mit einem Schatten und setzt den Titeltext „*Mein Fenster*“.

Statustext eines Fensters setzen oder ändern

SYNTAX: WIN S*(Nr,“Statustext“)*

Wurde als Fenstertyp 7 oder 8 gewählt, so ist es zusätzlich möglich, einen Statustext zu setzen bzw. diesen zu ändern.

Bsp.:
```
10 CLS
20 WIN C(1,0,30,0,5,5,30,25,7,1,“„)
30 WIN T(1,“Mein Fenster“)
40 WIN S(1,“Mein Statustext“)
```

Erstellt ein Fenster mit hellblauem Hintergrund an Position 5,5 bis 30,25 mit einem Schatten und setzt den Titeltext „*Mein Fenster*“ sowie den Statustext „*Mein Statustext*“.

Fensterparameter löschen und Fenster-Tiles ändern

SYNTAX: WIN R*(Nr)*

Mit diesem Befehl werden die Parameter eines gesetzten Fensters gelöscht. Dabei bleibt die Optik des Fensters aber erhalten, nur die Dimensionswerte werden gelöscht.
Der eigentliche Grund dieses Befehls ist, die Übernahme neuer Tiles zur optischen Veränderung des Fensters. Wem das Aussehen nicht gefällt oder für sein Programm ein anderes Aussehen der Fenster wünscht, kann mit eigenen Tile-Dateien die Fensteroptik individuell gestalten. Zu diesem Zweck sind im E-Ram ab Adresse $7E500 die Nummern der, für die Fenstergestaltung wichtigen Tilenummer abgelegt. Diese können mit der Poke-Anweisung verändert (angepasst werden).
Insgesamt gibt es 17 Speicherplätze für Fenster-Tiles mit folgender Bedeutung:

Adresse	**Bedeutung**
$7E500	linke obere Ecke Rand
$7E501	oberer Rand
$7E502	rechte obere Ecke Rand
$7E503	rechter Rand
$7E504	rechte untere Ecke Rand
$7E505	unterer Rand
$7E506	linke untere Ecke Rand
$7E507	linker Rand
$7E508	linke obere Ecke Titelzeile
$7E509	Titelzeile
$7E50A	rechte obere Ecke Titelzeile
$7E50B	rechter oberer Pfeil Scrollleiste
$7E50C	rechte Scrolleiste
$7E50D	rechter unterer Pfeil Scrollleiste
$7E50E	Statusleiste
$7E50F	linke untere Ecke Statusleiste
$7E510	rechte untere Ecke Statusleiste

Lade und starte zur Veranschaulichung das Programm „Window.Bas“. Dort werden alle Fenstertypen und verschiedene Möglichkeiten, die Optik der Fenster zu beeinflussen demonstriert. Wichtig ist, das die geänderten Tilenummern erst mit dem Befehl WIN R(Nr) übernommen werden. Ob dazu eine eigene Tile-Datei benutzt wird, ist Dir überlassen. Diese muss aber vor der ersten Verwendung geladen und mit dem STILE-Befehl aktiviert werden.

Willst Du wieder zum System-Tileset zurück wechseln, reicht der Befehl:STILE 15, alle Fensteraufrufe erfolgen wieder mit dem System-Tileset.

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 C(1,0,30,0,0,0,39,10,4,1,“„)
30 WIN C(2,0,40,0,0,11,39,19,5,0,“„)
40 WIN C(3,0,50,0,0,20,39,29,6,1,“„)
50 WSET b:PRINT“Ich bin hier!“
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

Text-Button

Syntax: BUTTON T(*ID,VFarbe,HFarbe,x,y,“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 dem PC.
Folgende Parameter werden dazu benötigt.

ID - Nummer des Buttons (Diese Nummer wird beim Anklicken mit der Maus zurückgegeben und identifiziert den gewählten Button.
VFarbe-bestimmt die Text-Farbe des Buttons
HFarbe-gibt die Button-Farbe an

x,y -Position des Buttons auf dem Bildschirm(x=0-39, y=0-29)

Bsp.:

```
10 CLS:CROFF:MOUSE 1,1
20 BUTTON T(1,30,10,10,10,“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 hellblauer 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 Icon-Button erstellen

Syntax: BUTTON I*(ID,Tile-Nr,Farbe1,Farbe2,Farbe3,X,Y)*

Mit diesem Befehl ist es möglich, einzelne Tiles einer Font-bzw. Tiledatei als sogenannte Icon-Buttons zu verwenden. Folgende Parameter werden benötigt.

ID	Nummer des Buttons
Tile-Nr	Nummer des Tiles aus der Tiledatei, das als Button dienen soll
Farbe1-3	die drei Farben für die Tile-Darstellung
X,Y	X-Y-Position, an der der Button erscheinen soll (X=0-39, Y=0-29)

Bsp.:

```
10 CLS:CROFF:MOUSE 1,1
20 BUTTON I(1,128,$E6,0,250,10,10)
30 a=MB 0
40 IF a=1 THEN PRINT a;
50 GOTO 30
```

Lade und starte zur Demonstration die Datei „Button2.Bas“

Befehlsknopf löschen

Syntax: BUTTON R *(ID)*

BUTTON R(ID) löscht einen zuvor erstellten Button mit der Nummer *ID* vom Bildschirm.

Bsp.: BUTTON R(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 oder Rechteck 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) oder 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 , Playerposition abfragen

Syntax: GETX *(option)*

Gibt als Funktionswert die Spaltenposition des Cursors (1) oder der Playerfigur (2) zurück.

Syntax: GETY *(option)*

Gibt als Funktionswert die Zeilenposition des Cursors (1) oder der Playerfigur (2) zurück.

Bsp.:

```
10 CROFF:CLS:POS 0,2
20 a=INKEY:IF a=0 THEN GOTO 20
30 PRINT CHR$(a);
40 x=GETX(1)
50 y=GETY(1)
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.

Bsp.:`x=GETX(2)`

Gibt die Position der Player-Figur in X-Richtung zurück (siehe Tile-Befehle, Player-Settings)

Bildschirmbereiche sichern und wiederherstellen

Bisher war die Darstellung von Windows zwar kein Problem, aber wenn man es wieder vom Bildschirm verschwinden lassen wollte, ohne den Gesamteindruck des Bildschirms zu zerstören, war man schon etwas herausgefordert. Entweder löschte man den gesamten Bildschirm und baute ihn wieder ohne Window auf, was zeitraubend und programmiertechnisch aufwendig war oder man löschte nur den Bereich, den das Fenster belegte und stellte nur diesen wieder her, was noch aufwendiger war. Das ist nun nicht mehr nötig, da Trios-Basic 2 über die Möglichkeit verfügt, den Bildschirminhalt, welcher sich hinter einem Fenster befindet, zu sichern und nach dem Entfernen des Fensters, genauso wieder herzustellen, wie er vor dem Erscheinen des Fensters war. Wie geht das?
Mit nur zwei Befehlen ist es möglich, Bereiche des Bildschirms (oder den gesamten Bildschirm) im E-Ram zu speichern und wieder herzustellen. Sehen wir uns das genauer an.

Syntax: BACKUP *x,y,xx,yy,Ram-Adr*

Wie der Name des Befehls schon vermuten läßt, wird mit BACKUP der gewünschte Bildschirmbereich gesichert. Dazu ist die Angabe folgender Parameter nötig:

x,y,xx,yy dies sind die Koordinaten des Bildschirmbereiches, der gesichert werden soll.

Ram-Adr Hier muss die Adresse eines freien Ram-Bereiches stehen.

An dieser Stelle auch gleich eine Warnung vorweg. Ram-Adr. muss unbedingt ein freier Bereich im E-Ram sein. Benutze keine Bereiche, welche vom Basic verwendet werden, da unvorhersehbare Reaktionen auftreten können und der Programmablauf gestört wird. Zu diesem Zweck wurde am Ende des E-Ram-Bereiches ein ca.70kB großer, für den Nutzer freien Bereich eingerichtet. Dieser beginnt bei Adresse Hex-$EE800 oder Dez-976896 und reicht bis Hex-$FFEFF oder Dez-1048319 also genau 71423 Bytes. Da ein kompletter Bildschirm 7200 Bytes belegen würde, passen also locker 9 komplette Bildschirmseiten in diesen Bereich (siehe dazu auch die Ram-Aufteilung am Ende des Handbuches). **Benutze bitte ausschliesslich diesen Bereich für die BACKUP-Funktion.**
Aber genug der Vorrede, jetzt wollen wir die Funktion mal ausprobieren. Gib das folgende kleine Programm ein.

Bsp.:
```
10 CLS:CROFF
20 FOR I=0 TO 12
30 PRINT STRING$(39,“X“)
40 NEXT I
50 BACKUP 5,5,30,25,$EE800:Pause 1000
60 WIN C(1,0,33,250,5,5,29,24,4,1,“Testfenster“)
```

Wenn Du das Programm startest, wirst Du sicher enttäuscht sein, da lediglich der Bildschirm mit dem Buchstaben X gefüllt wird (Zeile 20-40). In Zeile 50 wird der Bildschirmbereich von 5,5 bis 30,25 in den E-Ram ab Adresse $EE800 geschrieben. In Zeile 60 erstellen wir über den gesicherten Bereich ein Window. Die Ausdehnung in X und Y-Richtung ist jeweils um eine Position kleiner, da wir unser Fenster mit einem Schatten darstellen (immer dran denken, Schatten vergrößern den Fensterbereich um eine Position in X-und Y-Richtung).
Bisher haben wir noch nichts spektakuläres gesehen, aber jetzt wird es konkret.
Syntax: RECOVER *x,y,xx,yy,Ram-Adr*

Dieser Befehl befördert unseren gesicherten Bildschirmbereich wieder auf den Schirm. Schreiben wir unser Programm weiter...

```
70 Pause 6000
80 RECOVER 5,5,30,25,$EE800
90 WSET 0
100 WIN R(1)
```

Wenn due keine Tippfehler oder Ähnliches gemacht hast, sollte das Fenster verschwunden und der Bildschirm wieder mit X gefüllt sein.

Zeile 90 springt in Fenster 0 (Hauptfenster) und Zeile 100 löscht die Fensterparameter von Fenster 1.

HINWEIS:

-Benutze für diese Funktion nur freie Bereiche de E-Ram ($EE800-$FFEFF ca.70Kb)
-ein kompletter Bildschirm ist 7200 Bytes groß, beachte dies bei der Speicherzuweisung, um ein Überschreiben schon gesicherter Bildschirmbereiche zu vermeiden
-Fenster, welche mit Schatten erstellt wurden sind in X-und Y-Richtung je eine Position größer
-um die Größe des gewählten Bereiches zu errechnen verwende folgende Formel:
(XX-X)*(YY-Y)*6=Speichergröße in Bytes

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

Syntax: STR$ *K(String)*

String wird in Kleinbuchstaben konvertiert.

#a="Dies ist der erste String"
PRINT STR$ K(#a)
Ausgabe:
dies ist der erste string

Syntax: STR$ *G(String)*

Konvertiert *String* in Großbuchstaben.

#a="Dies ist der erste String"
PRINT STR$ G(#a)
Ausgabe:
DIES IST DER ERSTE 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.Dies funktioniert allerdings nur mit der Übergabe eines Variableninhaltes.

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$(3,#a)
```

oder

```
PRINT STRING$(3,"Hive-Computer ")
```

Ausgabe:

Hive-Computer Hive-Computer Hive-Computer

Die Ausgabelänge beträgt maximal 40 Zeichen !

Die Zeichenkette „Hive-Computer „ wird 3 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 Hive

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 größer als 33 Zeichen ist 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.**

Suche einer Zeichenkette in einer Zeichenkette

Syntax: INSTR*(Zeichenkette1,Zeichenkette2)*

Mit dieser Funktion wird überprüft, ob Zeichenkette1 vollständig in Zeichenkette2 vorhanden ist und gibt die Position des ersten Zeichens (von Zeichenkette1) bezüglich ihres Auftretens in Zeichenkette2 zurück. Ist Zeichenkette1 nicht in Zeichenkette2 enthalten, wird der Wert 0 zurückgegeben.

Bsp.:

```
#a="Hive"
PRINT INSTR(#a,"Ich programmiere meinen Hive")
25
```

oder

```
#a="Hive"
#b="Alles neu macht der Hive"
PRINT INSTR(#a,#b)
21
```

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

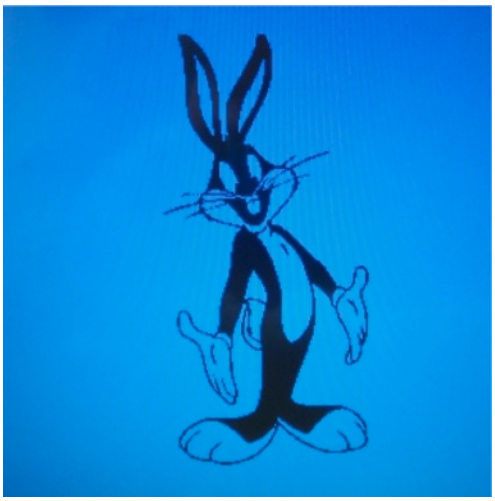
x,y - Position auf dem Bildschirm

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



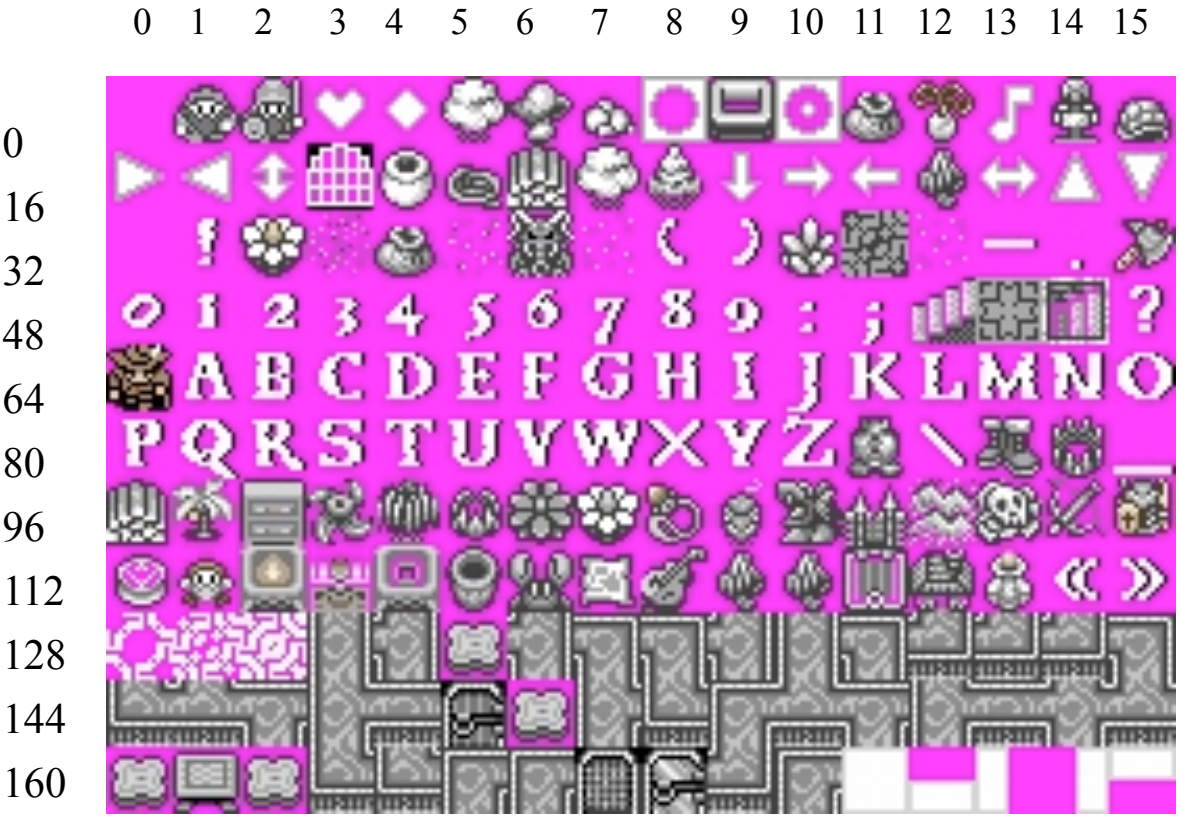
Einzelnes Tile auf dem Bildschirm ausgeben

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
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

MAP w fordert den Interpreter auf, alle nun folgenden Tile-Befehle in den E-RAM zu schreiben. 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.

HINWEIS:

Der MAP-RAM dient zugleich als Bildschirm-Schattenspeicher, das heisst alle Änderungen der Tiledarstellung sollten mit MAP w auch in den Ram geschrieben werden. Das ist Vorraussetzung, damit, wie in späteren Kapiteln (Sprites und Player-Befehle) erläutert wird, die entsprechenden Abfragen richtig funktionieren).

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.

MAP W wird nur durch MAP D deaktiviert!!!

MAP-Speicher löschen

Syntax: MAP C

Der Map-Speicher dient nicht nur zur Ablage von Map-Dateien, sondern fungiert auch als quasi-Bildschirm-Kopie um Sprite-Abfragen in Spielen zu ermöglichen.

Das bedeutet, das bei Verwendung von bewegten Tiles (Sprites-, Player- oder Item-Tiles) ständig Veränderungen in den Map-Speicher geschrieben werden, welche intern für diverse Abfragen benutzt werden. Wird also z.Bsp. ein Level eines Spieles wiederholt oder ein neues Level (Map) geladen, sollte der Map-Speicher zuerst mit dem Befehl MAP C gelöscht werden. So wird garantiert, das das geladene Level (Map) keine „Rest-Tiles“ aus voran gegangenen Aktionen beinhaltet, da die Veränderungen im Speicher nicht unbedingt sichtbar sein müssen.

HINWEIS:

Verwende beim Wechsel von Map-Daten den Befehl MAP-C, um interne Veränderungen im Map-Speicher zu eliminieren!

Player-Parameter (Spielfigur-Settings)

Syntax: PLAYER <*option*>

Um Spiele zu programmieren, in denen auch etwas Bewegung ins Spiel kommen soll sind entsprechende Spielfiguren-Settings erforderlich. Ich muss dazusagen, das sich dieser Teil noch im Anfangsstadium befindet und sicherlich zur Zeit nicht alle Wünsche, die man an Spielfiguren stellen würde, erfüllt werden.

Zum erstellen kleiner Spiele, sollte der Funktionsumfang aber ausreichen.

PLAYER ist wieder ein Befehl mit mehreren Optionen. Ohne weiteren Parameter, fragt er das Kollisionsflag in Bellatrix ab. Das heisst, kollidiert Deine Spielfigur mit einem Sprite, wird in Bellatrix das Kollisionsflag auf 1 gesetzt.

Bsp.: a=PLAYER(1)

Fragt das Kollisionsflag ab.

Rückgabewerte:

0=keine Kollision, 1=Kollision mit Sprite, 2=Kollision mit Kollisionstile

Diese Abfrage ist aber nur sinnvoll, wenn auch eine Spielfigur und Sprites gesetzt und in Bewegung gebracht wurden. Schauen wir uns das genauer an.

Folgende Optionen existieren für den Befehl PLAYER:

P-Parameter der Spielfigur

K-Spieltasten definieren

I -Item-Tiles definieren

B-Block-Tiles definieren

C-Kollisions-Tiles definieren

E-Ersatz-Item-Tiles definieren

Dies sind sehr viele Optionen, aber keine Bange, jeder Befehl wird genau erklärt. Also fangen wir mit der ersten Option an.

Syntax: PLAYER P*(Tnr,Farbe1,Farbe2,Farbe3,X,Y)*

PLAYER P (P wie Parameter) setzt die Spielfigur mit folgenden Einstellungen

Tnr =Nummer des Grafiktiles, das den Spieler darstellen soll

Farbe1 =Farbe1 des Player-Tiles

Farbe2 =Farbe2 des Player-Tiles

Farbe3 =Farbe3 des Player-Tiles

X =Startposition in X-Richtung

Y =Startposition in Y-Richtung

Bsp.:

```
10 CLS
20 TLOAD 1,“Font1.dat“,16,11
30 STILE 1
40 PLAYER P(2,0,255,$56,10,10)
```

Dieses Beispiel setzt das Tile Nummer 2 der Tiledatei „Font1.dat“ als Player auf den Bildschirm. Dies ist ähnlich dem Tile-Befehl und noch nicht besonders spektakulär, da ja noch keine Sprites geschweige Bewegung initiiert wurde. Machen wir also weiter.

Spielertasten definieren

Syntax: PLAYER K *(links,rechts,hoch,runter,feuer)*

Standardmässig sind die Steuertasten zum Spielen auf die Cursortasten und die Space-Taste als Feuer-Taste belegt. Diese Belegung kann mit diesem Befehl aber individuell angepasst werden.

Bsp.: `PLAYER K(2,3,4,5,32)`

Im Beispiel ist die Standard-Tastenbelegung definiert worden, kann aber auch mit

anderen Werten gefüllt werden, die dem ASCII-Code der gewünschten Taste entsprechen muss.

HINWEIS:

Denk daran, das die Tastatur im Normalfall auf Kleinschreibung eingestellt ist, die ASCII-Werte also den Kleinbuchstaben entsprechen muss, um Wirkung zu erzielen.

Definieren von einzusammelnden Gegenständen

In vielen Spielen ist es das Ziel, bestimmte Gegenstände einzusammeln, um das Level zu vollenden. Dafür existiert die folgende Option.

Syntax: PLAYER I *(i1,i2,i3,i4,i5,i6)*

Die mit i und einer Nummer bezeichneten Werte stellen Tile-Nummern der Tiles dar, die eingesammelt werden sollen. Gehe zum Bsp. zurück zum Tile-Befehl , dort suchst Du Dir , die im Spiel einzusammelnden Tiles aus der Tile-Datei heraus und gib sie ,wie im folgenden Beispiel ein.

Bsp.:PLAYER I(98,115,47,42,103,0)

Im Beispiel haben wir 4 Tiles benannt die im Spiel als Objekte, die eingesammelt werden sollen, definiert werden. Bis zu 6 Gegenstände können definiert werden. Sollte das nicht reichen, kann zu jeder Zeit (zum Bsp. wenn die ersten 5 Gegenstände eingesammelt wurden) neu definiert werden. Werden nicht alle 6 Einträge gebraucht, sollten sie mit Tile-Nummern belegt werden, die nicht auf dem Bildschirm zu sehen sind um die Auswertung nicht durcheinander zu bringen.

Abgefragt , ob ein Item eingesammelt wurde, wird mit

a=PLAYER(2)

Rückgabewerte:

0=Kein Item 1-6=Nummer des eingesammelten Items.

Die Rückgabe des Item-Wertes erfolgt in dem Moment, wo die Spielfigur auf dem entsprechendem Item-Tile steht also an den x,y-Koordinaten des Item-Tiles.Alle definierten Tiles müssen sich auch auf dem Spielfeld befinden, um eingesammelt werden zu können. Dies kannst Du mit dem Tile-Befehl erledigen.

Blocktiles definieren um den Spielerweg einzugrenzen

SYNTAX: PLAYER B*(b1,b2,b3,b4,b5,b6,b7,b8,b9,b10)*

Da sich die Spielfigur nicht wahllos auf dem Bildschirm bewegen lassen soll, (sie soll sich ja innerhalb des Spielfeldes, und nur dort bewegen) müssen wir Tiles definieren, die nicht begangen werden dürfen. Das geht folgendermaßen.

Bsp.:PLAYER B (146,131,114,138,167,165,166,157,148,155)

Wenn Du die Werte mit diesem Bild vergleichst, wirst Du verstehen, das wir jetzt die Mauern des Spielfeldes als nicht begehbar definiert haben, unsere Spielerfigur sich also nicht über diese hinweg bewegen kann.

Als letzte Option, gibt es noch die Möglichkeit, Tiles zu definieren, dessen Betreten als Kollision ausgewertet wird. Damit ist nicht die Sprite-Kollision gemeint (diese muss nicht definiert werden, da sie automatisch vorhanden ist), nein, hier können zum Beispiel Bereiche auf dem Spielfeld als „tödlich“ markiert werden.

Kollisions-Tiles definieren

Syntax: PLAYER C*(c1,c2,c3,c4,c5,c6)*

c1-c6 sind wieder Tile-Nummern, der Tiles die das Kollisions-Flag auslösen, wenn sie betreten werden. Dies kann benutzt werden, um auf dem Spielfeld z.Bsp.Fallen aufzustellen. Abgefragt wird dieses Kollisions-Flag ,wir erinnern uns, mit

a=PLAYER(1)

Rückgabewerte:

0=keine Kollision, 1=Kollision mit Sprite, 2=Kollision mit Kollisions-Tile

Tiles definieren, welche die eingesammelten Gegenstände auf dem Spielfeld ersetzen sollen

Syntax: PLAYER E*(Nr,Tnr,F1,F2,F3)*

Wir haben gelernt, wie Tiles definiert werden, die als Gegenstände in einem Spiel eingesammelt werden können und durch ihren Rückgabewert bestimmte Reaktionen auslösen können. Nun ist es manchmal wünschenswert, diese Gegenstände, nach dem sie eingesammelt wurden, vom Bildschirm verschwinden zu lassen. Dies wird mit diesem Befehl bewerkstelligt. Um ein Tile vom Bildschirm verschwinden zu lassen, muss das Ersatz-Tile die gleichen Eigenschaften wie die Tiles, welche den Hintergrund bilden besitzen. Sollte also unser Hintergrund mit dem Tile-Nr:123 und den Farbattributen F1=0 F2=$80 F=33 erstellt worden sein, muss unser Ersatzitem natürlich ebenso aussehen. An welcher Stelle auf dem Spielfeld dieses Ersatzitem auftaucht, ermittelt der Interpreter anhand der Item-Nr des eingesammelten Items. Insgesamt können bis zu sechs Ersatztiles definiert werden. Dabei ist eine dynamische Neuzuweisung nach jedem erfolgreich eingesammelten Gegenstand möglich.

Nr -Nummer des Ersatzitems (entsprechend PLAYER I)

Tnr -Tile-Nummer aus der Tiledatei, welches das Gegenstandstile ersetzen soll

F1..3 -Farbattribute des Ersatz-Tiles

Bsp.:

```
PLAYER E(1,123,0,$80,33)
PLAYER E(2,123,0,$80,33)
... PLAYER E(6,34,0,$80,33)
```

HINWEIS:

-es sind bis zu 6 Ersatz-Tiles definierbar

-Ersatz-Tiles sind dynamisch zuweisbar, ist ein Item eingesammelt, kann Item und Ersatzitem neu definiert werden

-um die Ersetzen-Funktion zu aktivieren ist mit PLAYER(2) auf Items zu scannen

Bsp: `a=PLAYER(2)`

-die Wirkungsweise wird mit dem Beispielprogramm“Level.Bas“ oder „Joystick.Bas“ demonstriert.

Spielfigur bewegen

Syntax: PLAYXY *wert*

Nun wollen wir unsere Spielfigur mit den Steuertasten über den Bildschirm bewegen. Dazu gibt es diesen Befehl. *Wert* entspricht dem Steuertastenwert.

Erweitern wir nun unser vorheriges Beispiel mit diesem Befehl.

Bsp.:

```
10 CLS
20 TLOAD 1,“Font1.dat“,16,11
30 STILE 1
35 MAP L,“Map.dat“
40 MAP D
50 PLAYER P(2,0,255,$56,10,10)
60 PLAYER I(98,115,47,42,103,0)
80 PLAYER B (146,131,114,138,167,165,166,157,148,155)
90 a=INKEY
100 PLAYXY a
110 GOTO 90
```

In diesem Beispiel ist schon etwas Bewegung, aber wir wollen mehr. Jetzt sollen ein paar Sprites hinzukommen.

Sprite-Parameter definieren

Syntax: SPRITE <*option*>

Folgende Optionen sind für Sprites verfügbar:

P - Spriteparameter setzen

M -Sprite-Bewegung an/aus und Reset 1,0,2

S -Sprite-Geschwindigkeit 1-255

Syntax: SPRITE P*(Nr,Tnr,Tnr2,F1,F2,F3,Dir,start,end,x,y)*

Nr	- Spritenummer 1-8
Tnr	- Tile-Nummer die den Sprite darstellen soll
Tnr2	- 2. Tile-Nummer für den Sprite für Bewegungsanimation
F1-F3	- Farben 1-3
Dir	- Richtung 1=links,2=rechts,3=hoch,4=runter,5=Verfolgung
start	- Richtung-Startposition
end	- Richtung-Endposition
x,y	- Start-Position x,y

Die Parameter *Nr,Tnr* und *F1-F3* sollten klar sein. Dir legt fest, ob sich die Sprite-Figur von links nach rechts, rechts nach links, von oben nach unten oder von unten nach oben bewegen soll oder ob sie die Spielerfigur verfolgt.

Mit *Tnr2* kann ein zweites Spritetile definiert werden, um dem Sprite eine kleine Eigenanimation zu verpassen, damit es in der Bewegung nicht zu statisch wirkt.

Start legt die Startposition in x oder y Richtung fest, je nachdem welcher Richtungsparameter angegeben wurde. *End* entspricht der Endposition der Spritebewegung (in X oder Y-Richtung). Wird diese erreicht, kehrt die Sprite-Figur in die Gegenrichtung um. **Beim Verfolgermodus, muss sichergestellt werden, das das verfolgende Sprite nicht mit anderen Sprites ins Gehege kommt, da es sonst zu Grafikfehlern kommen kann.** *X,Y* ist die Startposition des Sprites und sollte mit den *Start* und *End*-Parametern im Einklang stehen.

Nun soll sich unser Sprite bewegen.

Sprite-Bewegung an/aus, reset

SYNTAX: SPRITE M*(parameter)*

Als Parameter werden folgende unterschieden

0-Spritebewegung anhalten

1-Spritebewegung starten

2-Sprites resetten und von Bildschirm löschen

Sprite-Geschwindigkeit setzen

Syntax: SPRITE S*(wert)*

Wert darf im Bereich von 0 (Stillstand) bis 255 liegen. Kleine Werte bedeuten schnellere Bewegung, höhere Werte langsame Bewegung. Werte zwischen 50 und 100 sind am praktikabelsten.

Nun werten wir unser Programm mit diesen Parametern auf und sehen was geschieht.

Bsp.:

```
10 CLS
20 TLOAD 1,“Font1.dat“,16,11
30 STILE 1
35 MAP L,“Map.dat“
40 MAP D
50 PLAYER P(2,0,255,$56,10,10)
60 PLAYER I(98,115,47,42,103,0)
80 PLAYER B (146,131,114,138,167,165,166,157,148,155)
90 SPRITE P(1,15,14,$56,250,0,2,17,24,17,18)
100 SPRITE P(2,15,14,$56,255,0,3,13,18,7,15)
110 SPRITE P(3,15,14,$56,255,0,4,18,22,10,18)
120 SPRITE S(80)
130 SPRITE M(1)
140 a=INKEY
150 PLAYXY a
160 b=PLAYER(1)
170 IF b>0 THEN GOTO 200
180 GOTO 140
200 REM KOLLISION MIT SPRITE
210 FOR I=1 TO 20
220 BEEP 90:PAUSE 10:BEEP 100
230 NEXT I
```

```
240 SPRITE M(0)
250 POS 8,15:PRINT „DU HAST VERLOREN“
260 PAUSE 2000
270 SPRITE M(2)
280 GOTO 40
```

Herzlichen Glückwunsch, Du hast Dein erstes kleines Spiel auf dem Hive programmiert. Jetzt kannst Du noch Items, die eingesammelt werden müssen, einfügen oder Dein Spiel mit weiteren Sprites füttern, eigene Level-Maps erstellen usw,usw...

Siehe dazu auch die Beispiele auf der SD-Karte „Level.Bas“, “Joysick.Bas“, “Climber2.Bas“ oder „Clim2.Bas“.

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.

Der gültige Wertebereich von TRIOS-Basic liegt zwischen 1e+38 und 1e-38. Bei der Eingabe ist zu beachten, das Zahlenwerte größer *999999* oder kleiner *-999999* in Exponentialschreibweise eingegeben werden müssen Beispiel :

für 12340000 ist 1.234e+7 einzugeben. Wird dies nicht berücksichtigt, kommt es zu Konvertierungs- und demzufolge Rechenfehlern!!!*(Beschränkung der entsprechenden Routinen)*

Zu den grundlegenden Rechenoperationen gehören die vier Grundrechenarten + - * /.

Die Genauigkeit der Ergebnisse stellt das zur Zeit machbare dar!

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

Bsp.: PRINT 5.3*7.8+3.2 -> Ausgabe 44.54

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

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.

PREC(Wert)	Einstellen der Nachkomma-Genauigkeit (1-7)
SIN(X)	Sinus (X im Bogenmaß)
COS(X)	Cosinus (X im Bogenmaß)
TAN(X)	Tangens (X im Bogenmaß)
ASIN(X)	Arcussinus
ACOS(X)	Arcuscosinus
ATAN(X)	Arcustangens
LN(X)	natürlicher Logarithmus (X>0)
E(X)	Exponentialfunktion e^X
INT(X)	nächst kleinerer Integerwert von X
LOG(X)	Logarithmus zur Basis 10
SGN(X)	Signumfunktion x>0=1 x=0->0 x<0=-1
SHL(X1,X2)	X1 wird um X2 fach Bitweise nach links geshiftet
SHR(X1,X2)	X1 wird um X2 fach Bitweise nach rechts geshiftet
REV(X1,X2)	X2 Stellen von X1 werden invertiert

ODER(X1,X2)	Bitweise Oder-Verknüpfung von X1 und X2
XODER(X1,X2)	Bitweise XOder-Verknüpfung von X1 und X2
UND(X1,X2)	Bitweise Und-Verknüpfung von X1 und X2
DEG(X)	Rückgabe des Winkels X in Grad
RAD(X)	Rückgabe des Winkel X im Bogenmaß

Bsp.:

PRINT FN SIN(45) -> 0.850902

PRINT FN COS(45) -> 0.525324

PRINT FN TAN(45) -> 1.61977

PRINT FN ASIN(0.707) -> 0.785247

PRINT FN ACOS(0.707) -> 0.785549

PRINT FN ATAN(0.707) -> 0.615409

PRINT FN LN(2.45) -> 0.896088

PRINT FN E(1.2) -> 3.32012

PRINT FN INT(56.34) -> 56

PRINT FN LOG(12.3) -> 1.08991

FN PREC(3) -> Werteausgabe erfolgt nun mit maximal 3 Nachkommastellen

-> PRINT 3/77 -> 0.039

FN PREC(6) -> Werteausgabe mit bis zu 6 Nachkommastellen

-> PRINT 3/77 -> 0.038961

PRINT FN SGN(8.1) ->1 FN SGN(0) ->0 FN SGN(-12.4) -> -1

PRINT FN SHL(4,1) ->8

PRINT FN SHR(4,1) ->2

PRINT FN REV(111,6) ->61

PRINT FN ODER(8,2) ->10

PRINT FN XODER(55,12) ->59

PRINT FN UND($B8,$C5) ->128

PRINT FN SIN(FN DEG(30)) ->0.5

PRINT FN RAD(0.5) ->28.6478

BIT-Operationen

Die folgenden Bitoperationen sind nur in Verbindung mit numerischen Werten erlaubt. Sie sind nicht zu verwechseln mit den Logischen Operatoren!

Bitmuster nach links,rechts schieben „Shift-left“, „Shift-right“

Syntax:FN SHL*(wert,zahl)* , **FN SHR***(wert,zahl)*

Mit dieser Funktion wird das dem numerischen Ausdruck „wert“ entsprechende Bitmuster um die mit „zahl“ angegebene Anzahl oft nach links, rechts geschoben.

Bsp.:PRINT FN SHL(5,2)

Ausgabe: 20

Sehen wir uns das genauer an. Die Zahl 5 entspricht dem Bitmuster 0101.

Wir erweitern (in Gedanken) das Bitmuster um 4 Stellen also:0000 0101. Dieses Muster wird um 2 Stellen nach links geschoben, ergibt also 0001 0100 oder dezimal 20.

Überprüfen können wir diese Behauptung mit der Funktion

PRINT BIN(20) ->Ausgabe: 00010100

Schieben wir nun wieder zurück mit PRINT FN SHR(20,2) und die Ausgabe lautet nun wieder 5

Bit-Revers-Funktion

Syntax:FN REV*(wert,bit's)*

Die Bitweise Revers-Operation gibt die angegebene Anzahl Bits in umgekehrter Anordnung zurück. Dies schaut folgendermaßen aus.

Unser Start-Bitmuster sieht so aus: 01011101 dies entspricht der Zahl 93

Bsp.:PRINT FN REV(93,8)

Mit obiger Funktion werden alle 8 Bits revers (in umgekehrter Reihenfolge) zurückgegeben.

Ausgabe:186

Wir überprüfen das Bitmuster wieder mit PRINT BIN(186)

Ausgabe: 10111010 zum Vergleich noch einmal unser Ausgangs-Bitmuster

---------->01011101

Wie zu sehen ist, wurde das Bitmuster komplett umgekehrt.

Bit-Oder-Funktion

Syntax:FN ODER*(wert1,wert2)*

Mit dieser Funktion werden zwei Werte ODER verknüpft. Was heisst das?

Sehen wir uns das wieder am Beispiel an.

Bsp.: PRINT FN ODER(3,28)

Ausgabe:31

Zum Vergleich die Bitmuster

3 =00000011

28=00011100

Wir sehen, das bei der Oder-Verknüpfung, alle Bits mit dem Wert 1 in das Ergebnis übernommen werden.

Wir überprüfen das Ergebnis wieder mit PRINT BIN(31)

Ausgabe:00011111

Bit-XOder-Funktion

Syntax:FN XODER*(wert1,wert2)*

Im Gegensatz zur Oder-Verknüpfung, werden bei der XOder-Verknüpfung gleiche Bits mit dem Wert 1 zu einer Null

Bsp.:PRINT FN XODER(12,28)

Ausgabe:16

Unsere Bitmuster:

12=00001100

28=00011100

und das Ergebnis:

16=00010000

Bit-Und-Funktion

Syntax:FN UND*(wert1,wert2)*

Bei der Und-Verknüpfung, ist das Ergebnisbit nur dann 1 wenn beide Ausgangsbits 1 sind.

Bsp.: PRINT FN UND(12,28)

Ausgabe:12

Unsere Bitmuster:

12=00001100

28=00011100

und das Ergebnis:

12=00001100

Erzeugung von Zufallszahlen

Syntax: RND*(Wert)*

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

1.Bsp.: PRINT RND(15) - erzeugt eine Zufallszahl zwischen 0.000 und 15.000

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) .

2.Bsp.:PRINT FN INT(RND(15)) -> Ausgabe eines zufälligen Integerwertes.

Ebenso kann auch folgende Möglichkeit benutzt werden.

Bsp.: PRINT RND(1)*15 -> hat die gleiche Wirkung, wie im ersten Beispiel

oder

Bsp.: PRINT FN INT(RND(1)*15) -> gleiche Wirkung, wie Beispiel 2

Benutzerdefinierte Funktionen mit dem Befehl FUNC

SYNTAX: FUNC *var(op1,op2..op4)=[mathematische Formel]*

var	= Name der Funktion, hier darf ein Buchstabe A-Z stehen, dieser hat aber nichts mit der gleichnamigen Variablen A-Z zu tun.
op1	=erste zu übergebene Variable(A-Z), **diese ist unabdingbar, auch wenn sie in der Formel nicht verwendet wird**
op2..op4	=zweite bis vierte zu übergebene Variable (wenn nötig), diese sind optional

Die eigentliche Nutzerfunktion steht in eckigen Klammern!

In Programmen, welche komplexe Berechnungen anstellen sollen, die mehrfach verwendet werden, ist es sehr hilfreich nutzerspezifische Formeln als Funktionen zu definieren, denen man nur die entsprechenden Zahlenwerte übergibt.

Die Benutzung von Funktionsaufrufen ist nicht auf die fest im Basic enthaltenen Funktionen beschränkt. Du, als Nutzer kannst selbst solche Funktionen mit diesem Befehl festlegen. Im folgenden Beispiel (die Berechnung von zwei parallel geschalteten Widerständen) will ich das veranschaulichen.

Bsp.:

```
10 FUNC A(X,Y)=[1/(1/X+1/Y)]
20 A=270
30 B=180
40 C=FUNC A(A,B)
50 PRINT“Rges=“;C;“ Ohm“
```

Ausgabe -> Rges=108 Ohm

(siehe auch Beispielprogramm „Funct2.Bas“ im Basic-Verzeichnis)

In Zeile 10 wird die eigentliche Nutzerfunktion erzeugt. Der Buchstabe *A* repräsentiert den Namen der Funktion, unter der selbige aufgerufen werden kann. Die Variablen *X* und *Y* dienen zur Aufnahme der benötigten Zahlenwerte, welche in der Funktion zur Berechnung benötigt werden. Wichtig dabei zu wissen ist, das (in diesem Fall) *X* und *Y* den numerischen Variablen entsprechen.

Wird nur ein Operand benötigt, könnte die Funktion folgendermassen aussehen:

Hier wird der Arcus-Sinus von X berechnet.

Bsp.:

```
10 FUNC A(X)=[FN ATN(X/√(1-X*X))]
20 B=0.707
30 C=FUNC A(B)
50 PRINT“ARC SIN(X)=“;C
```

Ausgabe -> ARC SIN(X)=0.785247

(siehe auch Beilspielprogramm „Function.Bas“ im Basic-Verzeichnis)

Muss kein Zahlenwert an die Funktion übergeben werden, sieht das wie folgt aus

Bsp.:

```
10 FUNC A(X)=[230*13.5*0.9*√(3)]
20 C=FUNC A(0)
50 PRINT“Leistung betraegt=“;C;“ Watt“
```

Ausgabe -> Leistung betraegt=4840.22 Watt

Diese Funktion erzeugt immer das gleiche Ergebnis, da keine Variablen zur Berechnung benutzt werden.

Die gleiche Funktion mit Variablen würde so aussehen:

Bsp.:

```
10 FUNC A(X,Y,Z)=[X*Y*Z*√(3)]
20 A=230:B=13.5:C=0.9
20 D=FUNC A(A,B,C)
50 PRINT“Leistung betraegt=“;D;“ Watt“
```

Ausgabe -> Leistung betraegt=4840.22 Watt

HINWEIS:

Benutzerdefinierte Funktionen können mit bis zu vier Variablen erstellt werden.

26 unterschiedliche Funktionen sind gleichzeitig definierbar (A-Z), dabei hat der Funktionsname nichts mit der gleichnamigen Variablen zu tun.

Die eigentliche Funktion muss in eckigen Klammern stehen!

Auch wenn keine Variable in der Funktion verwendet wird, ist die Angabe der ersten Funktions-Variablen notwendig!

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 S(*on,baud)*

COM S 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 S(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: COM G

COM G 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 S(1,57600)
30 a=COM G
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: COM R

COM R 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 S(1,57600)
30 a=COM R
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 COM R-Befehl aufgerufen wird und demzufolge nur nach Empfang eines Zeichens von der seriellen Schnittstelle 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: SID 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.: SID 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: SID 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.: SID WAVE (0,1) ->wählt für Kanal 0 die Wellenform Dreieck

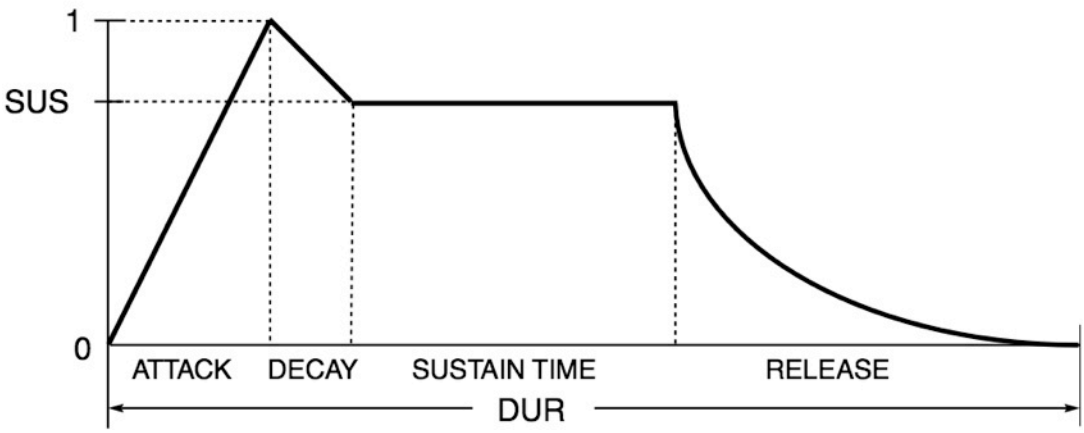
Als nächstes müssen wir noch den Lautstärkeverlauf oder in der Musiker-Fachsprache auch Lautstärke Hüllkurve des zu erzeugenden Klanges einstellen.

Lautstärke-Hüllkurve einstellen

Syntax: SID 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.: SID 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: SID NT(*Kanal,Tonhöhe)*

SID 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.:SID NT (0,60) -> es erklingt ein C auf Kanal 0

SID NT (0,0) -> Ton auf Kanal 0 ausklingen lassen

Filter setzen

Syntax: SID FLT(*lp,hp,bp)*

Setzt einen von drei möglichen Filtertypen.

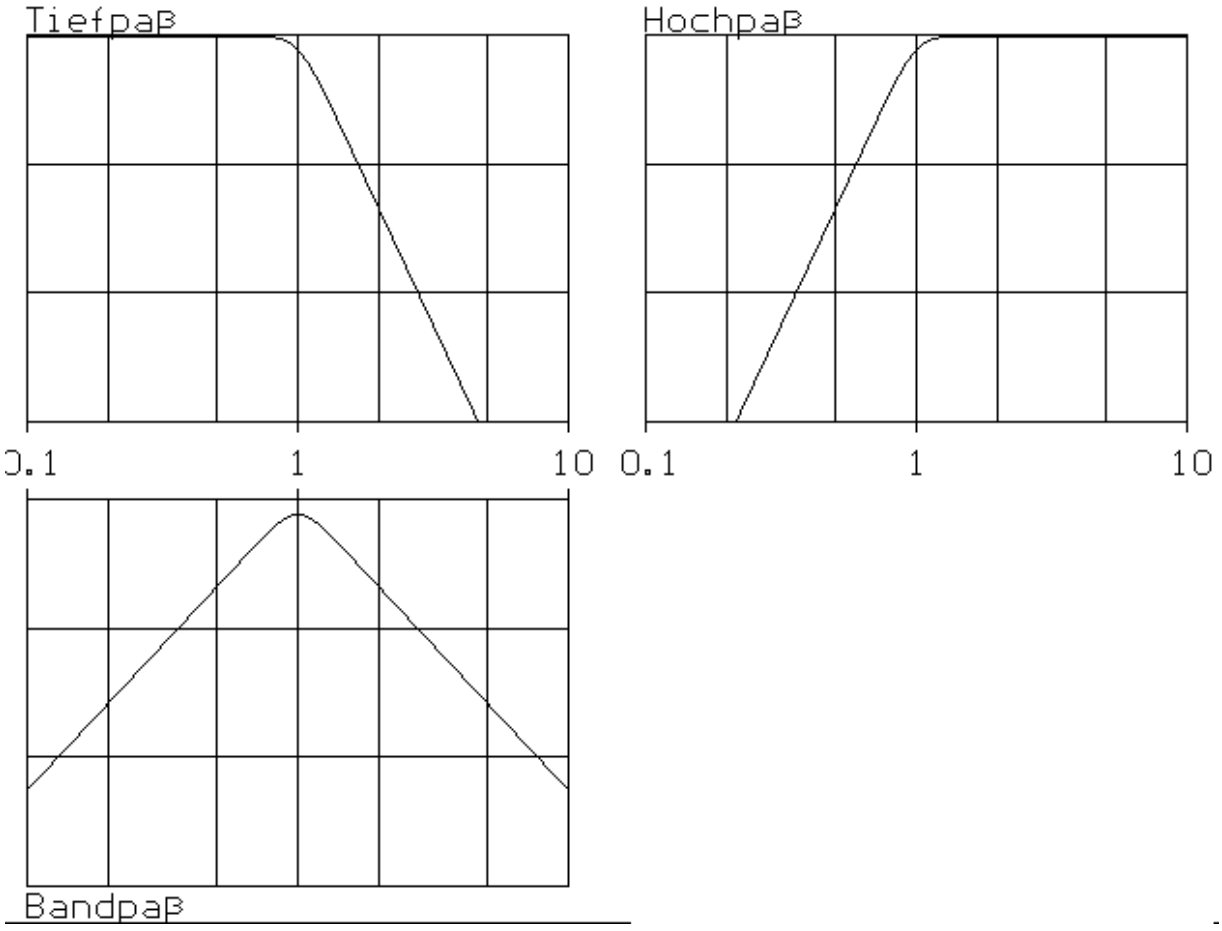
lp=Lowpass (Tiefpassfilter)

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.: SID FLT(0,0,-1) es wird der Bandpassfilter ausgewählt.

Filter zuweisen

Syntax: SID FMASK(*chan1,chan2,chan3)*

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

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

Filter-Cutoff-Resonanz-Einstellung setzen

Syntax: SID CUT(*Cutoff-Wert,Resonanz-Wert)*

Setzt die Cutoff-Frequenz des ausgewählten Filters.
Werte im Bereich von 0-1100 werden verarbeitet.
Die Resonanz-Frequenz des ausgewählten Filters kann Werte von 0-15 annehmen.
Bsp.: SID CUT(870,10)

Ringmodulator verwenden

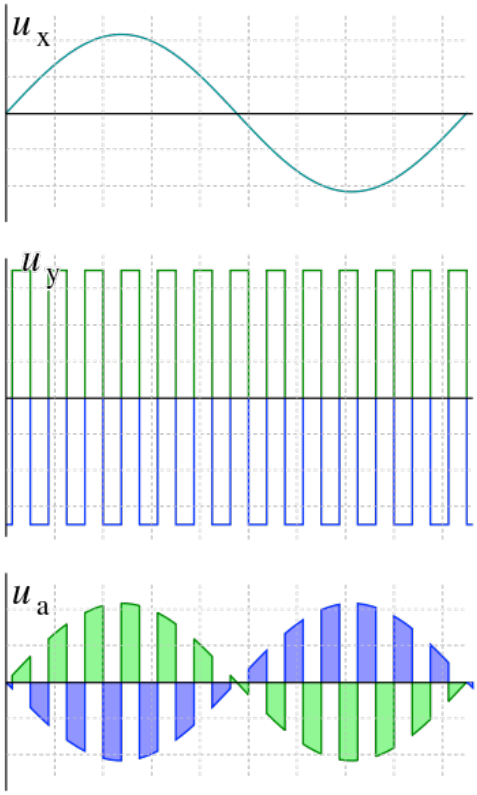
Syntax: SID RGMOD(*chan1,chan2,chan3)*

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.:SID RGMOD(0,-1,0) -> chan2 moduliert chan3

Wie die Ringmodulation zu verstehen ist, zeigt folgende Grafik.



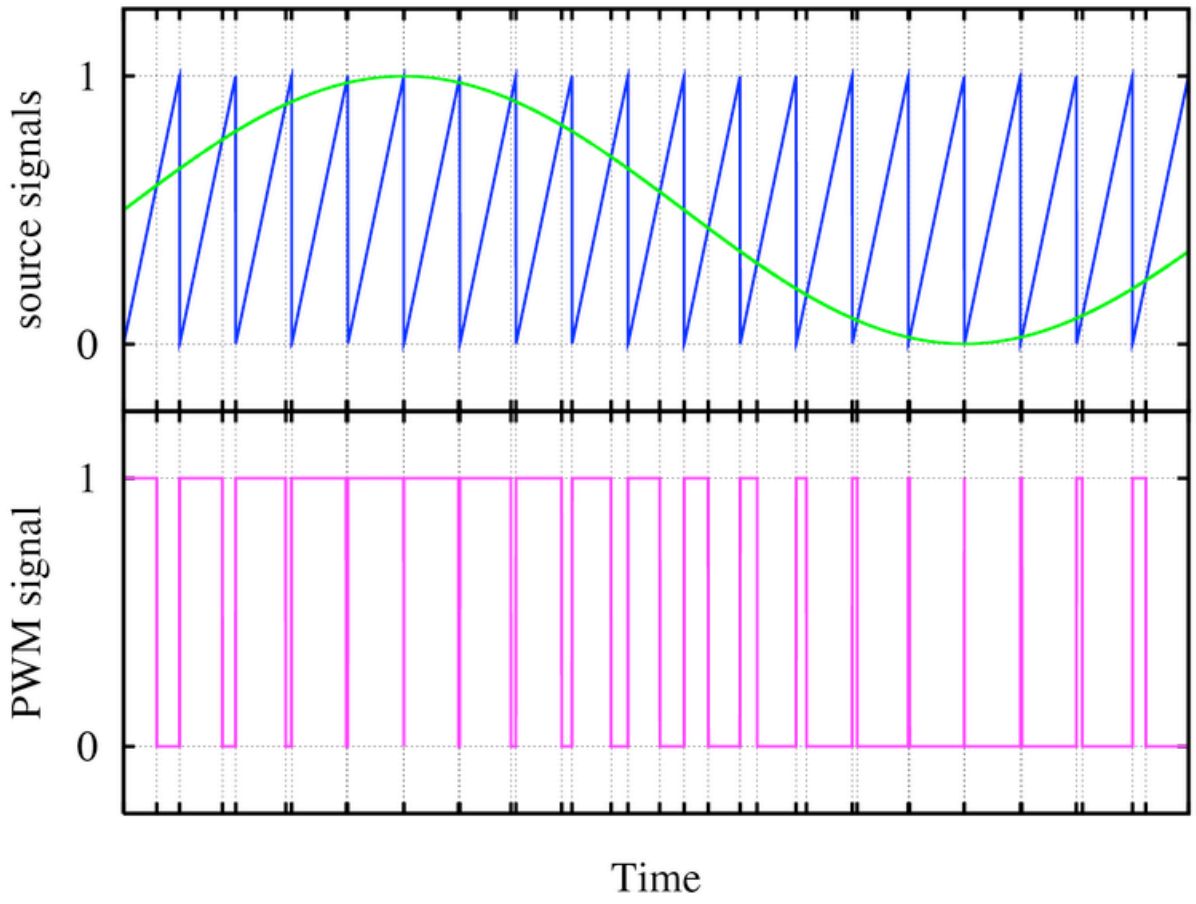
Pulsweitenmodulation

Syntax: SID PWM(*chan,wert)*

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

Bsp.: SID PWM(2,1024)

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



Synchronisation zweier Kanäle

Syntax: SID 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.: SID 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: FREAD *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 FREAD 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“

Eine Datei auf SD-Card erzeugen

Syntax: MKFILE“Dateiname.Ext“

Es wird eine Datei „Dateiname.Ext“ im aktuellen Verzeichnis erzeugt

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 kollidieren würde.

Verzeichnis anzeigen

Syntax: DIR *"FILTER",mode* **oder DIR** *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 10,2 -> Dateiliste wird in erweiterter Darstellung 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 F4 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 im Binärformat. Eine spätere Editierung am PC ist nicht möglich.

Wird als mode eine 4 eingegeben, wird das Programm als Textdatei exportiert. Eine spätere Editierung am PC ist möglich.

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

SAVE“Test.TXT“,4 ->Speichert das Programm als Textdatei

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 BAS oder TXT oder was weiss ich, ist egal).

Dateien, die mit dem mode 4 gesichert wurden, sind am PC als Textdatei editierbar.

Programm laden

Syntax: LOAD *“Name.Bas“,mode*

Ab Version 2.3 geschieht das Laden und Speichern von Basic-Programmen nicht mehr als Textdatei sondern binär. Das bedeutet, das die Basictoken beim Laden und Speichern nicht mehr in Klartext konvertiert werden, was einen erheblichen Geschwindigkeitsvorteil bringt. Um bereits vorhandene Programme weiter verwenden zu können, existiert eine Import- bzw. Exportfunktion. Aber eins nach dem anderen.

Folgende Modi werden unterstützt:

LOAD“NAME.BAS“

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

Die Syntax hat sich im Vergleich zu den Vorgängerversionen nicht verändert, nur wird, wie oben schon erwähnt, eine Binärdatei erwartet.

LOAD“NAME.BAS“,1

Die Binärdatei wird geladen und sofort gestartet.

LOAD“NAME.BAS“,2

Binärdatei wird an den ersten freien Speicherplatz im Ram abgelegt, man kann also Programmteile nachladen.

HINWEIS: Es erfolgt keine Überprüfung auf schon vorhandene Zeilennummern, also sorge dafür, das die logische Reihenfolge der Zeilen eingehalten wird.

LOAD“NAME.BAS“,3(260)

Binärdatei wird ab Zeile 260 in den Speicher geschrieben und danach weiter ausgeführt.

HINWEIS: Existiert die Zeile schon, wird sie durch den nachgeladenen Programmteil überschrieben.

LOAD“NAME.BAS“,4 -----> IMPORTFUNKTION

In diesem Modus kann ein, als Textdatei (also wie bisher) vorliegendes Basic-Programm geladen werden. Diese Funktion muss verwendet werden, um schon existierende Programme aus Vorgängerversionen weiter nutzen zu können.

HINWEIS:

Der Modus 4 ist dazu gedacht, Basic-Programme als Textdatei zu laden bzw. zu speichern. Alle anderen Modi verwenden nur noch das Binärformat. Möchtest Du ein Basic-Programm am PC weiterbearbeiten verwende zum Speichern ebenfalls Modus 4.

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
```

Datei-Namen abfragen

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 Eintraege:“;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 oder PLEXUS zurückzukehren.

Wird das ausgeführte Programm wieder verlassen, landest Du allerdings wieder in REGIME oder PLEXUS, da eines 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.

Interne Daten

DATA	Vereinbarung von Daten
READ	Lesen von Daten
RESTORE	Setzen des Datenzeigers

Der Hauptzweck eines Computers ist zweifellos, die Fähigkeit Daten jeglicher Art zu verarbeiten. Damit große Datenmengen nicht jedes mal neu eingegeben werden müssen, gelingt es mit den folgenden Anweisungen, größere Datenmengen in das Programm aufzunehmen und auf der SD-Karte zu speichern.

Syntax: DATA *Konstante,Konstante,Konstante...*

HINWEIS:

-Konstante kann eine numerische- oder Zeichenkettenkonstante sein.

-Die angegebenen Daten werden gespeichert und können ausschliesslich mit der READ-Anweisung gelesen werden.

-Die Position im Speicher muss mindestens nach der ersten Programmzeile sein.

-Sie können auch nach den entsprechenden READ-Anweisungen auftreten. Während des Programmablaufes werden DATA-Anweisungen übergangen.

-Anführungszeichen bei Zeichenketten können entfallen, wenn in ihnen keine Schlüsselwörter oder führende bzw. nachfolgende Leerzeichen enthalten sind.

-DATA Konstanten werden durch ein Komma voneinander getrennt

-hinter der letzten Konstante in einer Zeile darf kein Komma folgen

Syntax: READ *variable,variable...*

variable stellt den Namen einer numerischen oder Zeichenkettenvariablen dar, die den Wert einer Konstanten aus einer DATA-Anweisung aufnehmen soll.

Dabei erfolgt die Zuweisung fortlaufend in der Reihenfolge des Auftretens der READ-Anweisung und deren Variablen. Die Konstanten der DATA-Anweisungen werden in aufsteigender Folge entsprechend der Zeilennummerierung übernommen.

HINWEIS:

Der Variablentyp muss mit dem Typ der ihr zugewiesenen DATA-Anweisung übereinstimmen.
Bsp.:

```
10 CLS
20 DATA 1065.75, „Fritz Meyer“
30 DATA 960.35, „Sylvia Ender“
40 READ M,#N
50 PRINT #N;“ verdient „;M;“ Euro“
```

Ausgabe -> Fritz Meyer verdient 1065.75 Euro

Wie erwartet erhalten wir den Inhalt der ersten DATA-Zeile, da der READ Befehl nur einmal ausgeführt wurde.
Gib nun GOTO 40 ein und überprüfe das Ergebnis.

Ausgabe -> Sylvia Ender verdient 960.35 Euro

Wie oben erwähnt, arbeitet der READ-Befehl die DATA-Anweisungen in fortlaufender Folge ab. Eine erneute Eingabe von GOTO 40 bewirkt eine Fehlermeldung, da der Data-Zeiger das Ende des Datenfeldes erreicht hat und keine weiteren Daten verfügbar sind. Um erneut auf die gespeicherten Daten zugreifen zu können müssen wir den Data-Zeiger entsprechend setzen.

Syntax: RESTORE <*zeilennummer*>

RESTORE setzt den Data-Zeiger entweder an den Anfang des Datenfeldes (Angabe der Zeilennummer kann entfallen) oder auf die mit Zeilennummer angegebene Datenzeile. Dabei wird immer auf das erste Datenelement der DATA-Zeile gesetzt.

Gib nun RESTORE ein und GOTO 40
Gib RESTORE 30 ein und GOTO 40
Gib RESTORE 20 ein und GOTO 40

Überprüfe jeweils die Reaktion.

Auf der SD-Karte befindet sich ein kleines Demo-Programm mit dem Namen „DATA.BAS“, welches Dir die genannten Funktionen praktisch demonstriert.

Funktionen der Sepia-und Venatrix-Erweiterungskarte

Portadressen setzen

Syntax: PORT S *(AD-Adresse,Port-Adresse)*

Seit der Existenz der SEPIA-Erweiterungskarte gehört der Mangel an I/O-Ports und das Fehlen der Möglichkeit, Analogwerte auszuwerten, der Vergangenheit an. Der Hive verwendet dazu die freien Portleitungen von Administra als I2C-Bus

Auf dem Board sind 24 I/O-Leitungen und 4*8Bit AD-Wandler verfügbar. Diese wollen wir natürlich im Basic auch nutzen können. Um die Portbausteine ansprechen zu können, müssen wir nun dem Basic-Interpreter mitteilen, über welche Adressen die Chip‘s angesprochen werden. In der Standardbestückung der SEPIA-I/O-Erweiterungskarte werden die Digitalport-Bausteine auf den Adressen (dezimal) 32-34 (hexadezimal $20-$22) und der AD-Wandlerbaustein auf Adresse (dezimal) 72 oder (hexadezimal) $48 aktiv.

Jede andere Adressvergabe ist möglich, da auch mehrere SEPIA-Karten gleichzeitig verwendet werden können. Wichtig ist nur, das die Digitalport-Bausteine immer 3 aufeinander folgende Adressen haben müssen (systembedingt).

Fangen wir also an. Die SEPIA-Karte ist angeschlossen und die Adress-Jumper sind nach der genannten Restriktion gesteckt (in unserem Fall benutzen wir nicht die Standardeinstellung (diese ist im Basic schon vorbesetzt) sondern haben uns auf die Adressen 35,36,37 und 74 (für den AD-Wandler) geeinigt.

Wir setzen die Adressen mit folgendem Befehl

Bsp.:PORT S(74,35)

Im Beispiel wird nur die erste Portadresse angegeben, da die anderen automatisch in Reihenfolge belegt werden. Nun können wir schon mit der Außenwelt interagieren.

HINWEIS: Die Digitalports müssen auf 3 aufeinanderfolgende Adressen gejumpert sein!

Portausgabe

Syntax: PORT O *(Register,Wert)*

Die Portmanipulation und Abfrage erfolgt über die sogenannten Portregister. In der folgenden Aufstellung, sind die verfügbaren Registernummern und deren Bedeutung zu sehen.

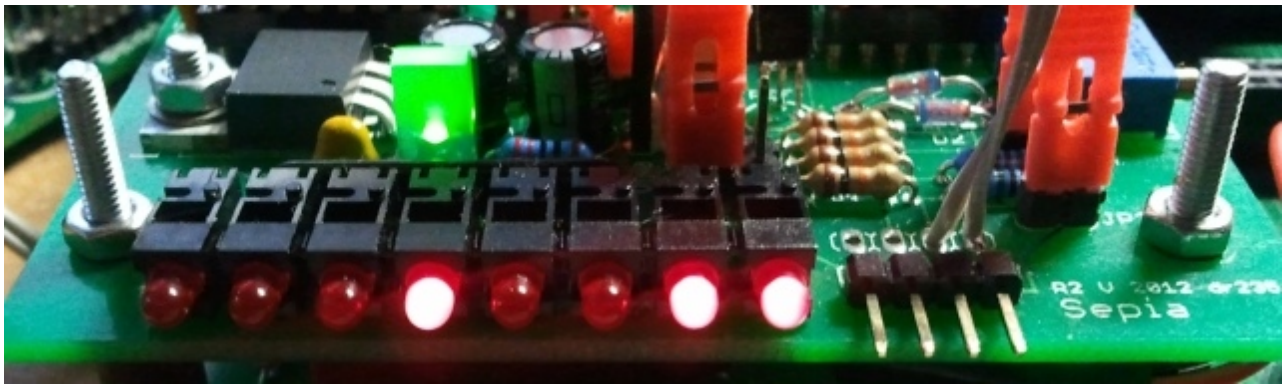
Registernummer	Port
0	AD-Kanal 1
1	AD-Kanal 2
2	AD-Kanal 3
3	AD-Kanal 4
4	Digitalport 1
5	Digitalport 2
6	Digitalport 3

Warum wird über Registernummern und nicht über die Adressen kommuniziert? wird sich jetzt der aufmerksame Beobachter fragen. Hier die Erklärung:

In Administra ist eine Cog nur damit beschäftigt, die Zustände aller Ports abzufragen. Die Abgefragten Werte werden in Registern abgelegt. Diese Register können zu jeder Zeit gelesen und manipuliert werden, ohne die Port-Adresse angeben und abfragen zu müssen.Dadurch sind schnelle Portwechsel- und Abfragen möglich, wie sie zum Beispiel zur Abfrage von Joysticks benötigt werden.

Schreiben wir also einen Wert in das Register 6 (dies ist der Port, welcher seine Zustände über LED‘s sichtbar machen kann) um das Ergebnis auch Optisch zu überprüfen.

Bsp.:PORT O(6,55)



Wenn Du keinen Fehler gemacht hast, sollte die im Foto gezeigte Konstellation zu sehen sein.

Experimentiere ruhig mit anderen Werten herum, denke aber daran, das nur der Port mit der Registernummer 6 über LED‘s zur Anzeige verfügt.

Portzustände abfragen

Syntax: PORT I *(Register)*

Um Portzustände in den HIVE einzulesen, ist dieser Befehl zu verwenden.

Kleiner Hinweis am Rande: um Joysticks abzufragen, existiert der gesonderte Befehl JOY(1 oder 2)

Wichtig zu wissen ist, das die Porteingänge Low-Aktiv sind, d.h. eine Portänderung wird durch das Schalten einer Portleitung gegen Masse herbeigeführt.

Lassen wir uns den letzten Ausgabewert (aus dem vorangegangenen Beispiel) wieder einlesen.

Bsp.:PRINT PORT I(6)

Ausgabe: 55 -> Diese Ausgabe erfolgt natürlich nur nach der Ausführung des vorangegangenen Beispiel‘s PORT O(6,55)

Portadressen abfragen

Syntax: PORT P *(Adresse)*

Es kann vorkommen, das man das Vorhandensein eines Portbausteins übeprüfen will oder man nicht genau weis, welche Portadressen schon belegt sind. Um dies herauszufinden, ist dieser Befehl entstanden.

Bsp.: PRINT PORT P($20)

Ausgabe: 0 -> eine Null bedeutet, der Portbaustein ist vorhanden.

HINWEIS:

Diese Ausgabe ist natürlich nur korrekt, wenn auch Portbausteine am I2C-Bus des Hive angeschlossen sind. Da ein inaktiver I2C-Bus (ohne Busteilnehmer) ebenfalls den Rückgabewert Null erzeugt, existiert im Basic-Verzeichnis das Programm „PING.BAS“ mit dem der gesamte Adressbereich des I2C-Busses abgefragt und angezeigt wird.

Existierende Busteilnehmer werden mit einem Kreissymbol dargestellt, wohingegen freie Adressen einen Bindestrich erzeugen.

Ist der Bus ohne Teilnehmer, werden alle Adressen als Kreissymbol dargestellt.Dies ist ein Hinweis, das keine Erweiterungskarte angeschlossen ist oder etwas mit der Verbindung nicht stimmt.

Joysticks abfragen

Syntax: JOY *(Nummer)*

Die wichtigste Funktion der Sepia-Karte ist sicher die Möglichkeit einen oder zwei Joysticks anschließen zu können. Hiermit ist der Hive nun endgültig retrofähig und steht in der Funktionalität keinem Selbstbau-Platinenrechner mehr nach.

Zur Zeit ist im Basic die Möglichkeit implementiert, Atari-, C64-oder kompatible Joysticks abzufragen und in eigenen Spielen oder Anwendungen zu verwenden. Um entsprechende Reaktionen zu bewirken, schauen wir uns das folgende kleine Programm an:

Bsp.:
```
10 CROFF:CLS
20 A=JOY(1)
30 POS 10,10:PRINT A;“  „
40 GOTO 20
```

In einer Schleife wird der Joystickport 1 abgefragt und der Rückgabewert auf dem Bildschirm angezeigt. Die Werte der Variablen A können zur Auslösung diverser Aktionen im Programm verwendet werden. Wird der Joystick am Port 2 angeschlossen, muss die Zeile 20 entsprechend in A=JOY(2) geändert werden. Es sind nur diese beiden Varianten möglich!

Direkte Bus-Kommunikation

Syntax: XBUS *(Chip,Typ,opt1,opt2,opt...)*

Da das Hive-Projekt noch in den Kinderschuhen steckt und auch spätere Funktionen für Basic erreichbar sein sollen, besteht mit diesem Befehl die Möglichkeit, eine direkte Kommunikation über den Hive-Bus zu realisieren. Natürlich sind hier auch Grenzen gesetzt. Sehr umfangreiche Kommandosequenzen werden wohl nicht immer korrekt funktionieren, da der Hive auch bei der Abarbeitung der Basiczeilen über den Bus kommunizieren muss. Aber für die Abfrage von Parametern oder das Senden von Kommandos ist diese Funktion wohl sehr effektiv einsetzbar.

Der Aufbau des XBUS-Befehls ist sieht wie folgt aus:

Chip : Hier wird angegeben, welcher Chip der Empfänger des Kommandos sein soll 1=Administra 2=Bellatrix 3=Venatrix

Typ : Angabe ob und welche Art von Variablentyp zurückgegeben werden soll 0=es wird keine Rückgabe erwartet 1=Byte 3=String 4=Long

Opt1... : Die eigentliche Kommandosequenz getrennt durch Komma‘s

Sehen wir uns dazu folgendes Beispiel an, in dem die Farbpalette von Bellatrix an der Position x=10 und y=12 auf dem Bildschirm dargestellt werden soll.

Bsp.: XBUS(2,0,0,54,10,12)

Die Kommandostruktur hat folgende Bedeutung

1.Zahl : Bellatrix ist der Empfänger des Befehls

2.Zahl : Es wird kein Rückgabewert erwartet

3.Zahl : Bellatrix-Kommando-Byte, hiermit wird Bellatrix mitgeteilt, das es sich nicht um ein Zeichen für die Bildschirmausgabe sondern um ein folgenden Befehl handelt

4.Zahl : Bellatrix-Kommando für die Ausgabe der Farbtabelle (siehe BelsVGA64.SPIN), dieses Kommando erwartet noch zwei Parameter, x und y Position

5+6.Zahl : X-und Y-Position

Beim Studium der TRIOS-BUS-Kommandos, ist eine Ableitung für die XBUS-Funktion recht einfach. Nun wollen wir uns den Namen der eingelegten SD-Karte über den BUS zurückgeben lassen und führen folgendes Beispiel aus.

Bsp.: 10 #B=XBUS(1,3,1,0,12)

20 PRINT #B

30 XBUS(1,0,24)

Nach ein paar Sekunden für die Abfrage der SD-Karte, erscheint der Name der SD-Karte (wenn dieser auch gesetzt wurde). Sehen wir uns die XBUS-Kommandosequenz an.

Zeile 10

1.Zahl : 1=Administra ist der Empfänger des Kommando‘s

2.Zahl : 3=Es wird ein String als Rückgabe erwartet

3.Zahl : 1=Um den SD-Karten-Namen lesen zu können, muss die SD-Karte gemounted werden. Dies wird mit der Befehlsnummer 1 bewerkstelligt

4.Zahl : 0=Da der Mount-Befehl einen Wert (erfolgreich 0 oder nicht 1) zurückgibt, muss eine Null für diesen Rückgabewert gesetzt werden, damit das Bus-Timing für den nächsten Befehl stimmt.

5.Zahl : 12=Nun folgt der eigentliche Befehl für die Rückgabe des SD-Karten-Namen (Befehl 12 siehe ADMSID.SPIN)

Zeile 30

1.Zahl :1=Administra ist der Empfänger des Kommando‘s

2.Zahl :0=Es wird kein Rückgabewert erwartet

3.Zahl :24=Unmount-Kommando

Starte auch mal die Basic-Datei „XBUS.BAS“, dort sind noch andere XBUS-Abfragen enthalten, welche diese Funktion näher erläutern.

HINWEIS: Die XBUS-Funktion erlaubt die direkte Kommunikation über den HIVE-BUS. Setze diesen Befehl mit Bedacht ein, da eine falsche Kommandosequenz Nebeneffekte haben kann (Datenverlust, Hängen des Hive).

Gibt eine TRIOS-Funktion einen Rückgabewert zurück, welcher nicht benötigt wird, muss eine Null als Parameter angegeben werden, um das BUS-Timing korrekt zu halten (wie im Beispiel des Mount-Befehls) .

Welche Dateien wurden mit TRIOS-BASIC mitgeliefert?

Im Verzeichnis BASIC

-Baller.Bas	kleines Ballerspiel
-Joystick.Bas	Demo der Joystickfunktion (Port1) der Sepia-Karte*
-Beep.Bas	Demo zur Funktion Beep
-Buttons.Bas	Demo zur Button-Funktion
-Button2.Bas	Demo der Text-und Icon-Button
-Climber2.Bas	ein Level des Spiels portiert vom KC87
-PING.Bas	Anzeige der im I2C-Bus vorhandenen Teilnehmer (Sepia)*
-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
-AD-Port.Bas	AD-Wandler-Werte der Sepia-Karte ausgeben*
-Data.Bas	Die DATA-Funktion in Aktion
-Fonts.Bas	Kleine Spielerei mit Tile-Fonts
-Kreis.Bas	farbige Kreise zeichnen
-LED.Bas	Portausgabe mit LED-Ausgabe der Sepia-Karte*
-Lmap.Bas	Laden einer Map-Datei
-Map.Bas	Demo Map-Datei erstellen
-Function.Bas Funct2.Bas	Demonstration der Definition von User-Funktionen (Befehl FUNC)
-Map.Dat	die mit Map.Bas erstellte Map-Datei
-Mappl.Bas	Programm, mit dem der Bildschirm für den Dmp-Player erstellt 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
-XBUS.Bas	Demonstration der XBUS-Funktion
-Timer.Bas	Timer-Demo
-Window.Bas	Demo der neuen Fensterfunktionen
-Level.Bas	Wie Joystick.Bas aber mit Pfeiltastenbedienung
-Clim2.Bas	Wie Climber2.Bas aber mit MAP zur Spielfeld-Darstellung
-Merge.Bas	Wie DATA.Bas nur werden hier die DATA-Zeilen während der Programmabarbeitung nachgeladen.
-ReadWr.Bas	Daten auf SD-Karte speichern und wieder laden
-Errors.Txt	Fehler-Texte von TRIOS-BASIC (nicht löschen!!!)
-Bel.sys	Grafiktreiber für TRIOS-BASIC
-Adm.sys	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.

*Für diese Programme ist der Anschluß der Sepia-Karte an den Hive erforderlich

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

Name	Funktion	TLOAD-Parameter
Font3.dat	Tile-Font	16,11
Font4.dat	Tile-Font	16,11
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

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

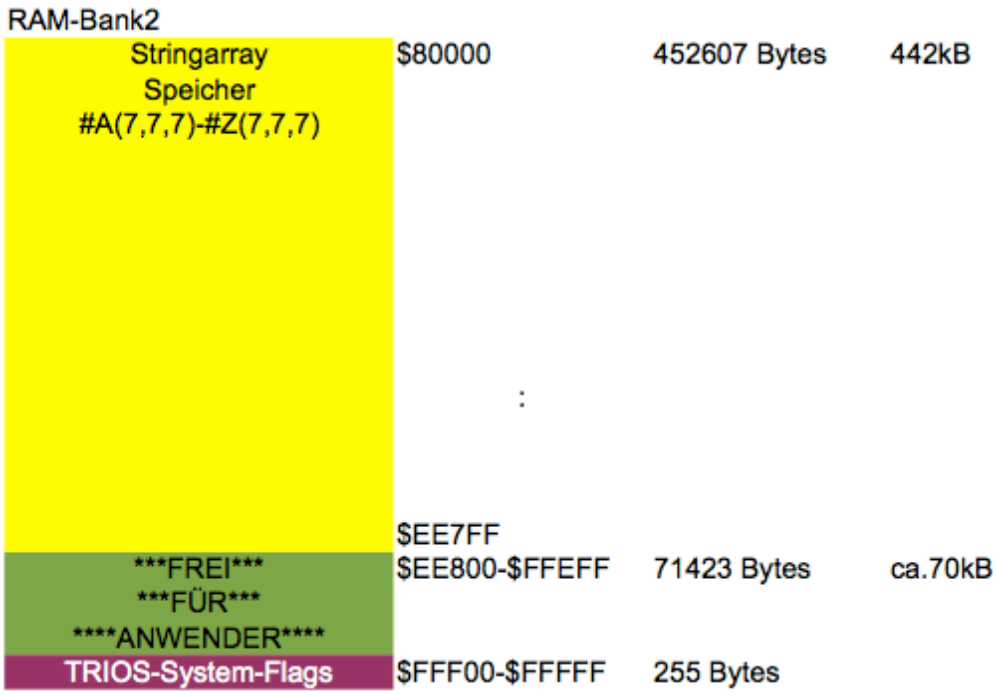
*Befehl nur über Funktionstaste aufrufbar!

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	131072 Bytes	128kb
Bearbeitungs-Speicher	$20000 : $3FFFF	131072 Bytes	128kb
Tilespeicher 14x11kB 256x176 Pixel	$40000 : $667FF	14x11kb	154kB
Systemfont	$66800-$693FF	11264 Bytes	11kB
Mouse-Tile	$69400-$6943F	64bytes	
Dir-Speicher	$69440-$6AFFF	7103Bytes	7kB
Arrayspeicher A(7,7,7)-Z(7,7,7)	$6B000-$77FFF	53247 Bytes	52kB
Map-Shadow-Speicher	$78000-$79C27	7208 Bytes	7kB
RESERVIERT	$79C28-$79FFF	984 Bytes	
DATA-Bereich	$7A000-$7DFFF	16384 Bytes	16kB
Button u. Win-Tile-Speicher	$7E000-$7E5FF	1536 Bytes	ca.1,5kB
FUNC-Speicher	$7E600-$7EFFF	2560 Bytes	ca.2,5kB
Error-Texte	$7F000-$7FAFF	2815Bytes	ca.2,7kB
DIM-Speicher	$7FB00-$7FBFF, $7FC00-$7FCFF		512Bytes
Lade-Save-Balken-Backup	$7FD00-$7FDFF	256 Bytes	
*** RESERVIERT ***	$7FE00-$7FEFF	256 Bytes	
Basic-System-Flags	$7FF00-$7FFFF	256 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 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!