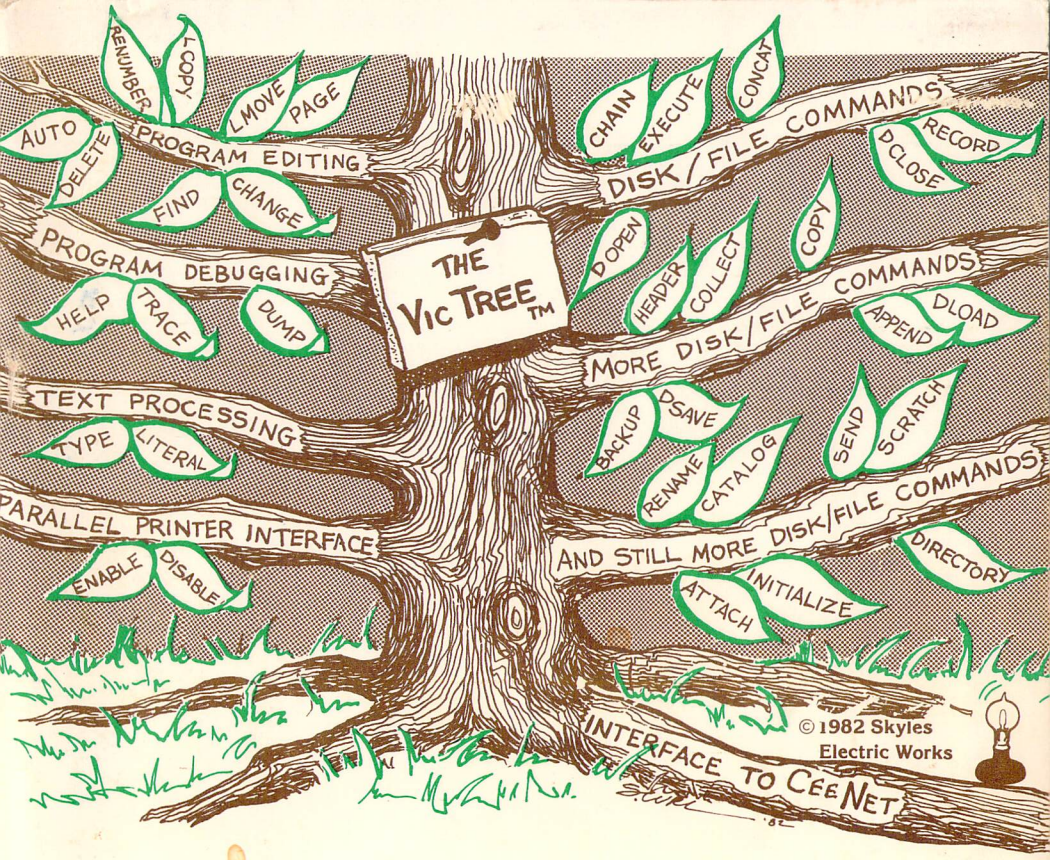

The VicTree™

Skyles Electric Works

231E South Whisman Road
Mountain View, CA 94041
(415) 965-1735

VIC-TREE
Programming Module

VIC-Tree Programming Module

By Rob Chang

December 1982

Skyles Electric Works

VIC-Tree Programming Module

By Rob Chang

December 1982

Revision B

Preface

Congratulations on your purchase of the *VIC-Tree* programming module! The *VIC-Tree* is a feature packed 'Toolbox' of powerful commands that enable you to program your VIC or CBM 64* more efficiently and easily. Specifically, the *VIC-Tree* package provides:

> Commodore BASIC 4.0 compatible disk commands and a 'translation' program which allows most existing Commodore PET/CBM programs to be used on the VIC or CBM 64.
>
> Printer interface for "Centronics standard printers."
>
> Editing commands which allow you to *renumber, interactively view, move, copy, find, change, append,* and *merge* BASIC programs or text.
>
> De-Bugging commands that let you see a program execute step by step as well as inspect variables and locate errors.
>
> Access to the *Cee-Net* 'Local Area Network'.

This manual assumes that the user is reasonably familiar with the Commodore VIC or CBM 64 and the BASIC language. There are four major sections, each of which describes disk, printer, editing, and de-bugging commands. For easy reference, a quick reference guide

*The VIC and CBM64 are trademarks of Commodore Business Machines, Inc.

is provided as a table of contents and in Appendix B. A technical reference is included in Appendix C. Lastly, Appendix A describes how to install the *VIC-Tree* .

Table of Contents

Printer Commands

Editing Commands

Vic-Tree Control

Appendices

Blank

Getting Started

Once installed (See Appendix A), the *VIC-Tree* is ready for use after you type a simple command. If you have a VIC, type:

```
SYS 45056:CLR
```

If you have a CBM 64, type:

```
SYS 32768
```

If a *Super Expander* is also installed, both modules will be ready for use. A copyright message should appear, signalling that the *VIC-Tree* has been installed properly. The screen should look something like this:

If you have a CBM 64..

```
     **** COMMODORE 64 BASIC V2 ****
  64K RAM SYSTEM 30719 BASIC BYTES FREE

READY.
SYS 32768
VIC-TREE
(C) 1982 ROB CHANG
READY.
■
```

If you have a VIC..

```
**** CBM BASIC V2 ****
2951 BYTES FREE
 -- SUPER EXPANDER --

READY.
SYS 45056:CLR
VIC-TREE
(C) 1982 ROB CHANG
READY.
■
```

Of course, the amount of memory available and the *Super Expander* message will depend on the hardware configuration of your computer. If the copyright message does not appear, or the screen fails to display anything at all, refer to Appendix A to diagnose the problem.

In some situations, you may prefer that the *VIC-Tree* is not enabled. To disable the *VIC-Tree* , simply type:

KILL <RETURN>

Where <RETURN> means 'press the RETURN key.'*

KILL removes all changes made by *VIC-Tree* to the computer's BASIC interpreter and operating system. This feature is useful if you wish to use specialized programs which interfere with the *VIC-Tree* 's operation.

The module provides 42 new BASIC commands to your computer's existing BASIC language. The commands can be broken into four major categories—disk, printer, editing, and de-bugging commands. Each command must be typed in full (i.e. Abbreviations are

*Words enclosed within angle brackets correspond to the associated key on the computer's keyboard.

not allowed). Command parameters, however, are not always necessary; *VIC-Tree* will automatically 'fill in the blanks' whenever possible. You may use the commands just as if they were BASIC commands with one exception—you must *precede* a *VIC-Tree* command with a colon if it immediately follows the THEN command. For example:

```
IF A = 0 THEN :DIRECTORY
```

The following sections describe all the commands in detail. Each command description includes the command's syntax, function, and additional notes.

Keep in mind the following notations:

Syntax

COMMAND *parameter 1, parameter 2,.. parameter n*

The syntax describes the manner in which a command must be expressed to the computer. Words shown in TYPEWRITER font must be typed as shown; words or phrases in *italics* correspond to the parameter(s) which you must specify. Parameters which are enclosed in square brackets '[..]' are optional. Do not include such brackets when entering commands. All other punctuation or characters must be entered as shown.

Line-Range

Line-Range's will often be used as parameters for various *VIC-Tree* editing commands in order to describe the range of lines which the editing command is to operate upon. A line range must be a constant. Some examples are:

```
DELETE 1000-2000
```

Delete lines 1000 through 2000, inclusive.

```
DELETE 1000-
```

Delete lines 1000 through the last line of the program, inclusive.

```
DELETE -1000
```

Delete the lines from the start of the program to line 1000, inclusive.

```
DELETE 1000
```

Delete line 1000.

Parameters

Parameters may be any valid expression which yields the needed data type. That is, a string parameter can be expressed as either a literal or a BASIC expression. For example:

```
DLOAD "WUMPUS"
```

Load the program, 'WUMPUS' from the disk.

```
DLOAD (A$+B$)
```

Load the program which name has the value of the string expression, A$+ B$.

Case

Although all examples will be shown in upper case, commands must by typed *without* the SHIFT or SHIFT-LOCK keys depressed. That is, if the computer is in lower case mode, all commands must be entered in lower case. Conversely, if the computer is in graphics

mode, all commands must be entered in upper case.

Blank

Disk Commands

The *VIC-Tree* 's disk commands allow you to use any of Commodore's disk drives simply and efficiently. There are twenty one disk commands in all.

Fifteen commands are identical in function to those in Commodore's BASIC 4.0. A translation program allows you to convert most Commodore PET/CBM BASIC 4.0 programs for use on the VIC or CBM 64. The translation program is shown on the next page.

The commands are summarized below and described in the following pages. At the end of this section, there is a detailed description of data files and a set of examples.

APPEND	Add information to a file
BACKUP	Duplicate a diskette
CATALOG	Display filenames on screen
CHAIN#	Load an overlay program
COLLECT	Check diskette
CONCAT	Put together files
COPY	Copy files
DCLOSE	Close a file
DIRECTORY	Same as CATALOG
DLOAD	Load a program
DOPEN	Open a file
DSAVE	Save a program
EXECUTE	Load and Run a program
HEADER	Prepare a blank disk for use
INITIALIZE	Prepare a disk drive for use
RECORD#	Move to specific location in a file
RENAME	Assign a new name to a file
SCRATCH	Delete (Erase) a file
SEND	Send DOS command
USE	Set assumed disk drive and device
@@	Load and run a user-defined command

Translation Program

```
100 REM BASIC 4.0 => VIC-TREE
110 D = 8 : REM USE DEVICE EIGHT
120 INPUT "BASIC 4.0 PROGRAM";B4$
130 INPUT "VIC-TREE PROGRAM";VT$
140 OPEN 1,D,0,B4$
150 OPEN 2,D,1,VT$
160 GOSUB 600
170 GOSUB 600
175 IF S1 THEN 300
180 GOSUB 600
190 GOSUB 400
200 IF A = 0 THEN GOSUB 500: GOTO 170
210 IF A < 204 THEN GOSUB 500: GOTO 190
220 PRINT#2,CHR$(203);
230 PRINT#2,CHR$(A-58);
240 GOTO 190
300 IF S1 <> 64 THEN PRINT DS$
310 CLOSE 1 : CLOSE 2
320 END
400 GET#1,A$
410 IF A$= "" THEN A$= CHR$(0)
420 A = ASC(A$)
430 S1 = ST
440 RETURN
500 PRINT#2,A$;
510 RETURN
600 GOSUB 400
610 GOSUB 500
620 GOSUB 400
630 GOSUB 500
640 RETURN
```

To use this program, follow the steps below:

1. Enter the program on the VIC or CBM 64. Save

the program on a diskette.

2. Insert the diskette which contains the BASIC 4.0 program which you wish to translate into the disk drive.

3. RUN the translation program. Include drive numbers if you have a dual disk drive. For example, if the BASIC 4.0 program, 'TANK' resides on drive zero, enter "0:TANK". Be sure to provide a VIC-Tree program name which is different from the BASIC 4.0 program name.

4. The translated program will be placed on the diskette.

Keep in mind the following notes:

Disk Device

All disk commands use the same device number unless otherwise specified. If you do not specify a device number, the *VIC-Tree* will use a device number of 8. Unless two or more disk devices are used, it is never necessary to specify a disk device number.

Disk Drive

Commodore currently manufactures both single and dual disk drives. A single disk drive (i.e. 1540, 1541, 2031, 6090, 9090) uses only one diskette; a dual disk drive (i.e. 2040, 4040, 8050, 8250) uses two diskettes. The BACKUP and COPY commands require that you have a dual disk drive. All other commands can be used with a single disk drive, although some features may not be available.

Secondary Address

The secondary address allows you to load specialized programs into memory. If you do not specify a secondary address in your command, the *VIC-Tree* will use a secondary address of 0. If you wish to load a special assembly-language program, you must specify a secondary address of 1.

Filename

A filename is a string no longer than sixteen characters. Certain characters are interpreted as 'wildcards' and allow you to manipulate more than one file at a time. The '*' wildcard causes the disk device to accept as correct any characters after the asterisk. For example, if you execute the command, SCRATCH "M*", all files which start with M will be erased from the dis-

kette. The '?' wildcard causes the disk to accept anything in the position where the question-mark resides. For example, if you execute the command SCRATCH "M?D", all files which have a three character filename starting with M and ending with D will be erased. You may combine the asterisk and question-mark wildcards (i.e. SCRATCH "M?R*" will erase any file which has M as its first character and R as its third character). Some commands do not allow the use of wildcards. The use of wildcards is discussed when applicable.

Data File

A data file is a collection of information which is stored on a diskette. There are four types of data files. The first type of data file is a PRG, or program file. These files are created with the DSAVE or SAVE commands. The second type of file is a SEQ, or sequential file. These files are created with the DOPEN or OPEN commands and allow you to store and retrieve information sequentially. The third type of file is is a REL, or relative file. These files are also created with the DOPEN or OPEN commands but allow you to store and retrieve *records* from anywhere in a file. A *record* is a line of information no longer than 254 characters. The last type of file is the USR or user file. These files are the same as sequential files with the exception that they displayed differently when the diskette directory is printed on the screen.

Logical File Number (lfn)

Logical file numbers are used to associate a given filename with a number. You may use up to nine logical file numbers at any one time.* A logical file number may be any number between 1 and 255, inclusive. If

*A maximum of ten logical file numbers are allowed, but the Vic-Tree reserves one logical file number for the DIRECTORY and MERGE commands. A logical file number of 113 is used unless you POKE location 49250 (28386 on the VIC) with a new logical file number.

the logical file number is larger than 127, a line-feed character will be sent in addition to a carriage-return character in all PRINT commands.

DS$ *and* DS *status variables*

After any disk command, the disk device returns a message indicating the success or failure of the command. This message is placed in the string variable, DS$. The numeric value of the error message is placed in the variable, DS.

Parameters

All parameters, unless otherwise specified, may be expressed as a literal constant or an expression. If you use an expression, be sure to enclose it within parenthesis. For example, the *Drive* number can be expressed as D0 or D(A) where A = 0. Parameters may be separated by commas for legibility. For example,

```
DLOAD D0 "QUAD"
```

may be expressed as

```
DLOAD D0,"QUAD"
```

It is recommended that you use the following sections and your disk manual to familiarize yourself with the capabilities of your disk device.

APPEND

Syntax

APPEND#*lfn* , [D*Drive*] *Filename* [ON U*Device*]

Function

The APPEND command allows you to add data to an existing sequential file. It is a special form of the DOPEN command.

Example

Sequential file 'NAMES' contains the following information:

(The file resides on drive 0 of device 8)

```
Maxwell Smart
Jorge Forje
Millicent McWhirter
Sam Slizinsky
```

The following program adds 'Sam Spade' to the file:

```
100 REM ADD DATA TO FILE
110 APPEND#1,DO,"NAMES" ON U8
120 PRINT DS$: REM SHOW STATUS
130 PRINT#1,"Sam Spade"
140 DCLOSE#1
```

Explanation

Line 110 informs the disk that you wish to add information to the file, 'NAMES'. The file is assigned a *logical file number* of one for all further operations. Note that you could have ommitted the DO and ON U8 parameters, since *VIC-Tree* will automatically supply these parameters if you leave them out.

Line 120 displays the the result of the APPEND command. If the disk device was unable to properly execute the command, the variables DS$ and DS will contain an error message. If APPEND was successfully executed, DS$ will contain 00, OK,00,00 and DS will contain 0.

Line 130 adds the name, 'Sam Spade' to the file.

Line 140 informs the disk device that you are finished using the file. You *must* CLOSE all disk files after using them or you may lose information.

Notes

APPEND allows you to specify 'wildcard' characters in your filename. When a wildcard is specified, the disk device will add data to the first file which matches the filename. For example, a filename of ROB? will match any of the files, ROB1, ROB2, and ROB!. APPEND will add information to the first file which matches the given filename (ROB1 in this case). The order of the files shown corresponds to the order in which they are displayed with the DIRECTORY or CATALOG commands.

BACKUP

Syntax

BACKUP D*Source Drive* TO D*Destination Drive*

Requires dual disk drive

Function

The BACKUP command will duplicate the contents of the diskette in the *Source* drive on a diskette in the *Destination* drive. Any previous information on the destination diskette will be lost.

This command requires a dual disk drive (i.e. 2040, 4040, 8050, or 8250 compatible device). If you have a single disk drive, a DOS syntax error will occur.

Example

To duplicate the diskette in drive 0, type:
BACKUP D0 TO D1

Notes

Make sure that you never transpose the *Source* diskette and the *Destination* diskette. If you do, all information on the *Source* diskette will be lost! To be safe, you should always place a *write-protect* tab on the source diskette before executing the BACKUP command.

CATALOG

Syntax

CATALOG [D*Drive*] [*Filename*] [ON U*Device*]

Function

The CATALOG and DIRECTORY commands both display the contents of a diskette. If no *Drive* is specified, *VIC-Tree* will attempt to display the contents of drives 0 and 1. The *Filename* allows you to display specific files. For example, a filename of "ROB*" will display all filenames which start with ROB. If you do not specify a filename, all files will be displayed.

The contents of the diskette will normally be displayed on the screen. If you wish to momentarily halt the listing, press the <SPACE> bar. Press <SPACE> to resume the listing. To end the the listing, press the <STOP> key.

If you wish to print the directory on a printer, follow the example below:

```
(Printer device is 4)
OPEN 4,4
CMD4
CATALOG
PRINT#4,""
CLOSE 4
```

Example

A typical directory will look like this:
On the VIC, each file entry will occupy two lines on the screen.

```
0    "COHERENT"      01 2A
10   "GAME"          PRG
12   "ROBS FILE"     SEQ
5    "MOON LANDER"   PRG
20   "DATABASE"      REL
10   "SPECIAL"       USR
587  BLOCKS FREE.
```

When shown on the screen, the first line would be displayed in reverse field. This line provides the diskette's drive, name, id, and version number. In the example above, The diskette resides in drive 0 and has the name, COHERENT, an identification number of 01 and a version number of 2A. Version numbers are used to identify the disk device which originally prepared the diskette for use.

Each file entry includes the file's size, name, and type. The size of a file is shown in multiples of 256 characters. That is, if the file size is shown as 10, the file is 2560 characters long.

The last line of the directory informs you how many blocks of free space remain on the diskette. Each BLOCK contains 256 characters.

Notes

If you have a single disk drive, be careful not to ask for a directory for drive one. In some cases, the disk may become confused and fail to display the directory properly. If you do specify a non-existent drive, you should either reset the disk (i.e. Remove the diskette and turn the disk OFF and ON) or type:

```
SEND "U:"
```

CHAIN#

Syntax

CHAIN#*Line-Number*, [D*Drive*] *Filename* [ON U*Device*]

Function

The CHAIN# command will LOAD a program under program control. This allows you to execute programs which are too large to fit in program memory. A program which is loaded with the CHAIN# command is called a *program overlay.*

CHAIN# will load the program overlay starting at the given *Line-Number.* The *Line-Number* must be a constant.

There are two very important limitations with CHAIN#:

Never use CHAIN# *when editing programs*

The program overlay must be smaller that the first program that uses CHAIN#
(The first program may be enlarged by the use of dummy lines.)

Example

When writing a very large application program, you may run out of program memory. In order to avoid this problem, program overlays can be used to create programs which are virtually unlimited in size. Such a program consists of an *Executive* program and many *Utilities.* The *Executive* program is responsible for

CHAIN#ing each utility as it is needed.

In the example below, the program provides a simple database. Two utilities are used, ADD and SHOW. These utilities add a name to the database and list all the names in the database, respectively. The Executive program is as follows:

```
50 DIM N$(100) : L = 0
100 "PLEASE SELECT FUNCTION"
110 PRINT " 1.  ADD A NAME TO DATABASE"
120 PRINT " 2.  SHOW NAMES IN DATABASE"
130 INPUT A
140 IF A = 1 THEN NA$= "ADD"
150 IF A = 2 THEN NA$= "SHOW"
160 IF (A < 1) OR (A > 2) THEN 100
170 CHAIN#1000,DO,(NA$)
180 GOSUB 1000
190 GOTO 100 : REM DUMMY LINES BELOW
1000 ::::::::::::::::::::::::::::::::::::
1010 ::::::::::::::::::::::::::::::::::::
1020 ::::::::::::::::::::::::::::::::::::
etc.
2000 ::::::::::::::::::::::::::::::::::::
```

Lines 1000 and on insure that the executive program is larger than the utilities. If a utility exceeds the size of the executive, all variables will be lost.

The utility, 'ADD' is stored on drive 0 as:

```
1000 INPUT "NAME";N$
1010 L = L+1
1020 N$(L) = N$
1030 RETURN
```

The utility, 'SHOW' is stored on drive 0 as:

```
1000 FOR I = 0 TO L
1010 PRINT N$(I)
1020 NEXT
1030 RETURN
```

The executive will LOAD the appropriate overlay program when needed; memory is not wasted when a utility is not in use. In order to expand the program, the executive only needs to CHAIN# a new file.

Notes

The CHAIN# command is similar to the CHAIN command (See Editing Commands) with one exception– whereas CHAIN resets BASIC's variable storage area, CHAIN# does not. This allows program overlays to share the same set of variables. For this reason, do *not* use the CHAIN# command when editing programs, as BASIC will become confused as to the size of your program. The computer may even 'crash' (i.e. You will have to turn your computer OFF and ON in order to regain control) in such situations.

Early versions of Commodore BASIC (Level 1, Version 2) contain an error which causes BASIC programs to be saved with an extra character. This extra character causes the CHAIN and MERGE commands to become confused. In order to eliminate the extra character from such programs, follow the directions below:

1. Load the Program; type:

```
A=PEEK(45)+PEEK(46)*256:A=A-1
POKE 46,A/256
POKE 45,(A/256-INT(A/256))*256
```

2. Save the Program

COLLECT

Syntax

COLLECT [D*Drive*] [ON U*Device*]

Function

The COLLECT command will check all files on a specified diskette for proper format and errors. If any errors are found, the disk device will attempt to correct them. An error message will be returned if COLLECT is unable to correct any errors. If you do not specify a drive number, both drives will be checked, if present.

Example

COLLECT D0
Check diskette in drive zero of disk device 8.

COLLECT
Check diskettes in drive zero and one of disk device 8.

Notes

The COLLECT command causes the disk device to check the BAM (Block Availability Map) on a specified diskette. The BAM contains a map of all available blocks on a diskette. If there are any errors in the BAM, a new BAM will be placed on the diskette.

CONCAT

Syntax

CONCAT [D*Drive*] *Source Filename* TO [D*Drive*] *Recipient Filename* [ON U*Device*]

Function

CONCAT reads the *source* file and adds it to the end of the *recipient* file. For example, if FILE1 contains A B C and FILE2 contains D E F, the operation:

```
CONCAT "FILE1" TO "FILE2"
```

would make FILE2: D E F A B C. If no drive numbers are specified, CONCAT will use a drive of 0. If there is no drive number included with the *Recipient* filename, the drive number from the first filename will be used.

Notes

CONCAT does not care what type of *source* file that you concatenate. You can CONCAT a PRG file to a SEQ file, or even a file to itself. The *recipient* file, however, must be a SEQ file.

COPY

Syntax

COPY D*Source-Drive* TO D*Destination-Drive* [ON U*Device*]

or

COPY [D*Drive*] *Source Filename* TO [D*Drive*] *Recipient Filename* [ON U*Device*]

Function

The COPY command exists in two different forms. The version of COPY without filenames is used to transfer all files from one disk drive to another disk drive. This form of COPY requires that you have a dual disk device. COPY differs from BACKUP in that files are only copied, and any files already on the destination diskette are not destroyed. The syntax of the COPY command is the same as the BACKUP command.

The Second form of COPY allows you to transfer single files to either the same disk drive or other disk drives. If the transfer is confined to the same disk drive, the *Recipient* filename must be different from the *Source* filename.

Both forms of COPY require that the destination diskette does not contain any filenames identical to the source filename.

Example

```
COPY DO TO D1
```

Copies all files from diskette in drive 0 to diskette in drive 1. Requires a dual disk device.

```
COPY DO,"BACKGAMMON" TO "BACKUP"
```

Copies the file BACKGAMMON *into the file* BACKUP *on drive 0.*

```
COPY DO,"TANK" TO D1,"NEW-TANK"
```

Copies the file TANK *from the diskette in drive 0 into the file* NEW-TANK *on drive 1.*

Notes

The COPY command will *not* copy relative (REL) files.

The DOS (Disk Operating System) in the Commodore 4040 and 8050 disk drives contains an error which causes the COPY command to fail when copying more than one file (First form of COPY). Sometimes, the disk drive will return an error after copying eight files. You should use the 'copy files' or 'unit to unit' utility program (supplied on the demonstration diskette included with your disk device) in order to avoid this problem.

DCLOSE

Syntax

DCLOSE [#*lfn*] [ON U*Device*]

Function

The DCLOSE command instructs the disk to close the specified file. If no *logical file number* is given, DCLOSE will close all files opened on the specified disk drive.

It is very important to CLOSE or DCLOSE all files that are written to the disk. Failure to close files will cause a loss of data.

Example

DCLOSE#1
Close logical file one.

DCLOSE ON U9
Close all logical files on device number 9.

DCLOSE#2 ON U9
Close logical file two on device number 9.

DIRECTORY

Syntax

DIRECTORY [D*Drive*] [*Filename*] [ON U*Device*]

Function

DIRECTORY is the same as the CATALOG command. The command is provided for Commodore Disk BASIC 4.0 compatibility. Please refer to the CATALOG command for more information.

DLOAD

Syntax

DLOAD [D*Drive*] *Filename* [CMD*Second-Address*]
[ON U*Device*]

Function

The DLOAD command will load BASIC or machine language programs from the disk into program memory. The command is the equivalent of:

LOAD "*Filename*",*Device*,*Second-Address*
(Or "0:*Filename*" *if drive zero is specified)*

If you do not specify a *drive* when DLOADing a file, both disk drives will be searched for the given filename, if present. If you specify a drive, only that drive will be searched for the filename.

Example

To load the BASIC program, 'AIRPLANE' from disk:
DLOAD "AIRPLANE"

Where AIRPLANE *resides on either drive 0 or 1 on the current disk device.*

To load an assembly language program, 'PONG' from disk:

DLOAD "PONG",CMD1

Where PONG *resides on either drive 0 or 1 on the current disk device.*

Notes

If you wish to load a *non-relocatable* program such as an assembly language program, you must specify a secondary address of one.

DLOAD allows the use of wildcards. When a wildcard is used, DLOAD will load the first matching filename which it finds. For example, if the diskette contains the files:

ROCK ROCKS ROCKSTAR and ROCKY

and you specify:

```
DLOAD "ROC*"
```

all the files are acceptable choices, but DLOAD will load the first file, ROCK into memory.

DOPEN

Syntax

DOPEN#*lfn*, [D*Drive*] , *Filename* [, L*Size*] [, R] [, W]
[ON U*Device*]

Function

DOPEN instructs the disk to either create a file or read a file. The file is also assigned a *logical file number* for all further file commands.

The *Logical file number* can be any number from 1 to 255, inclusive. Up to 9 logical files can be open at any one time. Refer to your disk operators manual in order to see how many disk files you may use at any one time.

The *Filename* may be any legal disk filename up to 16 characters long. The wildcard characters, * and ? may be used when a file is opened for reading only. The first file meeting the specifications will be used. If you attempt to create a file with wildcards, the disk will return a syntax error.

If no *Drive* number is selected, 0 will be assumed.

If the *Length* parameter is included, a RELative file will be created. The *length* of each record may be from 1 to 254 characters. After a REL file has been created, you do not need to include the L parameter each time you open that file.

The W parameter informs the disk that you wish to write information to the file. You may write to a relative file at any time without specifying the W parameter. You

must specify the W parameter if you intend to write to a sequential file.

The R parameter, in conjunction with the L parameter, will cause the disk to erase the file before writing to the file.

Example

```
DOPEN#1,"NAMES",W
```
Create the sequential file, 'NAMES' and prepare to write information to the file.

```
DOPEN#2,"ADDRESSES"
```
Open the file 'ADDRESSES' and prepare to read the file.

```
A$= "DATABASE"
DOPEN#3,(A$),L80
```
Open the relative file, 'DATABASE' and assign each record a length of 80 characters.

DSAVE

Syntax

DSAVE [D*Drive*] *Filename* [ON U*Device*]

Function

DSAVE will save the current program in memory on disk. If no *Drive* is specified, a drive number of 0 will be used.

The *Filename* may be any filename without wildcards. Strange results may occur if you include wildcards in your filename.

Example

To save the program currently in memory, type:
```
DSAVE "TEST"
```

The program will be stored on drive 0 on device 8

To save the program on disk device 9, drive 1:
```
DSAVE D1,"TEST" ON U9
```

EXECUTE

Syntax

EXECUTE [D*Drive*] *Filename* [CMD*Second-Address*]
[ON U*Device*]

Function

The EXECUTE command will load a BASIC program from the disk into program memory and immediately RUN the program. It is the equivalent of DLOADing a program and RUNning the program.

If you do not specify a *drive* when DLOADing a file, both disk drives will be searched for the given filename, if present. If you specify a drive, only that drive will be searched for the filename.

Example

To load and RUN *the program, 'AIRPLANE' from disk:*
EXECUTE "AIRPLANE"

Where AIRPLANE *resides on either drive 0 or 1 on the current disk device.*

Notes

EXECUTE allows the use of wildcards. When a wildcard is used, EXECUTE will load the first matching filename which it finds. For example, if the diskette contains the files:

ROCK ROCKS ROCKSTAR and ROCKY

and you specify:

```
EXECUTE "ROC*"
```

all the files are acceptable choices, but **EXECUTE** will load the first file, **ROCK** into memory.

HEADER

Syntax

HEADER [D*Drive*] *Disk-Name* [,I*Id*] [ON U*Device*]

Function

The HEADER command will prepare a blank or old diskette for use. If any files are on the diskette, they will be lost. If no *Drive* is specified, a drive number of 0 will be used.

If the *Id* parameter is included, the entire disk will be formatted and checked for errors. This must be done for blank disks. If the I parameter is not included when formatting an old disk, only the diskette directory will be cleared.

If you execute the HEADER command in immediate mode (i.e. not from within a program), HEADER will confirm your request with the following message:

```
ARE YOU SURE?
```

You may then enter N <RETURN> to abort the command or Y <RETURN> to proceed with the command.

Example

To format a new diskette, type:

```
HEADER DO,"MY DISK",IRC
```

The command takes approximately 2 minutes to complete on a 1541 disk drive.

To format an old diskette, type:

```
HEADER DO,"MY DISK"
```

The command takes approximately 10 seconds to complete.

Notes

If you do not specify a disk *id*, the disk device assumes that the disk has already been formatted. The diskette directory and BAM are cleared. This process takes no longer than ten seconds on a 1541 disk device.

If you specify a disk *id*, every sector on the diskette is formatted and checked for errors. This process takes aproximately two minutes to complete and must be done before using new diskettes.

INITIALIZE

Syntax

INITIALIZE [D*Drive*] [ON U*Device*]

Function

The INITIALIZE command will prepare a disk drive for use. Whenever you change diskettes, you should always INITIALIZE the drive before reading or writing to the disk.

If no *Drive* is specified, both disk drives will be initialized, if present.

Example

To initialize drive 0, type:
INITIALIZE D0

Notes

The INITIALIZE command causes the disk device to update its copy of the diskette's BAM (Block Availability Map) in internal memory. Normally, the disk will automatically detect when to update its BAM. If, however, you insert a diskette in a disk drive which has the same disk *id* as the previous disk, the disk device will *not* update its BAM. This can cause horrible file errors. Using the INITIALIZE command before writing to any newly-inserted diskette will insure that such problems do not occur.

RECORD

Syntax

RECORD#*lfn*, *Record-Number* [, *Position*]

Function

RECORD causes the disk device to prepare a record for reading or writing. If a *Position* is also specified, a pointer will be positioned to that character in the record. The first postion will be selected if the position parameter is ommited.

Example

The following code will read the second record from the existing file, 'NAMES'.

```
DOPEN#1,"NAMES"
RECORD#1,2
INPUT#1, A$
```

RENAME

Syntax

RENAME [D*Drive*] *Old-Name* TO *New-Name* [ON U*Device*]

Function

The RENAME command will assign a new name to a filename currently on the diskette. The *New-Name* can not already exist on the diskette.

Example

Change filename 'FOO' to 'ZOO'
RENAME "FOO" TO "ZOO"

SCRATCH

Syntax

SCRATCH [D*Drive*] *FileName* [ON U*Device*]

Function

The SCRATCH command will erase one or more files from a diskette. If you specify a wildcard, more than one file may be erased. After a file has been SCRATCHed from the diskette, you can not recover that file.

Example

To scratch all programs which start with 'A', type:
SCRATCH "A*"

all programs which start with 'A' will be erased from drive 0 of disk device 8

To scratch the program, 'STARTREK', type:
SCRATCH "STARTREK"

Where 'STARTREK' resides on drive 0 of disk device 8.

To scratch the program 'XX' on drive one, type:
SCRATCH D1,"XX"

To scratch the program referenced by A$*, type:*
SCRATCH (A$)
If A$= "BASEBALL", *the file 'BASEBALL' will be scratched from drive zero.*

Notes

The SCRATCH command will always tell you how many files were erased from the diskette with the following message:

```
DISK:01, FILES SCRATCHED,XX,00
```

where XX corresponds to the number of files erased from the diskette. If SCRATCH is issued as an immediate-mode command, you must confirm the operation by typing Y <RETURN> when the command displays:

```
ARE YOU SURE ?
```

Any other response will abort the command.

SEND

Syntax

SEND *CBM DOS command* [ON U*Device*]

Function

The SEND command allows you to send a CBM disk command. It is the equivalent of:

```
OPEN 1,8,15
PRINT#1,CBM DOS command
CLOSE 1
```

Example

To reset the disk, type:

```
SEND "U:"
```

or

```
A$= "U:"
SEND A$
```

Notes

The SEND command does not use a logical file in order to send the DOS command to the disk.

After sending a DOS command, you can see if there were any errors by typing:

```
?DS$
```

USE

Syntax

USE [D*Drive*] [CMD*Secondary-Address*] [ON U*Device*]

Function

Normally, if you do not include a *Drive* number, a *Secondary Address*, or a *Device*, *VIC-Tree* will assume a *Drive* of 0, a *Secondary Address* of 0, and a *Device* of 8. The USE command allows you to change these parameters.

Example

To assume a drive of 1, type:
USE D1

From now on, if you do not specify a drive number, the VIC-Tree will assume a drive of 1.

To assume a disk device of 9, type:
USE ON U9

From now on, disk drive 9 will be used if you do not specify a device number.

To a assume a Secondary Address of 1, type:
USE CMD1

From now on, DLOAD *and* EXECUTE *will use a secondary address of 1 if you do not specify a secondary address. This is convenient if you are loading many nonrelocatable programs such as assembly-language games.*

Notes

You may specify more than one parameter with the USE command. For example, the above examples could be expressed in one USE command:

```
USE D1 CMD1 ON U9
```

If a parameter is not included, the value in not changed.

Syntax

@@ *Command-Name* [*Parameters*]

Function

The @@ command allows you to execute user defined BASIC commands which are stored on the disk. With this command, you can create your own library of BASIC commands

This command is intended for advanced users; a knowledge of assembly language and the internal architecture of your computer's BASIC interpreter is required to use this command properly.

The command is primarily used for *Cee-Net* operating system commands.

Notes

Appendix C documents this command in detail.

Disk Examples

The following section shall discuss the concept of data files, both relative and sequential, and provide several examples. A full discussion of data files is beyond the scope of this manual and you are advised to further familiarize yourself about data files with the use of your disk operators manual.

Files. Generally

A data file is a collection of information which is organized as lines or individual characters. Every data file is assigned a unique name. The maximum size of a file, and the maximum number of files available on any diskette is determined by your disk. Please refer to your disk manual for such information. There are two methods of reading the contents of a file. You may inspect the contents of the file as if it were a 'ticker-tape' (From start to end), or you may read arbitrary lines in the file. When you read the file like a 'ticker-tape', you may not back up or skip forward along the file. This type of file is called a *Sequential* file. Filenames which are displayed with the name PRG, SEQ, or USR are all sequential files.

The second method of reading files utilizes the *relative-record* features of your disk device. When such a file is created, it is assigned a REL attribute. Such files are organized in groups of lines, each of which is called a *record.* A record may be from one to 254 characters in length. In order to read a specific record in a data file, all you need to do is to point to the record via the RECORD command and then read the record with the INPUT command. This method of reading files can be much faster than reading sequential files.

Examples

Sequential Files

Suppose you wish to save a list of creatures for an Adventure game. Each critter has the following features:

Name
Hit points to kill
Hit points to weaken
Magic spell

For example, you might have a 'Grumpy Gryphon' with the following features:

Name: `Grumpy Gryphon`
Hit points to kill: `1000`
Hit points to weaken: `500`
Magic spell: `"Plugh!"`

The task is to build a data file which contains these descriptions so an Adventure program can later read them. By having various data files with different descriptions of creatures, many different Adventures are possible.

A program to create such a data file is shown below:

```
10 PRINT "CREATURE MAKER"
20 INPUT "NAME OF CREATURE FILE";F$
30 DOPEN#1,(F$),W
40 CR$= CHR$(13)
50 PRINT
60 PRINT "USE 'QUIT' TO STOP"
70 PRINT
80 INPUT "NAME:";N$
```

```
90 IF N$= "QUIT" THEN 180
100 PRINT#1,N$;CR$;
110 INPUT "HIT POINTS TO KILL:";HK
120 PRINT#1,HK;CR$;
130 INPUT "HIT POINTS TO WEAKEN:";HW
140 PRINT#1,HW;CR$;
150 INPUT "MAGIC SPELL";S$
160 PRINT#1,S$;CR$;
170 GOTO 70
180 DCLOSE#1
190 END
```

DSAVE this program and RUN it.

A typical dialog would look like this: (*Slanted* words are your input)

```
CREATURE MAKER
NAME OF CREATURE FILE ?  BEASTS

USE 'QUIT' TO STOP

NAME: ?  FLUFF
HIT POINTS TO KILL: ?  1234
HIT POINTS TO WEAKEN: ?  290
MAGIC SPELL: ?  GRERTPHLOT

NAME: ?  GREEN YOCKALOCK
HIT POINTS TO KILL: ?  1000
HIT POINTS TO WEAKEN: ?  800
MAGIC SPELL: ?  OOMFA LOP

NAME: ?  QUIT

READY.
```

Briefly, line 20 asks for the name of the data file to create. In line 30, DOPEN creates the file. Note that the variable containing the filename is enclosed inside

parenthesis (It is an expression). The W parameter tells the disk device that you wish to create a sequential file.

If this program were to *add* information to the file, the DOPEN would be replaced with the APPEND command. Note that the APPEND command does not require the W parameter.

Line 40 defines the *carriage-return* character. This character is used to end a line of information.

Line 90 checks if the user is done. If the user is done, the program will then DCLOSE the file. You must always close open files before ending the program or data may be lost.

Now, let's read the information from the file:

```
10 PRINT "CREATURE FINDER"
20 INPUT "NAME OF CREATURE FILE";F$
30 DOPEN#1,(F$)
40 PRINT
50 INPUT#1,N$,HK,HW,S$
60 IF ST = 64 THEN 120
70 PRINT "NAME:"; N$
80 PRINT "HIT POINTS TO KILL:";HK
90 PRINT "HIT POINTS TO WEAKEN:";HW
100 PRINT "MAGIC SPELL";S$
110 GOTO 50
120 DCLOSE#1
130 PRINT "DONE."
140 END
```

Line 30 informs the disk device that you wish to read information from the filename contained in F$. The INPUT statement on line 50 reads the information from the file.

Line 60 determines whether if you have read the

entire file. For more information on the exact meaning of the ST variable, you should consult your disk manual.

Relative Files

Suppose we wish to implement the same creature file with relative records. Using a REL file, each record will represent the information about a creature. We can therefore recall information about any creature at random. The program is shown below:

```
10 PRINT "CREATURE MAKER- REL FILES"
20 INPUT "NAME OF CREATURE FILE";F$
30 DOPEN#1, (F$),L(100)
40 CR$= CHR$(13)
45 R = 1
50 PRINT
60 PRINT "USE 'QUIT' TO STOP"
70 PRINT
75 RECORD#1,(R)
80 INPUT "NAME:";N$
90 IF N$= "QUIT" THEN 180
110 INPUT "HIT POINTS TO KILL:";HK
130 INPUT "HIT POINTS TO WEAKEN:";HW
150 INPUT "MAGIC SPELL";S$
160 PRINT#1,N$;CR$;HK;CR$;HW;CR$;S$;CR$;
165 R = R+1
170 GOTO 70
180 DCLOSE#1
190 END
```

Line 30 now creates a relative file instead of a sequential file. The L parameter instructs the disk that each record will be 100 characters long at maximum. Once you have created a relative file, all subsequent DOPEN's do not need to specify the L parameter.

In line 45, the variable R contains the current creature number. This value is used to instruct the disk

which record which you wish to read or write.

After you have entered the information, line 160 stores all the information in the desired record. Note that all this information must be placed in one PRINT# statement, or each feature would be placed in succeeding records.

Line 165 advances the creature number.

Line 180 instructs the disk that you are finished. Again, you must be sure to CLOSE all data files or you will lose information.

In order to read the information from the file, you would use the following program:

```
10 PRINT "CREATURE FINDER-REL FILES"
20 INPUT "NAME OF CREATURE FILE";F$
30 DOPEN#1,(F$)
40 PRINT
45 INPUT "CRITTER NUMBER";R
46 RECORD#1,(R)
50 INPUT#1,N$,HK,HW,S$
60 IF ST <> 0 THEN 120
70 PRINT "NAME:"; N$
80 PRINT "HIT POINTS TO KILL:";HK
90 PRINT "HIT POINTS TO WEAKEN:";HW
100 PRINT "MAGIC SPELL";S$
110 GOTO 40
120 PRINT "EMPTY RECORD"
125 INPUT "QUIT";Q$
126 IF Q$= "NO" THEN 40
127 DCLOSE#1
130 PRINT "DONE."
140 END
```

This program will ask for individual creature numbers. Given a creature number, the corresponding

record will be read and displayed. If the record is empty (i.e. You never stored anything in that record), a message will be printed.

Line 45 asks for a creature number. This number is used to prepare the corresponding record to be read.

Line 46 notifies the disk that you wish to perform an operation on the *R*th record of the file. Note that after the RECORD command has been used, you may either write or read the affected record.

Line 50 reads the information from the desired record.

Printer Commands

The *Vic-Tree* allows you to use any printer that has a Centronics standard interface. The printer does not require any additional hardware; a printer cable attaches directly to the VIC's or CBM 64's parallel user port. When enabled, the *Vic-Tree* 's printer interface allows you to use the printer via conventional PRINT# statements. In addition, you may instruct the *Vic-Tree* to translate the computer's character codes into printer-compatible codes. There are two commands associated with the printer. They are shown below and described in the following pages.

ENABLE . . . . . Enable printer
DISABLE . . . . . Disable printer

ENABLE

Syntax

ENABLE [*Device-Number* [,*Mode*]]

Function

The ENABLE command will assign the attached printer a number. Acceptable *Device-Numbers* range from 4 to 31. If you do not specify a device number, *Vic-Tree* will use the number four.

Once ENABLEd, you may use the OPEN and PRINT# commands to print information. The OPEN command has the following syntax:

OPEN *lfn,Device-Number*

The *lfn* is used to reference the printer in all further statements. A logical file number may be any number from 1 to 255, inclusive. If your printer requires a *Line-Feed* character at the end of each line, you should specify a logical file number larger than 127.

The *mode* allows you to change certain characters so that they will properly appear on the printer. There are four possible modes. They are shown below:

0 Control codes expanded; translation enabled
1 Control codes expanded; translation disabled
2 Control codes printed; translation enabled
3 Control codes printed; translation disabled

Control codes are non-printable characters which correspond to the cursor control characters on the VIC or CBM 64. When control characters are ex-

panded, they will be preceded with ^. For example, the <CURSOR-DOWN> character, when printed on the screen, appears as a Q in reverse field. The same character appears as ^q on the printer. Shifted control keys such as <SHIFT> <CURSOR-UP> are similar, but they will appear as upper case characters on the printer. This feature allows you to LIST a program which contains quoted strings.

Many printers use control characters to enable various printing options. To send a printer control characters, you must select a mode which prints control codes (i.e. modes 2 or 3).

Upper and lower case characters on the computer are not numerically compatible with ASCII standard printers. In order to properly display upper and lower case characters, you must enable translation (i.e. specify modes 0 or 2).

If you do not specify a mode, a mode of zero will be assumed. A mode of three will transmit all characters to the printer unchanged.

After you have OPENed the printer for use, you may then print information with the PRINT# command. The command is identical to the PRINT command with the exception that you must include a logical file number. You may also print information with the CMD command. The CMD command instructs the computer to send all output from PRINT commands to the printer.

Examples

The following program will print the alphabet in upper and lower case on a standard ASCII printer. The printer does not require a line-feed character at the end of each line.

```
100 ENABLE 4,3
110 OPEN 1,4
120 FOR X = 97 TO 122
130 PRINT#1,CHR$(X);
140 NEXT
150 PRINT#1,""
160 FOR X = 65 TO 90
170 PRINT#1,CHR$(X);
180 NEXT
190 PRINT#1,""
200 CLOSE 1
```

Line 100 enables the printer to be used.

lines 120–180 print the alphabet in lower case and upper case. Each alphabet is on a different line. Line 150 causes the printer to start a new line.

If you were to use the CMD command, the program would look like this:

```
100 ENABLE 4,3
110 OPEN 1,4
115 CMD 1
120 FOR X = 97 TO 122
130 PRINT CHR$(X);
140 NEXT
150 PRINT
160 FOR X = 65 TO 90
170 PRINT CHR$(X);
180 NEXT
190 PRINT
200 PRINT#1,""
210 CLOSE 1
```

The CMD command causes all the PRINT commands to act as if they were PRINT#1 commands. Line 200 disables the CMD command.

Notes

Once ENABLEd, the printer will 'masquerade' as an IEEE device until disabled. If you assign the printer a device number of an existing IEEE device, you will not be able to send or receive information from the actual IEEE device. To be safe, always assign the printer a device number which is not assigned to any IEEE device.

If you wish to write your own customized routine to handle various translations (i.e. A mode to format numeric output), you can write the routine as an assembly language subroutine. Please refer to Appendix C for more information.

The ENABLE command operates by changing the *kernal.* The kernal is a set of routines which allow your computer to communicate with the outside world. ENABLE allows the *Vic-Tree* to intercept any information which would be normally sent to a specific IEEE device. Such information is then sent to the parallel user port.

Typing <STOP><RESTORE> will disable the printer. You must then re-enable the printer.

DISABLE

Syntax

DISABLE

Function

The DISABLE command will disable the attached printer. Any references to the device number assigned to the printer will now be sent to an IEEE device. If you wish to use the printer, you must then re-ENABLE it.

Notes

When your computer is first powered ON, the printer will be DISABLEd. DISABLE also turns off the *Cee-Net* local area network.

Editing Commands

Vic-Tree provides 14 new editing commands which enhance the VIC or CBM 64's text editor. They are summarized below and described in the following pages.

AUTO	Enable automatic line numbering
BASIC	Enable BASIC editor
CHAIN	Append program
CHANGE	Find and change text
DELETE	Delete line range
FIND	Find text
LCOPY	Copy line range
LITERAL	Enable word processing editor
LMOVE	Move line range
MERGE	Merge program
OFF	Disable AUTO
PAGE	Interactively LIST program
RENUMBER	Renumber line range
TYPE	Display word processed text

AUTO

Syntax

AUTO [*Starting line*[,*Increment*]]

Function

The AUTO command will automatically type the next line number in sequence when entering program lines. If you do not specify any parameters, AUTO will use the last specified line and increment. A starting line of 100 and increment of 10 will be used if there are no previous parameters. To temporarily exit AUTO, type <RETURN> to a blank line number. To re-start AUTO, type AUTO or enter a new program line. The OFF command will permanently disable AUTO until the AUTO command is re-typed.

Examples

```
AUTO <RETURN>
100 REM NO PREVIOUS PARAMETERS,
120 REM SO AUTO USES 100,10
130 ■
Awaiting input..

AUTO 10,1
10 REM USER SPECIFIED PARAMETERS
11 REM START LINE OF 10, INCREMENT
12 REM OF 1
13 ■
```

Notes

AUTO calculates the next line number in sequence by adding the current increment to the last line num-

ber entered. This feature allows you to skip forward or backwards without having to re-issue the AUTO command. When you exit the AUTO command by typing <RETURN> to an empty line, that line will be deleted from program memory, if present. If you do not wish for the line to be deleted, type <SHIFT><RETURN>.

AUTO 'types' the next line number by placing the appropriate keystrokes for the next line number into a keyboard buffer. The keyboard buffer allows up to 10 keystrokes to be 'saved' while the computer is not ready for input. When the computer becomes ready for input, keystrokes are taken from the keyboard buffer until the buffer becomes empty. If you type extremely fast, you may overflow the keyboard buffer and disrupt AUTO's operation.

BASIC

Syntax

BASIC

Function

The BASIC command is used in conjunction with the LITERAL command to switch between the word-processing and BASIC text editors. When using the BASIC editor, only BASIC statements can be entered into program memory.

Example

BASIC <RETURN>

Notes

The BASIC text editor ignores all shifted characters unless they are enclosed in quotes. In addition, any text which corresponds to BASIC statements is compressed into a one character token. Tokens reduce the space required for BASIC programs.

CHAIN

Syntax

CHAIN [D*Drive*] *Filename*[ON U*Device*]

Where
Drive has a value or 0 or 1
Filename is a string no longer than 16 characters
Device is 1,2,4–31

Function

The CHAIN command will load a program from a disk or tape device and place the program at the end of the program currently in memory. CHAIN does not modify any line numbers, so you must take care that programs are CHAINed in the right order. If you CHAIN from a tape device, do not specify a drive number. If you do not specify a device number, CHAIN will use a device number of 8.

Examples

Program 'FLIP' on tape contains the following lines:

```
900 PRINT "FLIP"
910 RETURN
```

The program currently in memory is:

```
100 GOSUB 900
110 PRINT "FLOP"
120 END
```

After typing:

```
CHAIN "FLIP" ON U1
```

The program in memory is:

```
100 GOSUB 900
110 PRINT "FLOP"
120 END
900 PRINT "FLIP"
910 RETURN
```

Notes

If you CHAIN from a disk and do not specify a drive number, the filename will be sent to the disk device unchanged. This will cause all Commodore disk drives to scan all available drives for the given filename. If you specify a drive number, a drive prefix will be added to the filename causing the disk device to scan only the given drive for the filename.

See the Disk Command Section for more information on disk drives.

Early versions of Commodore BASIC (Level 1, Version 2) contain an error which causes BASIC programs to be saved with an extra character. This extra character causes the CHAIN and MERGE commands to become confused. In order to eliminate the extra character from such programs, follow the instructions below:

1. Load the program; type

```
A=PEEK(45)+PEEK(46)*256:A=A-1
POKE 46,A/256
POKE 45,(A/256-INT(A/256))*256
```

2. Save the program

CHANGE

Syntax

CHANGE *Search* ,*Replace* [,R] [,*Line-Range*]

Function

CHANGE will search a given range of lines for *Search* text and replace all occurences of such text with *Replace* text. If you do not specify a line range, the entire program will be searched. CHANGE allows you to 'double-check' all changes with the R option. If you specify ,R, all changes must be confirmed before they are enacted. The command will display the change before replacement and wait for your response. If you type <RETURN>, the change will occur–any other key will skip the current replacement. If you wish to stop, press the <STOP> key.

The *Search Text* may be either BASIC text or literal text. BASIC text is not enclosed in quotes while literal text is enclosed in quotes. If your search text is BASIC text, your replacement text must also be BASIC text.

Example

The following program is in memory

```
100 PRINT "DEMONSTRATION PROGRAM"
110 FOR X=1 TO 10
120 PRINT "X PLUS ONE IS";X+1
130 NEXT
```

To change all BASIC + *statements to* - *statements, type:*

```
CHANGE +,-
```

The program has now been changed to:

```
100 PRINT "DEMONSTRATION PROGRAM"
110 FOR X=1 TO 10
120 PRINT "X PLUS ONE IS";X-1
130 NEXT
```

Now change the literal text

```
CHANGE "PLUS","MINUS"
```

The program has now been changed to:

```
100 PRINT "DEMONSTRATION PROGRAM"
110 FOR X=1 TO 10
120 PRINT "X MINUS ONE IS";X-1
130 NEXT
```

Notes

CHANGE operates by FINDing text and then replacing text. Whenever text matches the search text, it is displayed in reverse video. Reverse video will only appear with BASIC statements; literal text will not be reversed.

DELETE

Syntax

DELETE *Line-Range*

Function

The DELETE command will delete a given set of lines from program memory. If the *Line-Range* does not include any lines, no lines will be deleted.

Example

The following program is in memory

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
130 PRINT "X=";
140 PRINT X
150 REM LINES 130,140 SHOW X'S VALUE
160 NEXT
170 END
```

To delete lines 130 through 150, inclusive, type:

```
DELETE 130-150
```

The program has been changed to:

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
160 NEXT
170 END
```

Notes

The DELETE command works by finding the starting line address in memory and the ending line address in memory. The 'hole' between the line before the first line and after the last line is closed up by moving up all text to close up the hole. After you DELETE lines, there is no way to recover the lost lines, so take care in specifying open-ended line ranges such as 100– (Which would delete the entire program in the previous example).

FIND

Syntax

FIND *Search string*[,*Line-Range*]

Function

The FIND command will search for either BASIC or literal text. BASIC text consists of all regular BASIC expressions; literal text must be enclosed inside quotes. Whenever the search string is found, the line which contains the search string will be printed. If there is more than one match on any one line, FIND will only print the line once.

Examples

The following program is in memory:

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
130 PRINT "VALUE IS";
140 PRINT X
150 REM SHOW OUTPUT
160 NEXT X
170 END
```

To find all occurences of PRINT *type:*
```
FIND PRINT
```

FIND *displays:*
```
110 PRINT "START TEST"
130 PRINT "VALUE IS";
140 PRINT X
```

To find all occurences of the literal string, "VALUE IS", *type:*
```
FIND "VALUE IS"
```

FIND *displays:*
```
130 PRINT "VALUE IS"
```

To find all occurences of the statement, PRINT X, *type:*
```
FIND PRINT X
```

FIND *displays:*
```
140 PRINT X
```

LCOPY

Syntax

LCOPY *Source Line-Range, Destination Line*

Function

The LCOPY command will copy a line or group of lines to a new location in memory. The *source* line-range must start and end on existing line numbers. All references to the lines being moved will remain unchanged. That is, any GOTO's, GOSUB's, etc. will still point to the old line numbers. The copied lines will start at the given *Destination Line* and proceed upwards with an increment of one. If, at any time, LCOPY runs out of line numbers for the new lines, the command will stop and display :

OUT OF SPACE

You should then RENUMBER your program and copy the remaining lines.

Example

The following program is in memory:

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
130 PRINT "X=";
140 PRINT X
150 REM PRINT VALUES
160 NEXT X
170 GOTO 120
180 END
```

To copy lines 120 through 160 to line 500, type:

```
LCOPY 120-160,500
```

The program will now look like this

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
130 PRINT "X=";
140 PRINT X
150 REM PRINT VALUES
160 NEXT X
170 GOTO 120
180 END
500 FOR X=1 to 10
501 PRINT "X=";
502 PRINT X
503 REM PRINT VALUES
504 NEXT X
```

Note that the GOTO *statement on line 170 remains unchanged.*

Notes

The LCOPY command operates by loading each line to be copied into an intermediate buffer, assigning the line a new line number, and re-inserting the line into program memory. The process can take quite long if a large group of lines are copied to a location within the program.

LITERAL

Syntax

LITERAL [*Starting Line*[, *Increment*]]

Function

The LITERAL command enables the word processing editor. When in this mode, both upper and lower case characters can be entered into program memory. When in LITERAL mode, all immediate mode commands remain in effect. You may not RUN edited text, however, as the word processing editor does not properly store BASIC commands. When used with the TYPE command, you can easily print letters, memos, and reports.

The LITERAL command also acts as an AUTO command. If you do not specify any parameters when first entering literal mode, the starting line will be set to 100 and the increment will be set to 10. Once in literal mode, the LITERAL command acts just the same as the AUTO command.

Example

The □*character corresponds to the* <SHIFT> <SPACE> *character.*

```
LITERAL
100 □□Congratulations on your purchase
110 of the VIC-Tree programming module!
120 The VIC-Tree is a feature packed
130 toolbox of powerful commands that
140 allow you to program your VIC or
150 CBM 64 more efficiently and easily.
```

```
160 ■
```

Press <RETURN> *to stop automatic line numbering.*

```
TYPE
  Congratulations on your purchase
of the VIC-Tree programming module!
The VIC-Tree is a feature packed
toolbox of powerful commands that
allow you to program your VIC or
CBM 64 more efficiently and easily.
```

Notes

The LITERAL command replaces BASIC's conventional 'tokenizing' subroutine. Normally, the tokenizing subroutine 'crunches' all BASIC keywords into single character tokens. When in literal mode, the tokenizing subroutine does not crunch BASIC keywords and allows shifted characters (Graphics or Upper Case) to be stored in program memory.

When indenting a line, use the <SHIFT> <SPACE> character instead of the normal unshifted space. For example:

```
LITERAL
100 □□This line is indented
110 This line is not..
```

Where the □ character is the <SHIFT> <SPACE> character.

LMOVE

Syntax

LMOVE *Source Line-Range, Destination Line*

Function

The LMOVE command will move a line or group of lines to a new location in memory. All references to the lines being moved will be updated. That is, any GOTO's, GOSUB's, etc. will point to the new line numbers. The copied lines will start at the *Destination Line* and proceed upwards with an increment of one. If, at any time, LMOVE runs out of line numbers for the new lines, the command will stop and display :

OUT OF SPACE

You should then RENUMBER the program and move the remaining lines.

LMOVE requires that the *source* line-range start and end on existing lines.

Example

The following program is in memory:

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
120 FOR X=1 TO 10
130 PRINT "X=";
140 PRINT X
150 REM PRINT VALUES
160 NEXT X
170 GOTO 120
180 GOTO 120
```

To move lines 120 through 170 to line 500, type:

```
LMOVE 120-170,500
```

Program will now look like this

```
100 REM DEMO PROGRAM
110 PRINT "START TEST"
180 GOTO 500
500 FOR X=1 to 10
501 PRINT "X=";
502 PRINT X
503 REM PRINT VALUES
504 NEXT X
505 GOTO 500
```

Note that the GOTO *statements on lines 180 and 505 have been updated.*

Notes

The LMOVE command operates by loading each line to be moved into an intermediate buffer, assigning the line a new line number, and re-inserting the line into program memory. After the new line is inserted into program memory, all references to the old line are changed to the new line. When all lines have been moved to their new locations, the old lines are then DELETEd from memory.

MERGE

Syntax

MERGE [D*Drive*] *Filename* [ON U*Device*]

Where
Drive has a value or 0 or 1
Filename is a string no longer than 16 characters
Device is 4–31

Function

The MERGE command will load a program from a disk device and interleave the lines with existing lines in program memory. If a line to be loaded already exists in program memory, the line in program memory will remain unchanged and the line in question will be printed. If you do not specify a device number, MERGE will use device number 8.

Examples

Program 'FLIP' on disk contains the following lines:

```
115 PRINT "FLIP-FROM DISK"
125 REM END OF FLIP
```

The program currently in memory is:

```
100 REM START
110 PRINT "FROM MAIN"
120 PRINT "FROM MAIN"
130 END
```

After typing:

```
MERGE "FLIP"
```

The program in memory is:

```
100 REM START
110 PRINT "FROM MAIN"
116 REM END OF FLIP
120 PRINT "FROM MAIN"
125 REM END OF FLIP
130 END
```

Notes

If you MERGE from a disk device and do not specify a drive number, the filename will be sent to the disk device unchanged. This will cause all Commodore disk drives to scan all available drives for the given filename. If you specify a drive number, a drive prefix will be added to the filename causing the disk device to scan only the given drive for the filename.

Program stored on cassette can not be MERGEd.

Early versions of Commodore BASIC (Level 1, Version 2) contain an error which causes BASIC programs to be saved with an extra character. This extra character causes the CHAIN and MERGE commands to become confused. In order to eliminate the extra character from such programs, follow the instructions below:

1. Load the program; type:

```
A=PEEK(45)+PEEK(46)*256:A=A-1
POKE 46,A/256
POKE 45,(A/256-INT(A/256))*256
```

2. Save the program

OFF

Syntax

OFF

Function

The OFF command disables automatic line numbering (AUTO) and program tracing (TRACE). To re-enable these functions, you must issue the AUTO or TRACE commands.

If you wish to temporarily suspend automatic line numbering, press <SHIFT> <RETURN> after AUTO types a line number. Entering any program line will re-enable automatic line numbering.

PAGE

Syntax

PAGE [*List Control*]

Where *List Control* is:

<RETURN> To skip forward through a program
= <RETURN> To skip backwards through a program
↑ <RETURN> To jump to the beginning of a program
x <RETURN> To jump to a specific line in a program

Function

The PAGE command allows you to skip through a program listing. Each time PAGE is executed, a group of program lines will be listed. The number of lines listed is always adjusted so that no lines will scroll off the top of the screen. For your convenience, the PAGE command will be re-typed at the bottom of the screen if you wish to re-execute the command. PAGE allows four parameters which control which program lines shall be listed. If you only type <RETURN>, the next 'page' of program lines will be listed. The next page starts with the line which is currently four lines ahead of the top line displayed. If you type = <RETURN> the previous page of lines shall be displayed. To display the first page of a program, type ↑ <RETURN> and to move to a specific line in memory, type *x* <RETURN> where *x* is an existing program line.

To exit from PAGE, type <CURSOR-DOWN> and then <RETURN>.

Notes

When moving forward or backwards through a program, PAGE calculates the starting line of the next page to be displayed by adding or subtracting four lines from the current starting line. If you wish to change this offset, POKE location 49246 (28356 on the VIC) with the offset of your choice. Make sure that the offset is not too large, as PAGE will then skip program lines!

RENUMBER

Syntax

RENUMBER [*Start*[,*Increment*[,*Line-Range*]]]

Function

The RENUMBER command will renumber a set of lines in a program. The renumbered lines will begin with the *start* parameter and proceed upwards at the given *increment.* All references to the lines which are renumbered will be changed accordingly. If you do not specify any parameters, RENUMBER will provide a new starting line of 100, an increment of 10, and a line range which encloses the entire program.

Due to memory limitations of the VIC, RENUMBER has been designed to very memory efficient. This design, however, causes the RENUMBER command to be quite slow if your program has many GOTO's.

Example

The following program is in memory:

```
1 REM SQUARE ROOTS FROM 1 TO 10
5 REM
10 FOR X = 1 TO 10
12 PRINT "VALUE:";
13 PRINT X,
15 PRINT "ROOT:";
20 PRINT SQR(X)
25 NEXT
26 GOTO 1
30 END

RENUMBER 100

100 REM SQUARE ROOTS FROM 1 to 10
110 REM
120 FOR X = 1 TO 10
130 PRINT "VALUE:";
140 PRINT X,
150 PRINT "ROOT:";
160 PRINT SQR(X)
170 NEXT
180 GOTO 100
190 END
```

TYPE

Syntax

TYPE [LM:*Left Margin*] [*Line-Range*]

Function

The TYPE command will LIST a word-processing document without line numbers. The left margin will be at column one unless you specify otherwise. If you specify both a line margin and a line range, you must separate the two parameters with a comma.

If you want to TYPE to a printer or other IEEE device, enter:

```
OPEN 1,Device Number
CMD 1
```

before executing TYPE.

Example

The following document is in memory:
*(*LITERAL *mode)*

```
100 □□Congratulations on your purchase
110 of the VIC-Tree programming module!
120 The VIC-Tree is a feature packed
130 'Toolbox' of powerful commands that
140 enable you to program your VIC or
150 CBM 64 more efficiently and easily.
```

TYPE *will display the document like this:*

```
  Congratulations on your purchase
of the VIC-Tree programming module!
The VIC-Tree is a feature packed
'Toolbox' of powerful commands that
enable you to program your VIC or
CBM 64 more efficiently and easily.
```

Blank

De-Bugging Commands

De-Bugging commands help you to locate and diagnose errors in your program. *Vic-Tree* provides three debugging commands. They are summarized below and described in the following pages.

DUMP . . . . . Display non-array variables
HELP . . . . . Locate error in program
TRACE . . . Display program lines prior to execution

DUMP

Syntax

DUMP

Function

The DUMP command will display all non-array variables which currently have a value. The variables are displayed so you can easily change their values if you wish.

Example

The following program is in memory:

```
100 A$ = "THIS IS A STRING"
110 X = 132.69
120 X% = 10
130 Y(1) = 1
140 END
```

After RUN*ning the program,* DUMP *will display:*

```
A$ = "THIS IS A STRING"
X = 132.69
X% = 10
```

Note that the variable Y *is an array variable and is not displayed.*

Notes

If you wish to change the value of any of the variables which are displayed, merely move the cursor to the desired variable and change its value. Press <RETURN> to store the new value.

HELP

Syntax

HELP

Function

The HELP command is used to pinpoint the location of an error when it occurs during program execution. HELP will list the erroneous line and hi-light the approximate location where the error occured.

Example

The following program is in memory:

```
100 X = 10
110 Y = 0
120 Z = X/Y
```

When the program is RUN*, the VIC displays:*

```
?DIVISION BY ZERO ERROR IN 120
```

To pinpoint the location of the error, type:

```
HELP
```

and the following line will be displayed:

```
120 Z = X/□
```

The Y *will be in reverse field. This tells you that* Y *caused a division by zero error. A* DUMP *command following the* HELP *command further assists in troubleshooting.*

Notes

The HELP command will not always pinpoint the

exact location of an error. Sometimes, the error will be one statement to the left or right of the hi-lighted position. If there is no hi-lighted position, the error will usually be in the very first statement of the line.

If you stop a running program with the <STOP> key, HELP will tell you the approximate statement which was being executed when the program was stopped.

The HELP command must be used immediately after an error occurs.

The HELP command can only be used once after an error occurs. If you wish to see the error again, you must re-RUN your program.

TRACE

Syntax

TRACE [*Delay*]

Function

The TRACE command lets you see a program execute step by step. Each program line is displayed on the top lines of the screen before execution. If there is more than one statement on the line, the next statement to execute will be hi-lighted. After a line is displayed on the screen, the *Vic-Tree* will wait for either of three responses. If you press the <SHIFT> key, TRACE will step forward one statement and wait for a selected *Delay* before proceeding forward again. If you press the <CBM> key, TRACE will step forward with no delay. Lastly, if you press the STOP key, TRACE will stop the program.

The *Delay* slows down the execution of your program so you can read each line before it executes. If you do not specify any delay, a delay of 128 will be used (Approximately 1 second). The smaller the delay, the faster the program will execute. In order to 'skip' over sections which you do not want to TRACE, press the <CBM> key to step forward at maximum speed.

TRACE remains ON until you turn if OFF with the OFF command.

Notes

TRACE will not operate properly if you are RUNing a program which uses high-resolution graphics. In high-resolution mode, TRACE is unable to properly display program lines on the screen.

Blank

Appendix A

Installation Instructions

Make sure that your VIC or CBM 64 and all attached devices are OFF. Insert the module, label up, into the right hand expansion slot (When viewed from above). Turn on the computer. If the computer fails to display anything, turn off the computer and make sure that the module is not upside-down.

Troubleshooting

1. Output to printer is all on one line. (Overprinted). Your printer requires a line-feed character at the end of each line. Use a logical file number larger than 127.

If you send a line which is longer than the printer's available line width, some printers will ignore the extra characters, or fail to advance to the next line before printing the extra characters. In such cases you must change your program so that no lines are printed which are longer than the printer's lines. Some printers allow you to set certain switches which will eliminate the problem. Look at the printer manual for more information.

Blank

Appendix B
Index and Quick Reference

Disk Commands

Printer Commands

Editing Commands

De-Bugging Commands

Vic-Tree Control

Appendix C
Technical Reference

Introduction

VT1 (VIC)

The *VIC-Tree* is a coordinated hardware and software module which includes eight kilobytes of read only memory (ROM) and four kilobytes of random access memory (RAM). Of the four kilobytes of RAM, approximately one kilobyte is used for *VIC-Tree* commands. The remaining memory is used for *Cee-Net* communication buffers. The ROM resides at locations B000–BFFF and 7000-7FFF hexadecimal. The RAM resides at locations 6000-6FFF hexadecimal.

VT64 (CBM 64)

For the CBM 64, the *VIC-Tree* consists of an eight kilobyte ROM which resides at locations 8000–9FFF hexadecimal. Four kilobytes of RAM are used starting at location C000 hexadecimal. This configuration allows for a maximum of 32 Kilobytes of BASIC program storage. Programs which use memory locations C000-CFFF hexadecimal will interfere with *VIC-Tree* 's operation. *VIC-Tree* should be KILLed before running such programs.

The *VIC-Tree* provides routines which allow any Centronics standard printer to be attached to the VIC's or CBM 64's parallel user port. The interface protocol uses the –BUSY line for handshaking.

Method of Implementation

The *VIC-Tree* adds a total of 42 commands to the BASIC language. The *VIC-Tree* adds these commands by adding tokenizing and executive subroutines which recognize the new commands. Memory locations 0300-030B (hexadecimal) are modified in order to add the additional subroutines. The subroutines are described below.

0300–0301 Error Message Link

This location contains the address of the code which is executed whenever BASIC finds an error. *VIC-Tree* adds a routine which aids the HELP command to pinpoint your errors.

0302–0303 BASIC Warm Start Link

This location contains the address of the code which is executed when you press the <STOP><RESTORE> keys. *VIC-Tree* adds a routine which resets its scratchpad memory.

0304–0305 Crunch BASIC Tokens Link

This location contains the address of the code which is executed when BASIC tokenizes an input line. *VIC-Tree* adds a routine which tokenizes the new commands. The *VIC-Tree* tokenizer is executed *after* all other tokenizers.

0306–0307 Print Tokens Link

This location contains the address of the code which is executed when the LIST command needs to print a token. *VIC-Tree* adds a routine to print tokens for the new commands and also to enhance tokens for the FIND, HELP, CHANGE, and TRACE commands.

0308–0309 Start New BASIC Code Link

This location contains the address of the code which is executed when BASIC needs to execute a token. *VIC-Tree* adds a routine which executes the tokens for the new commands. In addition, a routine is added when you TRACE a program.

030A–030B Evaluate Expression Code Link

This location contains the address of the code which is executed when BASIC needs to evaluate a numerical or string expression. *VIC-Tree* adds a routine which checks for the disk status variables, DS$ and DS and updates the variables if necessary.

0318–0319 Non-Maskable Interrupt Vector

This location contains the address the code which is executed when a non-maskable interrupt occurs. When the *Cee-Net* is enabled, *VIC-Tree* adds a routine which allows the computer to receive data packets from the *Cee-Net* data buss.

031A–0325 Kernal I/O Vectors

These locations contain vectors to the kernal functions which perform operations on logical files. *VIC-Tree* adds routines which detect and process *Cee-Net* and printer commands.

Tokens

The *VIC-Tree* tokenizes all the new commands into a two token format. The first token is a 'prefix' token; the second token corresponds to the new command. The

prefix token has the value of the GO token.* The second token may be from 128 to 255. The token value of 164 (decimal) is reserved for the GO TO command. This method of tokenizing has the advantage that *VIC-Tree* commands do not interfere with the *Super Expander* or any other utility module. A table the token values is shown on the next page.

*BASIC allows two forms of the GOTO command. The first form is expressed as one word (GOTO) and is compressed into one token. The second form is expressed as two words (GO TO) and is compressed into two tokens.

VIC-Tree Tokens

All tokens are preceded with the token value 203 (decimal).

128	AUTO
129	OFF
130	LITERAL
131	BASIC
132	TYPE
133	KILL
134	DUMP
135	DELETE
136	HELP
137	RENUMBER
138	LCOPY
139	LMOVE
140	TRACE
141	FIND
142	CHANGE
143	PAGE
144	ENABLE
145	DISABLE
146	CONCAT
147	DOPEN
148	DCLOSE
149	RECORD
150	HEADER
151	COLLECT
152	BACKUP
153	COPY
154	APPEND
155	DSAVE
156	DLOAD
157	CATALOG
158	RENAME
159	SCRATCH
160	DIRECTORY

Additional Expansion

If you intend to add other modules, keep in mind the following precautions:

All *VIC-Tree* patches to the BASIC links memorize the previous link. This allows you to use more than one module at a time. Normally, the *VIC-Tree* should be the last module to modify the BASIC links. If you design your own module, make sure that your commands are not subsets of any of *VIC-Tree* 's commands. For example, RENUM would be a subset of *VIC-Tree* 's RENUMBER command.

VIC-Tree should be the *last* module which is initialized when the VIC or CBM 64 is powered ON.

The KILL command will remove all *VIC-Tree* patch vectors from BASIC and the Kernal; the patch vectors will be restored with the vectors which were resident previous to *VIC-Tree* 's initialization.

@@ command

The @@ command allows you to expand the BASIC language by writing new commands and saving them on disk. All extension commands must be preceded with the @@ characters. You must set aside a portion of memory for an extension command to be loaded. This will usually be at the very top of the memory. The extension command should be written as an assembly language subroutine. If the command requires parameters, you may use the CHRGET subroutine at location 0073 (Hexadecimal) to fetch characters from the BASIC line. CHRGET will return the next non-blank character in the accumulator. Make sure to update CHRGET to the end of a line or : before ending the command. Otherwise, the executive will return a ?SYNTAX ERROR. The last two bytes of the extension command must point to the

starting address of your code. These bytes must be the last two bytes loaded from the disk. The address is stored low byte first. The extension command should be stored on disk with the filename corresponding to the extension command.

A simple example of an extension command would be a command which places a given character in the first 255 screen positions. The command, named FILL would have the following syntax:

@@FILL *Character*

The program would be as follows: *(For CBM 64)*

```
START LDY #$FF
LOOPS STA $03FF,Y
DEY
BNE LOOPS
JSR $0073
RTS
.WOR START
```

The last two bytes of the program contain the starting address of the extension command. When saving your extension program on disk, you must take care to include such bytes in your file.

When the command is first started, the accumulator contains the first non-space character after the extension command. This character is used to fill the first 255 bytes of screen memory. After the screen memory is filled, the instruction, JSR **$0073** is used to advance BASIC's text pointer. This prepares BASIC for the next BASIC command to execute.

All extension commands are called as subroutines.

End an extension command with the RTS instruction.

This extension command would be saved on a diskette under the name of FILL. When choosing names for your extension commands, pick names which do not have BASIC or *VIC-Tree* command names imbedded within the extension command. For example, the extension command, BLOAD, has the BASIC command LOAD imbedded within the extension command. If you really must have an imbedded command within your extension command, the filename for the extension command must be named after the tokenized form of the command. That is, BLOAD would be stored on disk as B*X* where *X* corresponds to the token value for the LOAD command.

Once an extension command has been loaded into memory, it will remain in memory until another extension command is used. This eliminates the need to load the extension command from the disk each time it is used.

Compatiblity with CBM BASIC 4.0

All *VIC-Tree* Disk commands are syntax compatible with CBM BASIC 4.0. There are a few operational differences:

1. The R parameter in the DOPEN and DSAVE statements is not supported in CBM BASIC. Some disk units are unable to support this function correctly, so you should manually SCRATCH a file before creating it again.

2. The CMD parameter has been added to the DLOAD and EXECUTE commands in order to allow you to load non-relocatable programs.

3. The DLOAD and EXECUTE commands will scan all available disk drives if no drive number is specified.

CBM BASIC 4.0 will only scan drive zero.

4. The DIRECTORY command allows you to specify a filename if you wish to selectively list the diskette directory. CBM BASIC 4.0 does not allow a filename entry; the entire directory is always displayed.

5. CBM BASIC 4.0 will always assume a disk device of eight, a drive of zero, and a secondary command of zero if these parameters are not included in the command. *VIC-Tree* allows you to change these parameters with the USE command.

Memory Maps

The *VIC-Tree* 's scratchpad is organized as follows:

(Hexadecimal Addresses)

C000 . . . . . Error Link Save
C002 . . . . . Warm Start Link Save
C004 . . . . . Crunch Tokens Link Save
C006 . . . . . Print Tokens Link Save
C008 . . . . . Execute Token Link Save
C00A . . . . . Evaluate Expression Link Save
C00C . . . . . Prefix Token Flag
C00D . . . . . Read disk error link
C00F . . . . . Send disk command link
C011 . . . . . Print directory link
C013 . . . . . Cee-Net Packet processing
C015 . . . . . New Kernal I/O Flag
C016 . . . . . Old Kernal I/O Addressed
C036 . . . . . Cee-Net enabled flag
C037 . . . . . Printer enabled flag
C038 . . . . . Trap output flag
C039 . . . . . Trap input flag
C03A . . . . . Cee-Net trapping mode
C03B . . . . . Printer device number
C03C . . . . . Printer Mode
C03D . . . . . Translation Vector
C03F . . . . . Cee-Net scratch
C040 . . . . . Cee-Net OPEN patch
C042 . . . . . Cee-Net CLOSE patch
C044 . . . . . Cee-Net output patch
C046 . . . . . Cee-Net input patch
C048 . . . . . Current AUTO line
C04A . . . . . Current AUTO increment
C04C . . . . . AUTO patching address
C04E . . . . . AUTO enabled flag
C04F . . . . . LITERAL enabled flag
C050 . . . . . LITERAL patching address
C052 . . . . . LITERAL patching address
C054 . . . . . HELP pointer
C056 . . . . . HELP line
C058 . . . . . TRACE speed

C059 . . . . . TRACE enabled flag
C05A . . . . . TRACE patching address
C05C . . . . . Current PAGE
C05E . . . . . PAGE increment
C05F . . . . . Default disk device
C060 . . . . . Default Second address
C061 . . . . . Default disk drive
C062 . . . . . Reserved logical File number
C063 . . . . . Status variable update flag
C064 . . . . . Extension command scratch
C065 . . . . . Extension command starting address
C067 . . . . . Extension command name
C079–C09D . . . . . Scratchpad
C11E–C17F . . . . . DOS filename buffers/ flags
C17F–C39F . . . . . Cee-Net storage
C39F–CFFF . . . . . Cee-Net buffers

These addresses are for the CBM 64. To obtain addresses for the VIC, change the most significant digit from 'C' to '0' and add '6E80' to all addresses. *Cee-Net* storage and buffers reside in the low portion of scratch memory starting at 6000 (hexadecimal).

If you are not using *Cee-Net* on the CBM 64, you may use locations C17F to CFFF (Hexadecimal) for your own use. If you not using *Cee-Net* on the VIC, your BASIC program memory will end at location 6E7F.

Printer Interface

The printer interface operates by adding routines to the VIC's kernal. The new routines 'trap' all IEEE commands to the ENABLEd printer. The interface routines provide for appropriate translation for various modes. You may write your own translation routine. The routine must be an assembly language subroutine. The translation routine will be called when the secondary address is either 0 or 2. The routine is passed the character to print in the accumulator. You may do the appropriate translation and return the new character in the accumulator. Locations C03D and C03E (Low and High bytes of address, respectively) must point to your translation routine.

Cee-Net Local Area Network

The *Cee-Net* local area network is a powerful networking system which allows up to 64 VIC's or CBM 64's to be interconnected via a twisted cable interface buss. Carrier sense multiple access protocols are used to abitrate buss activity. In addition, collision and error detection are used to minimize transmission failures. Information is grouped into data packets in order to maximize throughput and response time.

Each computer is assigned a network address. By assigning certain duties to computers with specific network addresses, users may share the resources of a disk drive or printer.

VIC-Tree includes subroutines which provide packet level communication between network nodes. When the ATTACH command is invoked, *VIC-Tree* attempts to establish communication with a specific network server in order to load high-level subroutines which provide a BASIC interface to the network. If no server is present, the user is informed that the network is inoperative.

Once the network is enabled, users have the option of specifying which BASIC device numbers correspond to network addresses and which device numbers correspond to local devices such as a 1541 disk drive. If the mode is zero, network addresses are from 4–127 and local devices are from 128-255. That is, a local device which has an actual device number of 8 would be accessed with a device number of 128+8=136. If the mode is 128, the mapping is swapped.

For more information on the *Cee-Net* , please refer to the *Cee-Net* operator's manual.