HesWare™

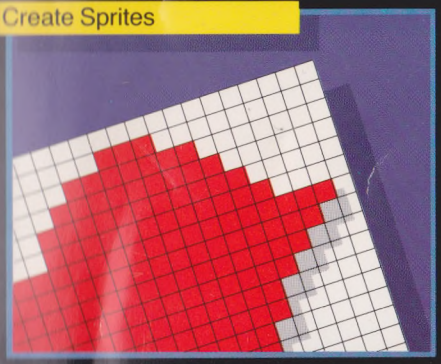
Graphics BASIC™

An Enhanced BASIC

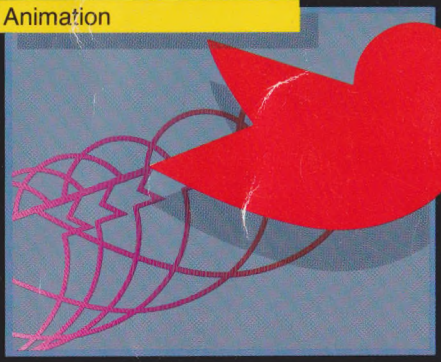
By Ron Gilbert and Tom McFarlane

Unlock the full potential of your computer with over 100 additional English-language commands. Now beginning programmers can create complex graphics, animation, and music without being a machine language programming expert.

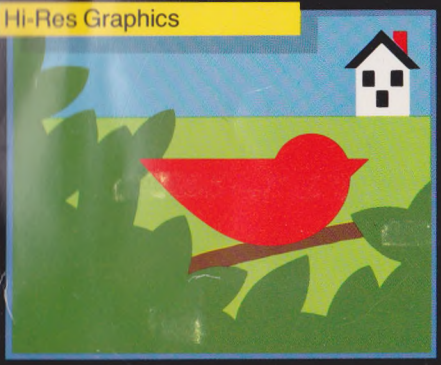
Create Sprites



Animation



Hi-Res Graphics



Synchronized Sound

Diskette for Commodore 64

Graphics BASIC

By

Ron Gilbert and Tom McFarlane

COPYRIGHT NOTICE

Copyright 1984 by Human Engineered Software Corporation. All rights reserved. No part of this publication may be reproduced in whole or in part without the prior written permission of H.E.S. Unauthorized copying or transmitting of this copyrighted software on any medium is strictly prohibited.

Although we make every attempt to verify the accuracy of this document, we cannot assume any liability for errors or omissions. No warranty or other guarantee can be given as to the accuracy or suitability of this software for a particular purpose, nor can we be liable for any loss or damage arising from the use of the same.

The software contained in this package was designed to work on the computer system designated on the package. H.E.S. is unable to guarantee that this product will work properly on systems that include other parts. Compatibility is determined by the companies that manufacture computer hardware and peripherals, not by H.E.S.

GRAPHICS BASIC is a TM of Human Engineered Software. Commodore 64 is a registered TM of Commodore.

CONTENTS

INTRODUCTION

Graphics BASIC gives you and your Commodore 64 a powerful extended BASIC. You have over 100 new commands and features combining incredible power and flexibility with simplicity and readability. Graphics BASIC complements the existing BASIC in your Commodore 64 to support its sound and graphic capabilities as well as many other features. These extended commands cover several areas: sprites, sound, high-resolution and multicolor graphics, programming aids, and input/output management.

NOTE: If you write a program in Graphics BASIC, you must have Graphics BASIC loaded into your Commodore in order for the program to run.

How To Use This Manual

The Graphics BASIC manual consists of two sections: a tutorial and a command reference.

The tutorial section demonstrates some of the special features of Graphics BASIC. The tutorial section is written for programmers already familiar with BASIC, however beginners will absorb some programming concepts by following the examples and studying the demo programs on the disk.

The reference section is divided by category (such as sprites, programming aids, and sound) and the commands are listed alphabetically within each category.

You can explore the features of Graphics BASIC in the tutorial, the reference section, and the demo programs. You can run the demo programs, list them, modify a line, change a number or word, and run the program again to see the effect of your changes.

Getting Started

What you need to use Graphics BASIC

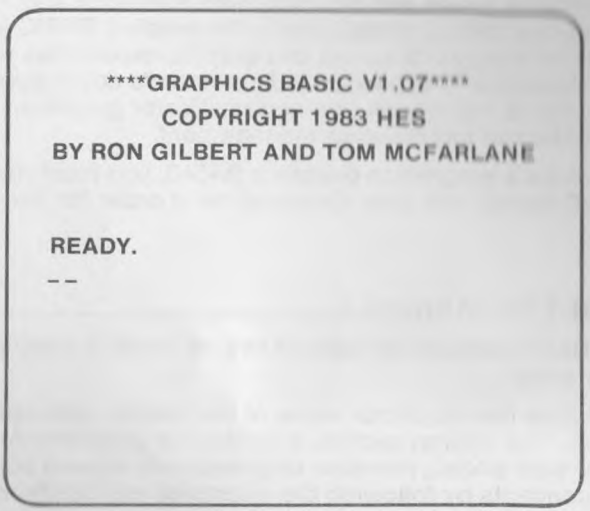
- Commodore 64
- Commodore 1541 disk drive
- Graphics BASIC disk
- formatted disk for program storage
- monitor or TV, preferably color
- VIC 1525 or Gemini printers, optional

Loading Graphics BASIC

1. Turn on the disk drive, the TV or monitor, and the Commodore 64.
2. Insert the Graphics BASIC disk into the disk drive with the label facing up and entering the drive last.
3. Type: **LOAD"*",8** and press RETURN.
4. At the ready prompt, type **RUN** and press RETURN.
5. When you are finished for the day, remove the Graphics BASIC or program storage disk from the drive before turning off the drive and computer.

First Impressions

Once Graphics BASIC is loaded into your computer, the following screen appears:



You have approximately 20K of memory remaining to create programs using Graphics BASIC.

Note: Some older models of the Commodore 64 may cause flickering of the screen.

Loading the Demo Programs

The Graphics BASIC disk includes several demo programs which show the features and functions of various commands. You can load the demos, list the programs, and make changes. You can then run the demos again and watch the results of your changes.

The DRIVING DEMO shows Graphics BASIC's sprite animation capabilities. To load the DRIVING DEMO, type:

LOAD"DRIVING DEMO",8 and press RETURN.

To run the demo, type **RUN** and press RETURN. A yellow car cruising down a country road should appear on the screen.

To list the program, first press the RUN/STOP key to stop the program. Then type **TEXT** and press RETURN. Type **LIST** to display the program listing.

Note: If you prefer to work with the program listing without trees and clouds floating by, press f4 (shift and f3) and press RETURN. This turns the sprites off.

Using the CRSR movement keys, change some of the values or colors. For example, in lines 250 to 320, you can alter the speed of the various sprites. Negative or positive numbers determine the direction the sprite will move. Press RETURN after each change to enter it. Type **RUN** again to see the differences your changes have made.

Several demo programs are included and a complete list appears on page 102. Also, the command reference includes names of demo programs which illustrate the command being described. Load some other demos and learn Graphics BASIC by experimenting, or go on to the tutorial.

A Few Commands and Concepts to Know

Graphics BASIC operates in three screen modes: text, hires (high resolution), and multi (multicolor). In text mode, you list and edit your programs. In hires or multi mode, you display graphics or run programs. You can switch easily from one mode to another by typing one of three commands:

TEXT switches from a hires or multi screen to text

HIRES switches from a text or multi screen to high resolution screen

MULTI switches from a text or hires screen to a multicolor screen

These three modes are discussed in detail in the tutorial section. In addition, Graphics BASIC allows you to split a screen into two parts. You can have text on one part of the screen and graphics on another. For example, the command TEXT FROM 20 will display text on the bottom part of the screen.

RESET

The command RESET sets all the graphics, sprites, and sound functions to normal and puts the screen into TEXT mode. You could use RESET at the beginning of a program to clear the screen and ensure that previous values and settings will not interfere in the running of the program. You can use the RESET command in a program line, or before you run a program. RESET also comes in handy when you find yourself in a hopeless mess. RESET will NOT destroy the program in memory.

The commands NEW and LIST work as in standard BASIC. Typing NEW destroys the program in memory. Typing LIST prints the current program in memory on the screen if you are in TEXT mode.

All of the commands in Graphics BASIC work in direct mode, (typing them directly without line numbers) and most work in program mode (with line numbers).

Function Keys

The function keys in Graphics BASIC have been designed to simplify and speed up your programming. The list below shows what each function key has been programmed to perform:

KEY	COMMAND PERFORMED	KEY	COMMAND PERFORMED
f1	RUN <RETURN>	f5	DIR <RETURN>
f2	BACKGROUND	f6	KEY LIST
f3	LIST <RETURN>	f7	TEXT <RETURN>
f4	SPRITE OFF	f8	DISK

Each of the commands will be performed when you press the corresponding key if no program is running. In addition, you can change the actions the function keys perform. (See the command reference for KEY, KEY LOAD, KEY SAVE, KEY ON, and KEY OFF.)

GRAPHICS BASIC TUTORIAL

The examples given throughout this tutorial section demonstrate the major features of Graphics BASIC. Type in the example exactly as shown. You can then list and run each sample and watch the magic appear on your screen. The tutorial is designed to show you how easy it is to create complex graphics and sound on your Commodore 64. For detailed explanations of the commands, see the command reference section.

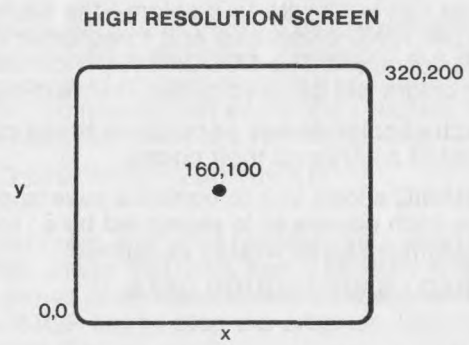
NOTE: If you are unfamiliar with BASIC, study your Commodore 64 User's Guide before proceeding with Graphics BASIC.

Graphics

What are HIRES and MULTI?

The Commodore 64 operates in two graphics modes: high-resolution (HIRES—pronounced hi-rez) and multicolor (MULTI). The graphics screen is made up of a series of dots or pixels. You create graphics by controlling the status of each dot on the graphics screen, that is, turning the dot on or off. To control each dot, you must know or set its location. The screen is divided up into a grid much like a piece of graph paper consisting of x and y coordinates. A dot's location is determined by the intersection of the x and y coordinates. The x coordinate indicates the horizontal positioning across the screen and the y coordinate indicates the vertical positioning up and down the screen. By using these locations you can control the dots—on or off—and you can position objects—such as sprites—at various locations.

The high resolution screen is made up of 320 dots horizontally and 200 dots vertically.



The dot at the center of the screen has the location 160,100.

In hires mode, you can control each dot on the screen and you can use two colors. In multicolor mode, you can use up to four colors (the background color and three others), but you lose half of the horizontal resolution. In multi mode, there are only 160 dots across the screen instead of 320. In other words, each dot in hires mode is one pixel wide; and each dot in multi mode is two pixels wide.

NOTE: When Graphics BASIC encounters an error in your program, the line containing the error is listed. The location of the error is indicated by a double angle bracket symbol < > (in reversed video). In making the corrections, be sure to remove the double angle brackets and press RETURN to enter the correction.

Changing Colors

You can change the border and background colors of the screen using the commands BORDER and BACKGROUND. These commands work if you include them within a program or if you type them directly without line numbers. If a demo program is still loaded into your Commodore memory, type NEW to clear the memory of that program. The program is stored on the Graphics BASIC disk permanently and you will not destroy the copy stored on the disk by typing NEW. Type in the following lines pressing the RETURN key after typing each line.

```
10 BORDER RED
15 BACKGROUND CYAN
```

Type **RUN,** press RETURN (or press f1), and watch the colors change.

In HIRES mode, you can work with two colors set with the HIRES COLOR <color> ON <color> command. If you do not specify which colors you want to use, the colors will be white on black. The COLOR HIRES command selects the color to be used with the next graphics command (DOT or LINE) performed.

In MULTI mode, you can work with four colors—the background color set with the BACKGROUND command, and three other colors set with the MULTI COLOR command. The COLOR MULTI command selects which of the three colors will be used to plot the next dot or line.

You can use the actual color names or codes in these commands. See Appendix E for a list of colors and their codes.

NOTE: Graphics BASIC allows you to combine several commands in one line as long as each command is separated by a : (colon). The above example could have been written as follows:

```
10 BORDER RED : BACKGROUND CYAN
```

NOTE ON EDITING PROGRAMS: When you want to change a value or mistake in a line, you can use the cursor movement keys (the two keys on the lower right of the keyboard) to position the cursor where you want the change. Type over the existing characters or use the INST/DEL key to insert or delete characters. Press the RETURN key to enter the corrections. (See the Commodore 64 User's Guide for more details.)

Plotting a Line

The LINE command draws a line from one point on the screen to another. The points are represented by the x and y coordinates of the screen. The point located on the lower left corner of the graphics screen is 0,0. To plot a line from the lower left corner to the upper right corner, specify the starting and ending locations of the line (the x and y coordinates). Type:

```
20 LINE 0,0 TO 320,200      <RETURN>
```

Type **HIRES,** and press RETURN to switch from the text to the high resolution screen. Type **RUN** (you will not see your typing on the screen) and press RETURN to watch the program in action. A diagonal line should appear on the screen. Notice that the background color turned to black. If you were to type MULTI the screen would appear with a CYAN background. Also notice how the line differs in HIRES and MULTI modes—the horizontal resolution makes each dot two pixels wide instead of one. Type **TEXT** (or press f7) to return to the text screen and type **LIST** (or press f3) to list the lines of your program.

Add a random element to the program by changing line 20, and create a loop by adding line 40. The RND(1) command is a BASIC function to generate a sequence of random numbers and perform calculations. GOTO is a BASIC command that sends the program to another line number; in this case the result is a repeating loop.

```
20 LINE TO 320*RND(1),200*RND(1)
40 GOTO 20
```

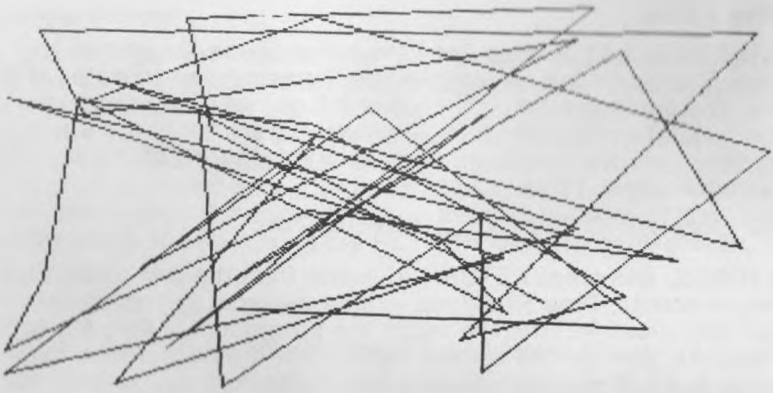
Type **CLEAR** and press RETURN to clear the screen of the diagonal line. Type **HIRES,** press RETURN, and type **RUN** and press RETURN (or press f1). A series of random lines should appear on the screen. Press the RUN/STOP key to stop the program, type **TEXT** and press RETURN. (or press f7).

Add a random color element to the program by adding line 30. This line randomly flips through four colors.

```
30 COLOR 1+4*RND(0)
```

Type **CLEAR** and press RETURN to clear the graphics screen. Then type **HIRES,** press RETURN, type **RUN,** and press RETURN. Random lines should appear on the screen in different colors. Run the same short program again, but type **MULTI** instead of HIRES before you type RUN.

Press the RUN/STOP key to stop the program, and type **RESET** before going on to the next section on drawing circles.



Drawing Circles

The CIRCLE command in Graphics BASIC allows you to draw circular shapes or arcs on the screen. With the CIRCLE command, you specify the x and y coordinates at the center of the circle. With the XYSIZE command, you specify the horizontal and vertical radii of the shape. By changing the values of the radii, you can draw circles or ellipses of varying sizes.

Type **NEW** to get rid of the short line program you just typed in. The following statements will draw a circle at the center of your screen.

```
10 CLEAR : HIRES
20 CIRCLE 160,100 XYSIZE 42,30
```

Type **RUN** and press RETURN. (You do not need to type HIRES because it was included as part of the program in line 10.)

NOTE: Because the screen is rectangular, the x and y radii will not be the same if you want to draw a perfect circle. To draw a perfect circle on a hires screen, the x radius should equal the y radius multiplied by 1.4. On a multicolor screen, the x radius should equal the y radius multiplied by 1.6. For example, in line 20 above the x radius of 42 was calculated by multiplying the y radius (30) by 1.4.

You can use the circle command to create an ellipse by changing the XYSIZE values. Type the following commands:

```
TEXT    <RETURN>
LIST    <RETURN>
```

NOTE: You could also press the f7 key for TEXT and the f3 key for LIST.

Add line 40 to draw an ellipse as follows:

```
40 CIRCLE 160,100 XYSIZE 80,15      <RETURN>
```

Type **RUN** and press RETURN. An ellipse should appear around the circle on the screen.

Experiment with different numbers for the x and y values and see what effects they have on the graphics when the program is run again. Be sure to put in the original variables before going on with the tutorial.

Filling in an Object

The FILL command in Graphics BASIC allows you to fill an enclosed area with color. The object is filled with the current color (set with the COLOR HIRES or COLOR MULTI command). You specify the point within the object where the color will start. To fill in the current shape on the screen in red, add line 30 to the listing:

```
30 COLOR HIRES RED : FILL 160,100      <RETURN>
```

Type **RUN** and press RETURN. Press f7 to switch to the text screen.

Plotting Dots

Place dots around your spheres by adding the following three lines before line 20.

```
13 FOR I=1 TO 320
15 COLOR WHITE : DOT I,320*RND(1)
17 NEXT I
```

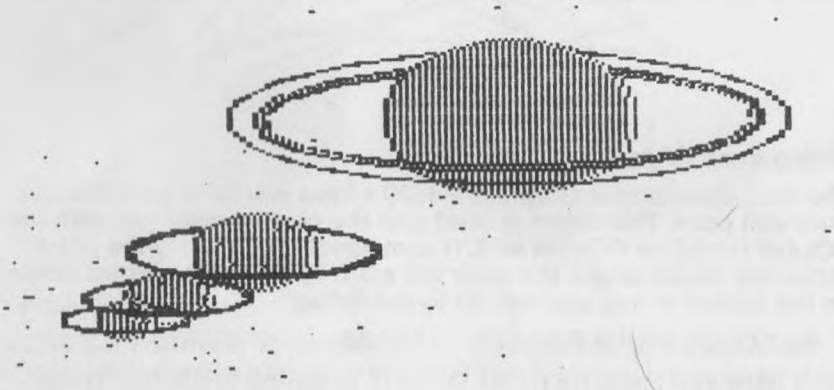
Type **RUN** and press RETURN. A series of dots should appear first, then your spheres.

Printing Text on a Graphics Screen

The GPRINT AT command allows you to put text on a graphics screen. You specify the location where the text is to start, and then type the text you want to appear. Locations for text commands differ from the locations for graphics. The location at the upper left corner of the screen is 0,0. Press f7 and add the following lines to the circle program:

```
50 GPRINT AT 1,1 "TO SEEK OUT"
60 GPRINT AT 1,3 "NEW LIFE AND"
70 GPRINT AT 1,5 "NEW CIVILIZATIONS"
```

Type **RUN**, press RETURN and the text should be added to your high resolution screen. You could change the size of the letters by adding the XYSIZE command as follows: GPRINT AT 1,1 XYSIZE 3,2 "TO SEEK OUT".



Printing Graphics and Program Listings

If you have a Gemini series or Commodore 1525 graphics printer, you can print out the graphics you create. The command COPY HIRES TO PRINTER prints out the image on the current hires screen. The command COPY MULTI TO PRINTER prints out the image on the current multi screen. (See the reference section for details.)

To print a program listing, use the following command sequence:

```
COPY TEXT TO PRINTER      <RETURN>
```

Saving a Program

WARNING: Do not save your programs on the Graphics BASIC disk. Use a separate formatted disk to store your programs. See Appendix G for instructions on formatting a disk.

Remove the Graphics BASIC disk from the drive, and insert a formatted disk. To save the circle program, type:

```
SAVE"CIRCLE PROGRAM",8
```

The number 8 specifies that you are using a disk drive with the device number 8 to store your programs. Once the program is saved, you can type NEW to clear the memory before going on to sprites.

Sprites

Creating and Moving Sprites

The Commodore 64 has the capability to produce high resolution, programmable graphics called sprites. Sprites are designs you can create in practically any shape.

In creating a sprite, you determine its characteristics by making the dots that form the sprite visible or invisible, that is, turning the dots on or off. You can display and animate up to eight sprites on the screen at one time.

With Graphics BASIC, you can move sprites automatically on both the text and graphic screens; animate sprites between several shapes automatically; set sprite positions, colors, shapes; and turn the sprites on or off. You can also check for sprite collisions as well as read any sprite's current position. Graphics BASIC also provides a built-in sprite editor which allows you to create your own HIRES and MULTICOLOR sprite shapes easily.

You can design up to thirty-two shapes at any one time and any sprite can select any one of these thirty-two available shapes for it to display. All eight sprites can select the exact same shape to display, choose different shapes, or animate (switch between) several shapes.

Using the Sprite Editor

Before you can put a sprite into a program, you need to create the sprite. You can create sprites in one of two modes: high resolution or multicolor. Hires sprites use only one color, and multicolor sprites can use four colors.

The sprite editor consists of a grid—24 dots wide by 21 dots tall for high resolution sprites. Each dot equals one pixel. You form a sprite by indicating which pixels are "turned off" and which are "turned on." The grid for multicolor sprites is 12 dots by 21 dots—the horizontal resolution is cut in half. Every two pixels represents one color, and you can use four different colors in one sprite.

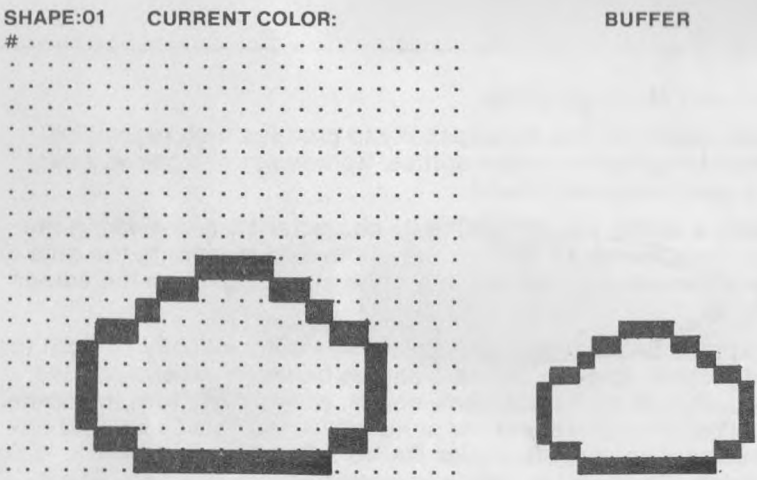
Several sprite shapes are included on the Graphics BASIC disk. Make sure the Graphics BASIC disk is in the drive. Load in the pre-defined shapes to be used in the tutorial that follows:

```
SPRITE LOAD"BBALL.SPR"        <RETURN>
```

To see the sprites in the sprite editor, type:

```
EDIT      and press RETURN.
```

The screen will clear and a grid will appear indicating that you are running the Graphics BASIC built-in sprite editor. The grid is an enlarged representation of the shape you are presently editing. The actual sprite is displayed on the lower right of the screen. The sprite edit area should look like this:



A number sign (#) indicates the current cursor position. The filled in spaces indicate the pixels that are turned on, thus giving the sprite its shape. The + key cycles forward through the sprite shapes, and the − key cycles back.

Because these shapes are pre-defined for the tutorial, you do not need to create new sprites or make any changes. However, the built-in sprite editor has many features you should learn about. Some of these features include a shape buffer to hold a sprite temporarily, scrolling, changing sprite size, and switching between multicolor and high resolution modes. See the EDIT command in the reference section on sprites for an explanation of these features

Putting Sprites into a Program

Leave the sprite editing session by typing the letter Q. The screen will clear and you will return to normal Graphics BASIC control. You now have sprites to put into a program. Type NEW and then type the following program to position, move, and animate your sprites. **Note:** do not type in the comments in parentheses.

```
10 RESET                          (this sets everything to normal)
20 SPRITE LOAD"BBALL.SPR"         (loads the sprites created
                                   for you)
30 BACKGROUND
     BLACK:BORDER BLACK
40 FOR I = 1 TO 8                 (loops through the sprites)
50 SPRITE I SHAPE 2 ON AT         (displays sprites 1 through 8
     10*I,50                       and positions them)
60 SPRITE I COLOR I               (cycles through the colors to
                                   give each sprite a color)
70 NEXT I                         (completes the loop)
```

Type **RUN,** press RETURN, and a series of eight balls should appear on the screen.

Sprite animation is one of the impressive features of Graphics BASIC. You can create sprites which change form and move across the screen. For example, a sprite could switch between the various shapes of a walking person while moving to the right. And amazingly, the animation and movement is totally transparent to BASIC—you can be editing a program while a man is walking across the screen.

Next, animate your sprites with the ANIMATE and SPEED commands. ANIMATE flips through the six sprite shapes. SPEED determines how fast the sprites will move—zero is the fastest and 127 is the slowest. You first set the animation sequence, set the speed, and then turn the animation on. Type the following lines replacing line 70.

```
70 SPRITE MOVE                    (directs sprites to move)
80 SPRITE I ANIMATE 1,2,3,4,5,    (flips through the sprite shapes in
     6,5,4,3,2  SPEED 8            assigned sequence; sets speed)
90 SPRITE I ANIMATE ON            (begins animation sequence)
100 NEXT I                        (completes the loop)
```

Before you type RUN, list line 20. Insert **REM** before the SPRITE LOAD command. The sprites have already been loaded into the sprite editor. Typing REM causes the program to treat the line as a remark, thus ignoring it.

Type **RUN,** press RETURN, and the balls should start to bounce. Add a slight delay before each sprite turns on. First, type **LIST** and press RETURN. Change line 100 and add line 110.

```
100 FOR T = 1 TO 60 : NEXT T
110 NEXT I
```

Type **RUN** and press RETURN.

Make the sprites move across the screen by defining the x and y speeds with the SPEED command. X and y values of 0,0 cause the sprite to stand still, and values of 5,5 cause it to move rapidly to the upper-right. Negative or positive values determine the direction the sprite moves. (Negative values move to the left; positive values move to the right.) Replace line 110 and add line 120.

```
110 SPRITE I SPEED -2,0
120 NEXT I
```

Type **RUN** and press RETURN. And see what amazing feats you can do with just a few lines.

LITTLE BITTY BOUNCING BALLS

Things to Play Around With

Make the change, enter it, and type RUN.

- change the SPEED value in line 80 (0 is the fastest; 127 is the slowest)
- change the size of sprites by adding one of the following commands to line 60 between the SPRITE I and COLOR I commands

  ```
  XYSIZE 2,2
  XYSIZE 2,1
  XYSIZE 1,2
  XYSIZE 1,1      (returns to normal size)
  ```

- change the movement direction in line 110—make the value after the SPEED command positive instead of negative
- change the amount of space between the sprites by removing the **10*** in line 50
- type SPRITE FREEZE directly (not with a line number); type SPRITE MOVE to start the sprites again
- change the x and y coordinates in line 50 to **150,I*25+10** or **150+I*2,I*10+10** (replacing the values **10*1,50**)
- change colors in line 60 to **15*RND(8)**

Experiment with other variables and commands to discover how easy it is to program with Graphics BASIC.

Saving the Sprite Program

To save the sprite program, first remove the Graphics BASIC disk from the disk drive. Insert a formatted disk to store your programs. You could remove REM from line 20 before saving the program. Type:

```
SAVE"SPRITUTOR",8      <RETURN>
```

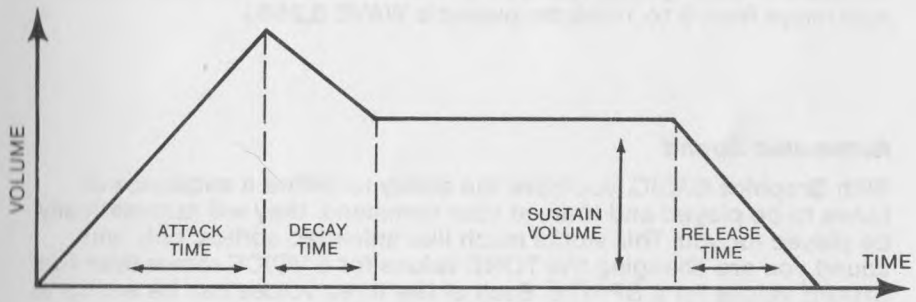
Making Sounds

The Commodore 64 has a complex sound generator which can be used to create sounds ranging from a laser zap to a three piece orchestra. With such a wide selection of sounds possible, the programming necessary would normally be extensive. With Graphics BASIC sound commands, the task is immensely simplified.

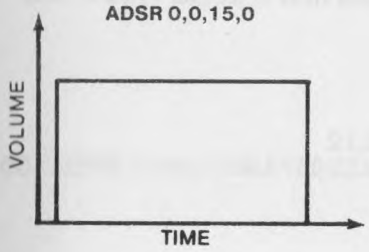
The Commodore 64 has three 'voices' which can be independently controlled. For example, while voice number one plays a high pitched note sounding like a harpsichord, voice two can play a low note like a violin and voice three can make a drum sound.

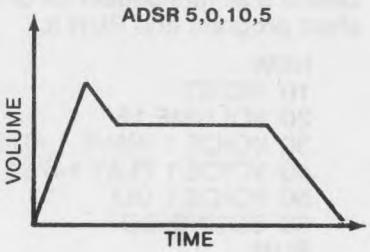
Type NEW before proceeding to the next section on sound. Press f4 and RETURN if you prefer programming without bouncing sprites on the screen.

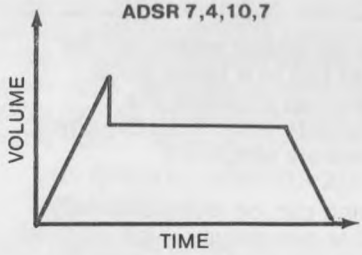
The type of sound that each voice makes is determined by its WAVE (triangle, sawtooth, pulse, and noise) and ADSR (attack, decay, sustain, release). The WAVE number selects the waveform for the voice and the ADSR number selects its attack time, decay time, sustain volume, and release time (also called the volume envelope). These four variables determine the form of the volume envelope for each note played. For example, a short attack time will make notes start suddenly, while a long attack time will produce notes which slowly rise up to full volume. Graphically, the ADSR volume envelope looks like this:

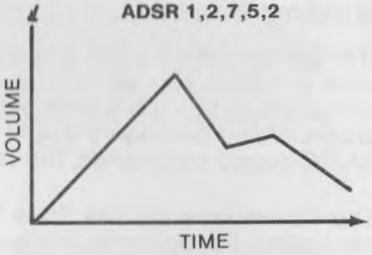


Following are some examples of various ADSR settings.









RUN the following program several times, substituting different values for the **ADSR** and note the difference in the sound for each setting. (Allowable **ADSR** values range from 0 to 15.)

```
NEW
10 SOUND CLEAR : VOLUME 15
20 VOICE 1 ADSR 0,0,15,0 WAVE 1 TONE 4000
30 VOICE 1 ON : REM START ATTACK
40 FOR I=1 TO 1000 : NEXT I : REM HOLD NOTE
50 VOICE 1 OFF : REM START RELEASE
RUN
```

Now change the values following **TONE** and **WAVE** to see what effect these have on the sound produced. (The **TONE** value may range from 0 to 65535 and **WAVE** values of 1 for triangle; 2 for sawtooth, and 4 for noise. Wave 3 (pulse) requires an additional pulse variable which may range from 0 to 1024, for example WAVE 3,258.)

Automated Sound

With Graphics BASIC, you have the ability to define a sequence of tones to be played and then, at your command, they will automatically be played for you. This works much like animated sprites, only with sound you are changing the TONE values for a VOICE rather than the SHAPE values for a SPRITE. Each of the three voices can be set up to play its own sequence of tones and this all happens independent of your BASIC program. In other words, you can be listening to a three part harmony while editing your program.

Define a simple sequence of notes to see how it works. Type in this short program and RUN it.

```
NEW
10 RESET
20 VOLUME 15
30 VOICE 1 WAVE 1 ADSR 15,9,12,12
40 VOICE 1 PLAY 14764,16572,13253,6573,9854,9854 SPEED 30
50 VOICE 1 GO
60 SOUND GO
RUN
```

If the volume on your TV or monitor is turned up, you will hear the four notes being played.

Now insert a comma and a back-arrow (the key in the upper-leftmost corner of your keyboard) in line 40 so that it reads as follows:

```
40 VOICE 1 PLAY 14764,16572,13253,6573,9854,9854,←
   SPEED 30
```

When you RUN the program again, the same sequence of notes will play again, only now it will start over after it is finished. To stop the notes, type SOUND OFF. Change the SPEED in line 40 from 30 to 1 and listen to the effects. SPEED values range from 0 to 255.

Learning More About Graphics BASIC

This tutorial has given you just a hint of the many features of Graphics BASIC. You can build upon what you have learned and go on to create complex programs with exciting graphics and sound. The Reference Section which follows describes all of the Graphics BASIC commands you will need.

REFERENCE SECTION

The command directory on the following pages explains each Graphics BASIC command in detail. The commands are grouped by function: graphics, sprites, sound, input/output commands, programming aids, and text commands.

Each command is described and illustrated by one or two examples to give you a sample of the use of the command. Some of the examples are taken out of context of a program and are not meant to be run. In these cases, the comment "for illustration only" precedes the example.

Demo programs for some of the commands are included on your Graphics BASIC disk. If one of these programs is listed, load and run it as follows:

```
LOAD"<name>",8
RUN
```

To stop and list the program, press the RUN/STOP and RESTORE keys. You will be returned to the Graphics BASIC text screen. Type LIST to print the program listing. Make changes in some of the values and run the program again to see the results.

The following conventions are used in Graphics BASIC commands:

1. Graphics BASIC commands are displayed in capital letters and must be entered precisely as shown.
2. Items in angle brackets (<. . .>) are required variables which you provide.
3. Variable separators (such as a comma, semi-colon, or colon) must be included in the statement.
4. Items inside squared brackets ([. . .]) are optional variables or limits that can be part of the command statement.
5. Optional statement parameters followed by three dots (. . .) can be repeated as needed.
6. Items separated by a slash (/) indicate a choice must be made.

COMMAND SUMMARY

All of the Graphics BASIC commands and syntax are grouped by category and listed on the following pages. Items in brackets [. . .] are optional; and items in angle brackets <. . .> are variables. The numbers and symbols preceding some commands indicate the following:

KEY DESCRIPTION

1 Interrupt-driven
2 Coordinates depend on ORIGIN
3 Coordinates depend on SCALE
4 Window clipped
5 Depends on RAM or ROM character set currently selected
6 Resets BASIC workspace and erases variables
7 Requires Commodore Graphics or Gemini series Printer
8 Disables all interrupts (temporarily) RESET leaves them off

Utilities and Enhancements

Joysticks

<var>=JOY(<n>)

Programming Aids

FIND "<string>"
CHANGE "<string>" TO "<string>"
REN [<n>] [,<n>]
ON ERROR GOTO <n>
ON ERROR ON
ON ERROR OFF
PROCEDURE <name> [(<var>,<var>, . . ., <var>)]
DO <name> [<var>, <var>, . . . , <var>)]
IF <condition> THEN <command> : ELSE <command>
ELSE <command>
GOTO <exp> (also GOSUB, RESTORE, LIST, etc.)

Miscellaneous

KEY ON
KEY OFF
KEY (<n>)=string
KEY LIST
KEY SAVE "<filename>" [,<device>]
KEY LOAD "<filename>" [,<device>]

```
   8  RESET
      BACKGROUND <color>
      BORDER <color>
   1  TEXT [FROM <n>] [TO <n>]
   1  HIRES [FROM <n>] [TO <n>]
   1  MULTI [FROM <n>] [TO <n>]
      CLEAR [<n>]
      HIRES COLOR <color> ON <color>
      MULTI COLOR <color>, <color>, <color>
      COLOR [HIRES] [MULTI] <color>
2 3 4 DOT <x>, <y> [;<x>, <y>] . . . etc
2 3 4 <var> = DOT(<x>,<y>)
2 3 4 LINE [<x>,<y>] [TO <x>,<y>] . . . etc
      SCALE [<x>,<y>]
   3  SETORIGIN [<x>,<y>]
  2 3 WINDOW [<x>,<y>,<x>,<y>]
2 3 4 BOX <x>,<y> TO <x>,<y>
2 3 4 BOX <x>,<y> XYSIZE <x>,<y>
2 3 4 CIRCLE <x>,<y> XYSIZE <x>,<y> [FROM <n>] [TO <n>]
        [STEP <n>]
   5  GPRINT [AT <x>,<y>] [XYSIZE <x>,<y>] <output data>
2 3 4 FILL <x>,<y>
  4 7 COPY HIRES TO PRINTER
   4  COPY HIRES TO SPRITE <n>
   4  COPY SPRITE <n> TO HIRES
   5  COPY TEXT TO HIRES
   8  HIRES SAVE "<filename>" [,<device>]   }
   8  HIRES LOAD "<filename>" [,<device>]   }   default device
   8  MULTI SAVE "<filename>" [,<device>]   }      number
   8  MULTI LOAD "<filename>" [,<device>]   }   is 8 (disk)
      COPY MULTI TO SPRITE <n>
      COPY SPRITE <n> TO MULTI
```

KEY	DESCRIPTION
1	Interrupt-driven
2	Coordinates depend on ORIGIN
3	Coordinates depend on SCALE
4	Window clipped
5	Depends on RAM or ROM character set currently selected
6	Resets BASIC workspace and erases variables
7	Requires Commodore Graphics Printer
8	Disables all interrupts (temporarily) RESET leaves them off

Sprite Commands

```
   8  RESET
   8  EDIT
      SPRITE MULTICOLOR <color>, <color>
      SPRITE <n>  ON
                  OFF
 2 3              AT <x>,<y>
                  COLOR <color>
                  MULTI
                  HIRES
                  XYSIZE <x>,<y>
                  ON BACKGROUND
                  UNDER BACKGROUND
                  ANIMATE ON
                  ANIMATE OFF
   1              SPEED <x>,<y>
   1  SPRITE <n> ANIMATE <n>,<n>,...,<n>  SPEED <n>
      SPRITE MOVE
      SPRITE FREEZE
 2 3  <var> = XPOS (<n>)
 2 3  <var> = YPOS (<n>)
      <var> = SPRITE (<n>)
      <var> = BACKGROUND (<n>)
      SPRITE <n> CLEAR HIT
      SPRITE CLEAR HIT
      COPY SPRITE <n> TO HIRES
      COPY HIRES TO SPRITE <n>
      SCALE [<x>,<y>]
   3  SETORIGIN [<x>,<y>]
   8  SPRITE SAVE <n>,<n> "<filename>" [,<device>]   } default
   8  SPRITE LOAD "<filename>" [,<device>]           } device
      COPY MULTI TO SPRITE <n>                       } number
                                                     } is 8 (disk)
```

KEY	DESCRIPTION
1	Interrupt-driven
2	Coordinates depend on ORIGIN
3	Coordinates depend on SCALE
4	Window clipped
5	Depends on RAM or ROM character set currently selected
6	Resets BASIC workspace and erases variables
7	Requires Commodore Graphics Printer
8	Disables all interrupts (temporarily) RESET leaves them off

Sound Commands

8 RESET
SOUND CLEAR
VOLUME <n>
SOUND ON
SOUND OFF
SOUND GO
SOUND FREEZE
VOICE <n> TONE <n>
ADSR <n>,<n>,<n>,<n>
WAVE <n> [,<n>] (or WAVE <wavename>)
[,<n>]
1 VOICE <n> PLAY [CONT] <n> [;<n>;<n>],<n>
[;<n>;<n>], . . .SPEED <n>

Copy Commands

COPY TEXT TO HIRES
COPY TEXT TO PRINTER
7 COPY HIRES TO PRINTER
COPY SPRITE <sprite number> TO HIRES/MULTI
7 COPY MULTI TO PRINTER
COPY TEXT TO MULTI (not recommended)
COPY HIRES TO SPRITE <n>
COPY MULTI TO SPRITE <n>

KEY	DESCRIPTION
1	Interrupt-driven
2	Coordinates depend on ORIGIN
3	Coordinates depend on SCALE
4	Window clipped
5	Depends on RAM or ROM character set currently selected
6	Resets BASIC workspace and erases variables
7	Requires Commodore Graphics Printer
8	Disables all interrupts (temporarily) RESET leaves them off

8 RESET
BACKGROUND <color>
BORDER <color>
1 TEXT [FROM <n>] [TO <n>]
1 HIRES [FROM <n>] [TO <n>]
1 MULTI [FROM <n>] [TO <n>]
5 PRINT AT <x>,<y> <output data>
SCROLL <direction> <n> [WINDOW <x>,<y>,<x>,<y>]
ROLL <direction> <n> [WINDOW <x>,<y>,<x>,<y>]
5 COPY TEXT TO HIRES
COPY TEXT TO PRINTER
8 TEXT SAVE "<filename>"[,<device>]
8 TEXT LOAD "<filename>" [,<device>]
CHAR ROM
CHAR RAM
CHAR (<ascii>) = <n>,<n>,<n>,<n>,<n>,<n>,<n>,<n>
CHAR (<ascii>,n) = "<an 8-character string>"
8 CHAR SAVE "<filename>" [,<device>]
8 CHAR LOAD "<filename>" [,<device>]
6 COPY UPPERCASE TO RAM
6 COPY LOWERCASE TO RAM
6 CHAR SET MEMORY
6 CHAR RESET MEMORY

KEY	DESCRIPTION
1	Interrupt-driven
2	Coordinates depend on ORIGIN
3	Coordinates depend on SCALE
4	Window clipped
5	Depends on RAM or ROM character set currently selected
6	Resets BASIC workspace and erases variables
7	Requires Commodore Graphics Printer
8	Disables all interrupts (temporarily) RESET leaves them off

I/O (Input/Output) Commands

8	DIR [,<device>]	default device number is 8 (disk)
	DISK [,<device>]	
	DISK "<command string>" [,<device>]	
8	TEXT SAVE "<filename>" [,<device>]	
8	TEXT LOAD "<filename>" [,<device>]	
8	HIRES SAVE "<filename>" [,<device>]	
8	HIRES LOAD "<filename>" [,<device>]	
8	MULTI SAVE "<filename>" [,<device>]	
8	MULTI LOAD "<filename>" [,<device>]	
8	SPRITE SAVE <n>,<n> "<filename>" [,<device>]	
8	SPRITE LOAD "<filename>" [,<device>]	
8	CHAR SAVE "<filename>" [,<device>]	
8	CHAR LOAD "<filename>" [,<device>]	
8	KEY SAVE "<filename>" [,<device>]	
8	KEY LOAD "<filename>" [,<device>]	
	COPY TEXT TO PRINTER	
4 7	COPY HIRES/MULTI TO PRINTER	
	<var> = JOY (<n>)	

KEY	DESCRIPTION
1	Interrupt-driven
2	Coordinates depend on ORIGIN
3	Coordinates depend on SCALE
4	Window clipped
5	Depends on RAM or ROM character set currently selected
6	Resets BASIC workspace and erases variables
7	Requires Commodore Graphics Printer
8	Disables all interrupts (temporarily) RESET leaves them off

BACKGROUND <background color>

Description: Changes the background of the text screen to <background color> you specify. You can use either the color numbers or the actual screen color names themselves. See Appendix E for a list of the colors available. This command also selects the background color to be used in multicolor mode. The f2 key has been pre-defined to print out the word: BACKGROUND. You can use this command in either direct or program mode.

Example: BACKGROUND RED

```
10 BACKGROUND BLACK
RUN
```

```
10 BACKGROUND 1+2*7
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. The color number given is out of range.

Command Format: BACKGROUND <color>

BORDER <border color>

Description: Changes the color of the border to <border color>. Either the color numbers or the actual color names may be used. You can use this command in either direct or program modes.

Example: BORDER PURPLE

```
10 BORDER RED
RUN
```

```
10 BORDER 8/2+1
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. The color number given is out of range.

Command Format: BORDER <color>

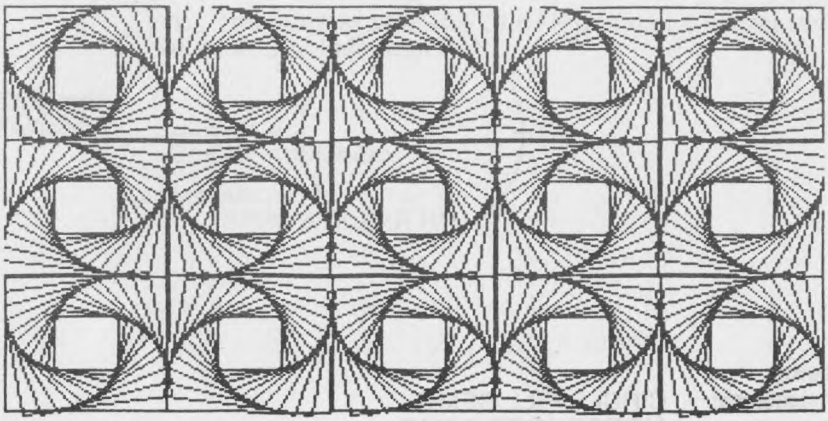
BOX <corner x>,<corner y> [**XYSIZE** <x>,<y>]
[TO <corner x>,<corner y>]

Description: Draws rectangular boxes on the HIRES and MULTI screens. The coordinates <corner x> and <corner y> specify the location on the screen where a corner of the box will be drawn. The XYSIZE values determine the size and shape of the box. If the values of <x> and <y> are positive, the <corner x> , <corner y> point will be the lower left corner of the box. By using negative values for the XYSIZE, the box will be drawn down and to the left of the point <corner x>,<corner>. When using the TO option, you can specify the upper right corner of the box rather than the size of the sides.

Examples:

```
10 HIRES
20 BOX 10,10 XYSIZE 30,40
RUN
CLEAR
RESET
```

```
10 HIRES
20 FOR A=0 TO 360 STEP 10
30 BOX 100,100 XYSIZE 30,30
40 NEXT A
RUN
CLEAR
RESET
```



CIRCLE <center x>, <center y> XYSIZE <x size>,<y size> [FROM <starting angle>] [TO <ending angle>] [STEP <angle>]

Description: Draws circles, ellipses, arcs, and regular polygons. The CIRCLE variables <center x> and <center y> specify the location on the screen of the center of the proposed circle. The XYSIZE variables determine the size and shape of the circle. By changing the <x size> and <y size>, larger circles and arcs can be made, as well as ellipses and elliptical arcs.

Omitting the optional FROM . . . TO command draws a full circle. Including the FROM TO command creates an arc. By making the <angle> large in the optional STEP sub-command, polygons can be drawn. All angles are in degrees and may take on any values.

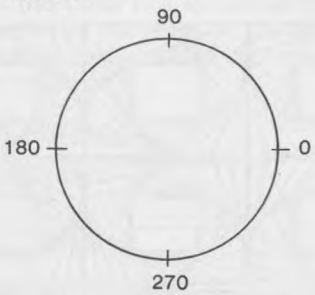
Examples:

```
10 HIRES
20 CIRCLE 50,60 XYSIZE 10,30
RUN
RESET
```

```
20 CIRCLE 100,100 XYSIZE 28,20 STEP 360/5
RUN
RESET
```

```
20 CIRCLE 100,120 XYSIZE 50,40 FROM 10 TO 180 :
     REM DRAWS ARC
RUN
```

The angle values used in the FROM . . . TO command are as follows:



Demo Programs: CIRCLE FILL DEMO
MODERN ART DEMO

CLEAR [<BYTE>]

Description: Fills the high-resolution (and multicolor) screen memory with <byte>. CLEAR can be used either before or after the HIRES COLOR n ON n or MULTICOLOR n1,n2,n3 commands. The screen will become the designated HIRES color, however, only after both commands have been used.

CLEAR or CLEAR 0 turns the screen to the background color. CLEAR 255 turns the screen to the foreground color if used in HIRES mode. CLEAR 255 also turns the foreground color to <color3> of the MULTICOLOR command if the screen is in MULTICOLOR mode. Also, CLEAR 85 will turn the MULTICOLOR screen to <color1> and CLEAR 170 will turn it to <color2>.

Example:

```
10 MULTI : BACKGROUND PURPLE
20 MULTICOLOR RED,GREEN,BLUE
30 CLEAR
RUN
```

Effect: The HIRES screen is now purple.

Example:

```
RESET
10 HIRES
20 HIRES COLOR RED ON YELLOW
30 CLEAR 255
RUN
```

Effect: The HIRES screen is now red.

Errors: ILLEGAL QUANTITY ERROR

1. The number after the CLEAR command is greater than 255 or less than 0.

Command Format:

CLEAR
CLEAR <number>

see also
SPRITE CLEAR HIT

COLOR [HIRES] <color>

COLOR [MULTI] <color>

Description: Selects the color to be used with the DOT and LINE commands. Either the color number or the actual color name may be used. When drawing in HIRES mode, the COLOR HIRES <color> command should be used. When drawing in MULTI mode, the COLOR MULTI <color> command should be used. However, you can omit the HIRES or MULTI following the word COLOR. If omitted, Graphics BASIC will assume you are going to draw in the mode in which the computer is presently operating. For example, if you are in HIRES mode, COLOR <color> sets the color for drawing on the HIRES screen.

Example:

```
RESET
10 HIRES
20 HIRES COLOR BLUE ON BLACK
30 FOR D = 1 TO 300 : NEXT D
40 COLOR HIRES YELLOW
50 BOX 100,100 XYSIZE 30,40
Press the f1 key several times to see the effects.
```

Effects: The HIRES screen will be black and the box plotted will switch from yellow to blue.

Example:

```
RESET
NEW
10 MULTI : BACKGROUND 0 : CLEAR
20 MULTICOLOR RED,BLUE,YELLOW
30 COLOR BLUE
40 LINE TO 100,120
RUN
```

Effects: The multicolor screen will be black and the next line plotted will be in blue.

NOTE: If a <color> is used which is not one of the principal colors selected in the HIRES color n1 ON n2 or MULTI COLOR n1,n2,n3 and BACKGROUND commands, an error will not occur. Instead, the next line or dot plotted will be drawn in the selected color. Caution must be taken when doing this, however, because when two such lines cross, their colors will "bleed" onto one another. This will not occur if your figures do not come into contact with each other, and

bleeding will *never* occur if only the principal colors are selected. This is a limitation of the Commodore 64 itself and cannot be avoided.

Errors: ILLEGAL QUANTITY ERROR

1. <color> is not in the range of 0 to 15.

Command Format:
COLOR <color>
COLOR HIRES <color>
COLOR MULTI <color>

Demo Program:
MULTICHANGE DEMO
COLORWHEEL DEMO

COPY HIRES TO SPRITE <sprite number>

COPY MULTI TO SPRITE <sprite number>

Description: Transfers—or "picks up"—the image underneath the specified sprite and puts it into the sprite shape data. This command is useful for "picking up" objects on the hires or multi screen and moving them around. When used in conjunction with the COPYSPRITE <sprite number> TO HIRES command, images can be easily transferred from one part of the graphics screen to another.

Example:

```
NEW
10 BACKGROUND 1 : CLEAR
20 MULTI : MULTICOLOR PURPLE, GREEN, BLUE
30 FOR T = 1 TO 25 : COLOR 3*RND(8) + 4
40 LINE TO 320*RND(8), 150*RND(8) : NEXT T
50 SPRITE 1 ON AT 150,100 MULTI
60 COPY MULTI TO SPRITE 1
70 FOR T = 200 TO 100 STEP – 20
80 SPRITE 1 AT 150,T
90 COPY SPRITE 1 TO MULTI : NEXT T
RUN
```

The short program above creates a sprite and copies a multicolor image onto the sprite. That sprite is then copied back onto the screen with the COPY SPRITE 1 TO MULTI command. Keep pressing the f1 key (located on the right side of the keyboard) to RUN the program again.

With the following commands, you can pick up letters (from a character set) and put them in a sprite. Type the following commands to load a font, and copy them to a sprite. Make sure the Graphics BASIC disk is in the drive.

```
NEW
RESET
CHAR LOAD "COMPUTER.FONT"
CHAR RAM
HIRES TO 15
CLEAR
GPRINT AT 1,1 XYSIZE 3,2 "HESWARE"
SPRITE 1 ON AT 25,200
COPY HIRES TO SPRITE 1
SPRITE 1 SPEED 1,0 : SPRITE MOVE
```

While the sprite is visible on the screen, type:

```
COPY SPRITE 1 TO HIRES      <RETURN>
```

DOT <x>,<y> [;<x>,<y>] [;<x>,<y>] . . .

Description: Plots a dot [or series of dots] at location <x>,<y>. The dot will appear in the color defined by the COLOR command. The <x>,<y> coordinates will always depend on the current setting of the origin and the present SCALE. Note also that dots will not be plotted if they lie outside of the specified WINDOW.

If multicolor dots are being plotted, only half the horizontal resolution of HIRES is possible. The coordinate systems of the two modes, however, are identical. The only difference lies in the fact that the multicolor dots will be twice as wide as high resolution dots. Thus, the 160th multicolor dot will have an x-coordinate of 320 just as if it were a HIRES dot.

Example:

```
RESET
NEW
10 MULTI : BACKGROUND RED
20 MULTICOLOR WHITE,PURPLE,PEACH
30 CLEAR : COLOR MULTI WHITE
40 DOT 10,8
RUN
```

Effects: The multicolor screen is now red and a white dot should appear at (10,8)—the lower left corner of the screen.

Example:

```
RESET
NEW
10 HIRES
20 HIRES COLOR WHITE ON BLACK
30 CLEAR
40 FOR I=1 TO 32 STEP .10 : x(1)=I*10 :
    Y(1)=ABS(100*SIN (I))
50 COLOR WHITE : DOT X(1), Y(1)
60 NEXT I
RUN
```

Effect: Plots a partial white sine wave. Stop the program by pressing the RUN/STOP key and then press f7.

Add the following two lines and run the program again:

```
55 COLOR BLACK : DOT X(6), Y(6)
57 FOR J = 5 TO 1 STEP −1 : X(J+1)=X(J):
   Y(J+1) = Y(J) : NEXT J
```

Command Format:

DOT <x>,<y>
DOT <x>,<y>;<x>,<y>;<x>,<y> . . .

Demo Programs: SATURN DEMO
SHUTTLE DEMO

FILL <x>,<y>

Description: Fills an *enclosed* object with the current color on the HIRES screen. The specified point (x,y) must be within the object, otherwise the entire area outside the object will be filled.

Example:

```
RESET
NEW
10 HIRES : COLOR PURPLE
20 BOX 100,100 XYSIZE 50,50
30 FILL 130,120
RUN
```

Demo Program:

CIRCLE FILL DEMO
PIE CHART DEMO
MODERN ART DEMO

GPRINT [AT <cursorx>,<cursory>] [XYSIZE <xfactor>,<yfactor>] [,] [<output data>] [;/,]

Description: Prints letters and numbers on the HIRES and MULTI screens. This command works just like the standard BASIC PRINT command with a few exceptions. GPRINT recognizes only alphanumeric and graphic characters, color codes, reverse on/off codes, and return. Instead of responding to the conventional cursor movements, GPRINT allows x,y positioning of the 'hires' cursor. GPRINT also has the ability to expand the size of printed characters in both the horizontal and vertical directions.

Example:

```
RESET
NEW
10 HIRES TO 20 : CLEAR
20 FOR I=1 TO 13
30 GPRINT AT I,I "GRAPHICS BASIC"
40 NEXT I
RUN
```

GPRINT (Continued)

Example:

```
RESET
NEW

10 CHAR RAM : CHAR LOAD "OLD.FONT"
20 REM <— TEN COLOR CODES*:C$="[PUR][BLU]
     [SKY][LGN][GRE][YEL][PIN][RED][ORG][BRN]"
30 HIRES : CLEAR
40 FOR I=1 TO 10
50 GPRINT AT 1,2*I+1 XYSIZE 2,2 LEFT$(C$,I); "GOTHIC
     COLORS BATMAN"
60 NEXT I
RUN
```

*To get the ten color codes, press the keys listed below (press the CTRL key or the Commodore logo key (C<) and the number at the same time):

purple	CTRL 5
blue	CTRL 7
sky	C< 7
lt. green	C< 6
green	CTRL 6
yellow	CTRL 8
pink	C< 3
red	CTRL 3
orange	C< 1
brown	C< 2

NOTE: The GPRINT will not 'wrap' characters around to the start of the next line if you try to print beyond the right side of the screen. However, if you try to print off the bottom of the screen, the cursor will wrap up to the first line and characters will appear starting at the top of the screen.

Errors: ILLEGAL QUANTITY ERROR

1. The x,y position given for the cursor does not lie on the screen.
2. The x,y expansion factors given are negative or too large.

Command Format:

GPRINT
GPRINT <output data>
GPRINT AT <x>,<y>,<output data>
GPRINT AT <x>,<y> XYSIZE <x>,<y>, <output data>

Demo Program: GPRINT DEMO

HIRES

Description: Causes the screen on your Commodore 64 to display high resolution graphics instead of the normal TEXT screen. You can switch from a hires screen to a multicolor screen by typing MULTI. Switch to a text screen by typing TEXT.

Example:

```
RESET
NEW
10 HIRES
RUN
```

Effects: You will now see the HIRES screen. To see what you are typing, simply type TEXT and hit RETURN.

See also
MULTI
TEXT
HIRES TO/FROM
TEXT TO/FROM

HIRES [FROM <firstline>] [TO <lastline>]

Description: Splits the screen into two sections: one for text and the other for high resolution graphics. High resolution graphics are visible from <firstline> to <lastline>. A text screen is displayed on the remaining portion of the screen.

Examples: (for illustration only)

```
10 HIRES FROM 1 TO 10

10 HIRES FROM 10

10 HIRES TO 2

10 HIRES FROM 1 TO 2*9

10 A=1
20 B=9
30 HIRES FROM A TO B+8
```

Errors: ILLEGAL QUANTITY ERROR

1. Either of the numbers given in the HIRES command are not in the range of 1 to 25.

HIRES (Continued)

Command Format:
HIRES
HIRES TO <line>
HIRES FROM <line>
HIRES FROM <line> TO <line>

Demo Programs: SPLIT DEMO

HIRES COLOR <foregroundcolor> ON <backgroundcolor>

Description: Defines the principal foreground and background colors for use on the HIRES screen.

Colors may be defined using any of the 16 Commodore color numbers (0-15) or by using the 16 color names that Graphics BASIC recognizes (see page 109).

NOTE: Normally, only two colors are used in HIRES mode. Other colors can be used, but every character-block of dots on the HIRES screen must always share the same two colors: either the background color or the foreground color. If colors other than these two principal colors are used, lines which cross each other may "bleed" colors onto one another. This is a limitation of the Commodore and cannot be avoided. No problems of this nature will occur if you limit yourself to the two principal colors or prevent drawings of different colors from coming into contact.

Examples:

```
RESET
NEW
10 HIRES COLOR WHITE ON BLACK
20 COLOR WHITE : LINE 0,0 TO 100,200
HIRES
RUN
```

Effect: Draws a white line on screen with a black background.

Examples: (for illustration only)

```
10 HIRES COLOR 1 ON 0
```

```
10 B = 4
20 HIRES COLOR B+1 ON 6*2
```

Errors: ILLEGAL QUANTITY ERROR

1. A color number is not in the range of 0 to 15.

Command Format: HIRES COLOR <color> ON <color>

LINE [<x>,<y>] [TO <x>,<y>] [TO <x>,<y>] . . .

Description: Draws lines from any point on or off the screen to any other point. The LINE command will always draw lines in the color selected by the COLOR command. The LINE command can also be used with the DOT command where the DOT command determines the starting point for the LINE command.

The <x>,<y> coordinates used in the LINE command are referenced from the origin as set by the SETORIGIN command, and also affected by the present SCALE.

Following are the various forms of the LINE command.

LINE x1,y1 TO x2,y2
Draws a line from (x1,y1) to (x2,y2)

LINE TO x,y
Draws a line from the last point plotted to (x,y)

LINE x1,y1 TO x2,y2 TO x3,y3
Draws a string of lines from (x1,y1) to (x2,y2) to (x3,y3).

LINE x,y
Sets the starting point of the next LINE TO command to (x,y). Nothing is plotted.

Example:

```
RESET
NEW
10 HIRES
20 HIRES COLOR WHITE ON BLACK
30 COLOR WHITE
40 CLEAR
50 FOR I=1 TO 690 STEP 5
60 LINE 0,0 TO I,200
70 NEXT I
RUN
```

```
RESET
10 HIRES
20 HIRES COLOR SKY ON BLUE
30 COLOR SKY
40 CLEAR
50 X1=5 : X2=315
60 Y1=5 : Y2=195
70 LINE X1,Y1 TO X2,Y1 TO X2,Y2 TO X1,Y2 TO X1,Y1
RUN
```

Effect: Draws a frame around the screen.

LINE (Continued)

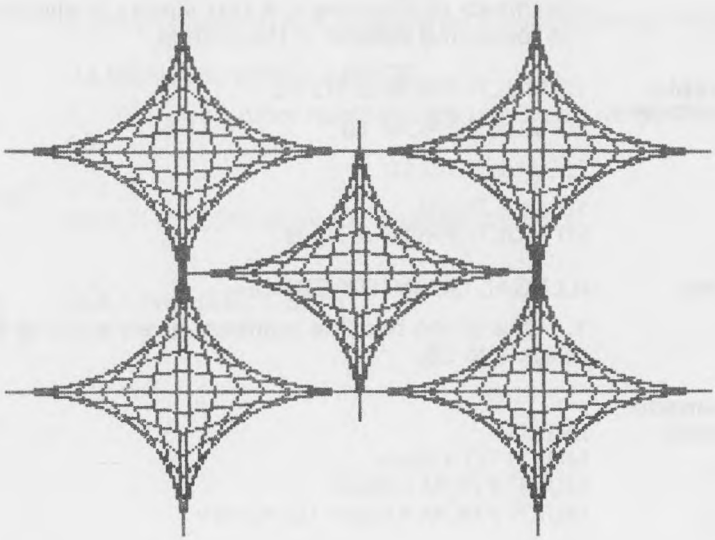
```
RESET
NEW
10 HIRES : CLEAR
20 HIRES COLOR WHITE ON BLUE : COLOR WHITE
30 FOR I = 1 TO 4 : X(I)=150*RND(1)+50 :
     Y(I)=80*RND(I)+20 : NEXT I : Y(4)=Y(4)+100
40 DOT X(1), Y(1)
50 FOR I =2 TO 3 : LINE TO X(I),Y(I) : NEXT I
60 LINE TO X(1),Y(1)
70 FOR I = 1 TO 3 : DOT X(4),Y(4) : LINE TO X(I),Y(I) :
     NEXT I
RUN
Press the f1 key to see another random pyramid.
```

Command Format:

LINE <x>,<y>
LINE<x>,<y> TO <x>,<y>
LINE TO <x,y>
LINE <x>,<y> TO <x>,<y> TO <x>,<y> . . .
LINE TO <x>,<y> TO <x>,<y> . . .

Demo Program:

DESIGN DEMO (press space bar see each design)
MULTISTAR DEMO
DIAMOND DEMO
SHIELD DEMO



MULTI

Description: The MULTI command will cause the entire screen to display multicolor graphics instead of the normal text screen. You can switch from the multicolor screen to the hires screen by typing HIRES, and you can get back to the text screen by typing TEXT.

Example:

```
10 MULTI
RUN
```

Effects: The screen will display multicolor graphics. To return to text mode simply type TEXT.

See also
HIRES
TEXT

MULTI [FROM <firstline>] [TO <lastline>]

Description: Splits the screen with multicolor graphics visible from <firstline> to <lastline>. A text screen is displayed on the remaining portion of the screen.

Example: (for illustration only)

```
10 MULTI FROM 5 TO 15

10 MULTI FROM 10

10 MULTI TO 12

10 A=5 B=20
20 MULTI FROM A TO B
```

Errors: ILLEGAL QUANTITY ERROR

1. One of the two line numbers given is out of the range 1 to 25.

Command Format:

```
MULTI
MULTI TO <line>
MULTI FROM <line>
MULTI FROM <line> TO <line>
```

MULTI COLOR <color1>,<color2>,<color3>

Description: Selects the three principal colors that will be used for plotting on the multicolor screen. The background color for the multicolor screen is the same as the background color for the text screen. Use the BACKGROUND command to select the multicolor background color. You can indicate colors using the 16 color names the Graphics BASIC recognizes. (A list of the color names can be found on page 109.)

WARNING: Do not use the PRINT"[CLR/HOME]" statement **after** using the MULTICOLOR command.

NOTE: Normally, only four colors are used in MULTI mode: the three principal colors defined with the MULTI COLOR n1,n2,n3 command and the BACKGROUND color. When other colors are used, intersecting lines may "bleed" colors onto one another. Also, when a color other than one of the three principal colors or the background color is selected, it simply replaces <color1> in this command.

Examples:

```
RESET
NEW
10 MULTICOLOR RED, GREEN, BLUE
20 BACKGROUND YELLOW : COLOR GREEN
30 LINE 100,100 TO 200,200
MULTI
RUN
```

Effect: A green diagonal line appears on a yellow background.

Errors: ILLEGAL QUANTITY ERROR

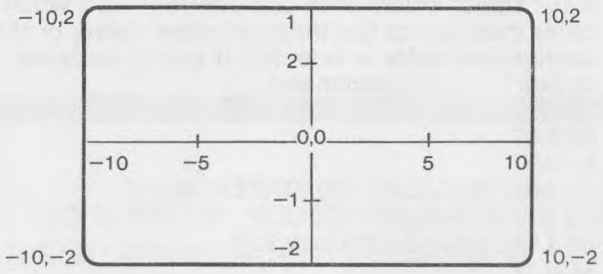
1. The given color numbers are out of the range 0 to 15.

Command Format: MULTI COLOR <color>, <color>,<color>

Demo Program: COLORWHEEL DEMO

SCALE [<x range>,<y range>]

Description: Changes the scale of the HIRES and MULTI screens. For example, the width of the screen can be set to be 1 across or 10,000 across. The same can be done for the vertical scale. Once the scale has been set, the origin can be moved around with the SETORIGIN command. Once the screen has been scaled, the coordinates used for defining the origin are scaled according to the preset scale. The commands SCALE 20,4 and SETORIGIN 10,2 set up the screen to the following scheme.



Demo Programs: SATURN DEMO

See also
SETORIGIN

SETORIGIN [<origin x>,<origin y>]

Description: Sets the origin anywhere on or off the screen. The origin acts as your reference point. You indicate the location of 0,0 of an x,y coordinate system and plot various commands in reference to that location. After you use the SETORIGIN command, all new lines, dots, window settings, and sprites will be positioned with respect to the new origin.

If you were to plot a DOT at (0,0) it would normally appear in the lower left hand corner of the screen. If you were to use the SETORIGIN 100,100 command, the same DOT command would cause a dot to appear at 100,100 relative to the lower left hand corner of the screen.

Typing SETORIGIN with no parameters resets the origin to the lower left corner of the screen. Note that the coordinates given for the origin will be affected by the current SCALE.

Examples: (for illustration only)

```
50 SETORIGIN 100,100
60 LINE -10,0 TO 10,0

50 SETORIGIN 1000,1000
60 LINE 0,0 TO -1000,-950

50 SETORIGIN
60 DOT 0,9
```

Command Format:

```
SETORIGIN
SETORIGIN <x>,<y>
```

Demo Program:

```
ORIGIN DEMO
SETORIGIN DEMO
```

See also
SCALE

WINDOW [<lowerleft x>,<lowerleft y>, <upperright x>,<upperright y>]

Description: Sets up a WINDOW on the screen. This window prevents any lines or dots to be plotted outside the designated area. The coordinates for the WINDOW are always relative to the ORIGIN and affected by SCALE.

This WINDOW feature is useful when drawing figures larger than the screen. When the figure is drawn, it will be as if you are looking at it through a window—only a portion of the drawing appears.

When you first load Graphics BASIC, the WINDOW is defaulted to the same size as the screen. If you type WINDOW (without any bounds), the window will be reset to its default value. You cannot define the WINDOW to be larger than the screen.

Example:

```
RESET
NEW
10 RESET : HIRES
30 FOR J=1 TO 50 : K=J*3+10 : LINE 0,K TO
   320,K : NEXT J
RUN
To see what WINDOW does, add the following lines
(press f7 to switch to the text screen):
20 FOR L=1 TO 3 : READ A,B,C,D : WINDOW A,B,C,D
40 NEXT L
50 DATA 10,40,120,180 : REM 1ST WINDOW
60 DATA 180,50,310,160 : REM 2ND WINDOW
70 DATA 20,10,300,36 : REM 3RD WINDOW
RUN
```

Example:

```
NEW
RESET
10 HIRES
20 HIRES COLOR WHITE ON BLACK
30 WINDOW 180,50,310,160
40 LINE 0,130 TO 360,120
RUN
```

Effects: There will be a long line starting at the left edge of the screen extending to the right side.

Command Format:
WINDOW
WINDOW <x>,<y>,<x>,<y>

Demo Program:
WINDOW DEMO
WIND/CIRC DEMO

BACKGROUND (<sprite number>)

Description: Checks for sprite to background collision. If the given sprite has been in contact with any characters or graphics, the function will be true—a signal that a collission has occurred. Do not confuse this command with the BACKGROUND command that controls the background color. Notice the required parentheses around <sprite number>.

Example:
(for illustration only)

```
10 IF BACKGROUND (2) THEN PRINT "IT HIT!"
```

As with the SPRITE(<sprite number>) function, the sprite's background collision flag will be cleared after each use of the function, or when SPRITE CLEAR HIT is performed.

NOTE: Sprite to background collisions work identically to sprite to sprite collisions. Thus to check if a sprite has or is touching a background, simply use the BACKGROUND(n) command. If you want to check if a sprite is in contact with the background at an instant use the SPRITE CLEAR HIT command.

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range. The number should be 1 to 8.

Demo Program: BOUNCE BALL DEMO
AIRPLANE DEMO

See also
SPRITE CLEAR HIT

SPRITES

COPY SPRITE <sprite number> TO HIRES/MULTI

Description: Displays pre-defined sprite shapes on the HIRES or MULTI screens. For example, by moving a sprite with the shape of a man around and repeatedly executing the COPY SPRITE command, you will create many identical men on the HIRES or MULTI screen. You can produce interesting effects when copying sprites several times while they are both moving and animating automatically. To erase sprites, simply COPY them when the sprite is turned off.

NOTE: When copying multicolor sprites to the multicolor screen, the screen multicolors must be set as follows for the color to be copied properly:

given SPRITE MULTICOLOR <sprite color1>, <sprite color2> set MULTICOLOR <any color>, <sprite color1>,<sprite color2>

Examples: (for illustration only)

COPY SPRITE 1 TO HIRES

COPY SPRITE N TO MULTI

See the examples under the COPY HIRES TO SPRITE command on page 32.

Demo Program: HATMAN RACE DEMO (demonstrates MULTI)
SPACEMAN DEMO (demonstrates HIRES)

EDIT

Description: Clears the screen and displays a grid indicating that you are now running the Graphics BASIC built-in sprite editor. This editor allows you to design your own hires and multicolor sprite shapes for use in your own programs. The grid on the screen is an enlarged representation of the shape you are presently editing. Using the standard cursor control keys along with a few other specially defined keys, you can quickly design and manipulate shapes. A special shape buffer can hold a shape temporarily. This is useful when you wish to copy or move a sprite shape from one place to another.

EDIT (Continued)

Following is a list of all the recognized keys and their functions within the Graphics BASIC sprite shape editor:

[right CRSR]	Moves the cursor right.
[shift] [left CRSR]	Moves the cursor left.
[shift] [up CRSR]	Moves the cursor up.
[down CRSR]	Moves the cursor down.
[RETURN]	Moves the cursor to the left of the next line.
[period]	Plots a point in the present color and moves right.
[space bar]	Erases a point and moves right.
[shift CLR/HOME]	Clears the present sprite shape.
[CTRL] R	Reverses the bit pattern of the present shape.
[CLR/HOME]	Moves the cursor to the upper left corner.
[INST/DEL]	Moves to the left and deletes a dot.
Q	Terminates the editor.
+	Edit next shape.
−	Edit preceding shape.
X	Toggle the X size.
Y	Toggle the Y size.
←	Flip the sprite horizontally.
↑	Flip the sprite vertically.
@	Scroll sprite up.
/	Scroll sprite down.
:	Scroll sprite left.
;	Scroll sprite right.
£	Rotate sprite clockwise.
M	Toggle the mode between HIRES and MULTICOLOR
S	Save the present shape in the shape buffer
C	Copy the shape buffer contents to present shape
f1	Set current color to multicolor 1
f2	Choose color for multicolor 1
f3	Set current color to sprite color
f4	Choose color for sprite color.
f5	Set current color to multicolor 2.
f6	Choose color for multicolor 2.
f7	Set current color to background color.
f8	Choose color for background.

EDIT (Continued)

When you leave the sprite editor, the screen clears and you return to normal Graphics BASIC control with both your program and variables unaltered. Save your sprite shapes with the SPRITE SAVE command.

Before the sprite editor is entered, an equivalent to the RESET command is performed. Therefore, all animators and sound will clear.

The sprite editor ignores all unassigned keys and returns no BASIC error messages.

The EDIT command can be used in a program with a line number:

```
20 EDIT : PRINT"DONE"
30 END
```

Example: EDIT

If you do not have any sprites loaded into the sprite editor, you will see a "garbage" or undefined sprite in the sprite editing area. Clear the sprite editing area by pressing the SHIFT and CLR/HOME keys at the same time. Move from one sprite shape to another with the + and − keys. Use the . (period) to plot a point, and the space bar to erase a point. Leave the sprite editor by typing the letter **Q**.

NOTE: Some older models of the Commodore 64 cause flickering in the sprite edit area.

NOTE ON EDITING MULTICOLOR SPRITES

You can create multicolor sprites using up to four colors—the background color, the sprite color, and two multicolors. However, you have only half the horizontal resolution. In the sprite editor, you are working with 12 pairs of dots across in multi mode instead of 24 dots in hires mode. Each pair of dots represents one of the four colors as follows:

background color	. . (two blanks)
multicolor 1	X X (two solids)
multicolor 2	. X (one blank, one solid)
sprite color	X . (one solid, one blank)

You do not need to worry about working with pairs of dots—the multicolor sprite editor takes care of this for you. First, type **M** to enter the multicolor mode in the sprite editor, and notice that the cursor is now < > instead of #. Also notice that the sprite has turned to

EDIT (Continued)

three colors—black, white, and purple. (Black represents multicolor 1, white represents the sprite color, and purple represents multicolor 2.) You can select which colors you want to use as listed below.

Think of the function keys as a palette. Pressing the function key without shifting selects the color (background, sprite color, multicolor 1, and multicolor 2) you want to work with. Pressing the shifted function key cycles through the available colors; stop when you reach the desired color. The color is shown in the "current color" square at the top of the screen.

f1 Selects multicolor 1.
f2 Chooses the color to be used for multicolor 1.

f3 Selects sprite color.
f4 Chooses the color to be used for sprite color.

f5 Selects multicolor 2.
f6 Chooses the color to be used for multicolor 2.

f7 Selects background.
f8 Chooses the background color.

Once the sprite is plotted, you can cycle through the colors again (using the shifted function keys) to select the best combination of colors. When you leave the sprite editor or save the sprites, the actual colors you selected will not be saved (sprites are saved with the default colors of black, purple, and white). When putting the multicolor sprites into a program, use the SPRITE COLOR, SPRITE MULTICOLOR, and BACKGROUND commands to set their colors.

SPRITE(<sprite number>)

Description: Checks for sprite to sprite collision. If the specified sprite has *previously* come in contact with any other sprite, the function will be true.

Example: (for illustration only)

```
60 IF SPRITE(3) AND SPRITE(5) THEN GOSUB 1000
```

The Commodore 64 is unable to detect *which* sprite has collided with *which* sprite. Therefore, the IF . . . THEN comparison in the above example would be true if sprite 3 was in contact with sprite 8 and sprite 5 was in contact with sprite 2. Then both sprites would have their sprite to sprite flags set, but this would not necessarily mean that they were in contact with *each other.*

Once this function has been called, the sprite to sprite collision flag will be cleared. It will be set again when the sprite collides with another sprite.

NOTE: Caution must be taken when checking the sprite-to-sprite collision of moving sprites. When two sprites collide, their collision flags are set within the computer. When the two sprites move away from each other, however, their flags will still indicate that they have collided. To clear these old values, simply execute the SPRITE CLEAR HIT command.

If you want to check if a sprite *has* hit, or *is* hitting another sprite, simply use the SPRITE(<n>) command. If you want to know if a sprite is in contact with another sprite at any instant, first execute the SPRITE CLEAR HIT command.

Example: Type RESET and NEW.
Enter the sprite editor and create a sprite shape to use in the example:

EDIT	(enter sprite editor)
SHIFT CLR/HOME	(clears the edit area)
CTRL R	(reverses the bit pattern, that is, all bits on instead of off, and creates a sprite shape)
Q	(leave sprite editor)

SPRITE (Continued)

```
10 SPRITE 1 ON AT 150,100 SPEED 1,0 COLOR
   WHITE SHAPE 1
20 SPRITE 2 ON AT 300,100 SPEED -4,0 COLOR
   BLACK SHAPE 1
30 SPRITE MOVE
40 FOR T=1 TO 100 : NEXT T : SPRITE CLEAR HIT
50 IF SPRITE(1) AND SPRITE(2) THEN SPRITE 1
   SPEED -1,0 : SPRITE 2 SPEED 4,0
55 GOTO 70
60 GOTO 50
70 FOR T=1 TO 100 : NEXT T : SPRITE CLEAR HIT
80 IF SPRITE(1) AND SPRITE(2) THEN SPRITE 1
   SPEED 1,0 : SPRITE 2 SPEED -4,0
85 GOTO 40
90 GOTO 80
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not in the range of 1 to 8.

Demo Program: BALL&BOX DEMO

SPRITE <sprite number> ANIMATE ON

SPRITE <sprite number> ANIMATE OFF

Description: Allows the animation of individual sprites to be turned on and off. When the animation is turned on using this command, the sprite will flip between the specified sprite shapes at the given rate (set with the SPEED command) ONLY after the SPRITE MOVE command has been entered.

Example:

```
RESET
NEW
  3 SPRITE LOAD "GRBASIC.SPR"
  5 HIRES
10 SPRITE 1 ON AT 100,100 COLOR WHITE SHAPE 1
20 SPRITE 1 ANIMATE OFF
30 FOR X = 1 TO 8
40 SPRITE X ANIMATE 1,2,3,4,5 SPEED 10
50 SPRITE X ANIMATE ON
60 NEXT X
70 SPRITE MOVE
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range.

Demo Program: HATMAN DEMO
ANIMATE DEMO

SPRITE <sprite number> **ANIMATE** <shape1>, [<shape2>] . . . **SPEED** <speed>

Description: Selects the shapes to be flipped between when animation is on. If, for example, shapes 10, 11, 12, 13, and 14 were all different shapes and you wanted to flip between the shapes very quickly, you would type:

SPRITE 1 ANIMATE 10,11,12,13,14,13,12,11 SPEED 1

This command simply changes the shape of the given sprite automatically from 10 to 11 to 12 and on through the sequence to 11, and then repeats the sequence. A small number for the speed will cause it to flip through the given shapes very quickly, while a large number will cause the shape of the sprite to change slowly. Zero is the fastest, 127 is the slowest. A speed of 60 will flip the shape approximately once every second. The number of shapes in a sprite's animation sequence cannot exceed 16.

NOTE: The sprite will not begin to change between shapes until the SPRITE n ANIMATE ON and SPRITE MOVE commands are executed.

Examples:

```
RESET
NEW
10 SPRITE LOAD"HATMAN.SPR"
20 HIRES
30 SPRITE 1 ON AT 100,100 COLOR WHITE SHAPE 1
40 SPRITE 1 ANIMATE 1,2,3,4,5,6,7,8,9  SPEED 30
50 SPRITE 1 ANIMATE ON : SPRITE MOVE
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range, <shape> is out of range, or <speed> is out of range.

Demo Program: 8BALLS DEMO
ANIMATE DEMO

SPRITE <sprite number> **AT** <sprite x>,<sprite y>

Description: Positions sprites at specific locations on the screen. <sprite number> must be an integer from 1 to 8. The coordinates are always relative to the origin as set by the SETORIGIN command and affected by the SCALE. This should be taken into account when positioning sprites. Also, when the coordinates <sprite x>, <sprite y> exceed the boundaries of the screen, the sprite will wrap around to the other side. The WINDOW command has no effect on sprites.

Example:

```
NEW
5 RESET : HIRES
10 SPRITE 1 ON COLOR WHITE SHAPE 10
20 SPRITE 1 AT 100,146
RUN
```

Effects: Sprite 1 will appear as shape 10 and will be white. The location where it will appear is (100,146).

Command Format: SPRITE <number> AT <x>,<y>

Demo Program: BOUNCE BALL DEMO

SPRITE [<sprite number>] **CLEAR HIT**

Description: Clears the sprite collision status for the specified sprite, or for all eight. By executing this command immediately before the SPRITE (n) or BACKGROUND (n) functions are checked, the immediate status of the sprite's collision is returned. Otherwise, the SPRITE (n) and BACKGROUND (n) function will indicate the collision status since it was last cleared and not at the instant the function is executed. Thus, to check if a sprite has hit, simply use the collision functions normally. To check if a sprite is hitting, use SPRITE CLEAR HIT first.

Command Format:

SPRITE CLEAR HIT

SPRITE <n> CLEAR HIT

Example: See the example listed under the SPRITE (<sprite number>) command on page 50.

SPRITE <sprite number> COLOR <sprite color>

Description: Sets the color of <sprite number> to <sprite color>. Either the color number or the color name may be used. If the sprite is in HIRES mode, the whole sprite will appear in the given color. With multicolor sprites, this command only sets one of the sprite colors. The other two sprite colors are selected using the SPRITE MULTICOLOR command.

Examples: (for illustration only)

```
10 SPRITE 1 COLOR RED

10 X=7
20 SPRITE X COLOR 4
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> or <sprite color> is out of range.

Command Format: SPRITE <number> COLOR <color>

SPRITE <sprite number> HIRES

Description: Puts an individual sprite into high resolution mode.

Examples: (for illustration only)

```
10 SPRITE 3 HIRES

10 SPRITE 1 HIRES
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range.

Demo Program: 8BALLS DEMO

SPRITE LOAD "<file name>" [,<device number>]

Description: Loads sprite shapes which have been saved using the SPRITE SAVE command. The SPRITE LOAD command will load from the disk unless a device number other than 8 is specified. The SPRITE LOAD will bring stored shapes back into the same place from which they were originally saved. In other words, if you were to save shapes 10 through 17, for example, then the 8 shapes saved on disk would load back into shapes 10 through 17. You can use the SPRITE LOAD command within a program. The sprites will be loaded into the sprite editor and program execution continues with the next BASIC statement.

Example:

```
10 SPRITE LOAD "GRBASIC.SPR"
RUN
```

Effects: The sprite shape data stored on the disk will be loaded into memory. View the sprites by typing EDIT.

If the SPRITE LOAD encounters a disk error, it will abort the load and return with no error messages. The only indication that an error has occurred will be the flashing red light on your disk drive.

NOTE: The SPRITE LOAD command can be used to vector-load machine language programs without affecting BASIC memory pointers.

Command Format:

SPRITE LOAD "<file name>"
SPRITE LOAD "<file name>", <device number>

SPRITE MOVE and SPRITE FREEZE

Description: These two commands act as master controls for all sprite animation and movement. When you use the SPRITE FREEZE command, no sprite movement or animation will occur until you enter the SPRITE MOVE command. At that time, all sprites with a pre-defined speed (set with SPRITE n SPEED x,y) will start to move and all sprites with pre-defined animation sequences (set with SPRITE n ANIMATE n,n,n,. . . SPEED x) will begin to animate if they were told to do so with the SPRITE n ANIMATE ON command.

This master control of animation and movement is provided so that the movement and animation of several sprites may be synchronized. Otherwise, they would take off as soon as their command was given, irrespective of what other sprites were doing. Once you use the SPRITE MOVE command, however, any of the SPRITE n SPEED x,y or SPRITE n ANIMATE 1,2,3 . . . SPEED x, or SPRITE n ANIMATE ON/OFF commands will immediately take effect.

Make sure the Graphics BASIC disk is in the drive. Try the following sequence:

```
10 SPRITE LOAD "GRBASIC.SPR"
20 SPRITE 1   SHAPE 1 ON AT 10,30
30 SPRITE 1   SPEED 1,1
40 SPRITE MOVE
50 SPRITE 1 ANIMATE 10,11,12,11,10 SPEED 2
60 SPRITE 1 ANIMATE ON
RUN
```

Demo Programs: DRIVING DEMO

SPRITE <sprite number> MULTI

Description: Puts an individual sprite into multicolor mode. The sprite chosen will appear with colors corresponding to those set with the SPRITE <sprite number> COLOR <sprite color> and SPRITE MULTICOLOR <multicolor1>, <multicolor2> commands.

Examples: (for illustration only)

```
10 SPRITE 1 MULTI
10 SPRITE 3*2 MULTI
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range.

Demo Program: MULTIHATMAN DEMO

SPRITE MULTICOLOR <sprite multicolor1>, <sprite multicolor 2>

Description: Defines the two colors which may be used with multicolor sprites.

Example: First, type RESET and load the pre-defined sprites stored on the disk as follows:

SPRITE LOAD"HATMAN.SPR"

and then type in the following short program:

```
10 SPRITE MULTICOLOR RED, WHITE
20 SPRITE 1 ON AT 100,100 SHAPE 10 SPEED 1,0
30 SPRITE 1 MULTI
40 SPRITE 1 COLOR YELLOW
50 SPRITE 1 ANIMATE 1,2,3,4,5,6,7 SPEED 5
60 SPRITE 1 ANIMATE ON
70 SPRITE MOVE
RUN
```

Change the colors in lines 10 and 40 to see how the colors work when you RUN the program again.

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range.
2. <sprite multicolor1> or <sprite multicolor2> is out of range.

Demo Programs: PENCIL DEMO

SPRITE <sprite number> ON or SPRITE <sprite number> OFF

Description: Turns individual sprites on and off. <sprite number> must be an integer from 1 to 8. When a sprite is turned on, the sprite's present shape will be displayed on the screen at the present sprite position. When a sprite is turned off, it still retains its shape, position, and other parameters. It simply is no longer displayed.

Example:

```
NEW
 10 SPRITE LOAD "HATMAN.SPR"
 20 HIRES:HIRES COLOR BLUE ON GRAY1
 30 PRINT "(CLR)" : REM SHIFTED CLR/HOME
    KEY IN QUOTES
 40 FOR SP = 1 TO 8
 50 PRINT "CURRENT SPRITE IS ";SP
 60 HIRES FROM 10
 70 PRINT "(CLR)"
 80 IF SP=8 THEN SPRITE 8 ON:END
 90 SPRITE SP OFF
100 FOR SP = 1 TO 8
110 PRINT "CURRENT SPRITE IS ";SP
120 SPRITE SP ON AT 100,100
130 FOR DELAY = 1 TO 500:NEXT DELAY
140 IF SP=8 THEN SPRITE 8 ON:END
150 SPRITE SP OFF
160 NEXT SP
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not an integer from 1 to 8.

Command Format:

SPRITE <number> ON
SPRITE <number> OFF

SPRITE <sprite number> ON BACKGROUND

SPRITE <sprite number> UNDER BACKGROUND

Description: Controls the priority of an individual sprite with respect to the background.

Examples: (for illustration only)

```
10 SPRITE 3 ON BACKGROUND
```

Effects: This will result in sprite number 3 appearing *on top* of any characters or graphics that appear at the same place on the screen. In other words, the sprite will be in front of everything else.

```
10 SPRITE 3 UNDER BACKGROUND
```

Effects: This will cause sprite 3 to appear underneath, or behind, all other graphics and characters.

A sprite's priority with respect to the characters and graphics (as set with these two commands) will *not* change the sprite to sprite priority. (i.e. Sprite 1 will *always* appear in front of other sprites.)

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not in the range of 1 to 8.

Demo Program: PRIORITY DEMO

SPRITE SAVE <first shape> , <last shape> , "<file name>" [, <device number>]

Description: Saves sprite shape data on cassette or disk so it will not be lost when the computer is turned off. With the SPRITE SAVE command, any sequential group of sprite shapes may be saved to be retrieved later using the SPRITE LOAD command. Note that the default device number is 8 rather than 1 as with the conventional SAVE.

Example: (for illustration only)
10 SPRITE SAVE 13,13 "MY-SPRITES"

Effect: Sprite shape 13 will be saved on disk.

Example:
10 SPRITE SAVE 1,17 "GAME-SPRITES",1

Effect: Sprite shapes 1 through 17 will be saved on cassette.

Example:
A$="2 SPRITES"
SPRITE SAVE 1,2 A$,10

Effect: Sprite shapes 1 and 2 will be saved on disk device 10, with the file name 2 SPRITES.

If the SPRITE SAVE encounters a disk error, it will abort the save and return to BASIC control *without* an error message. The red light on your disk drive will flash, and this will be the only indication that something has gone wrong.

Command Format:
SPRITE SAVE <shape number>,<shape number> "filename"
SPRITE SAVE <shape number>,<shape number> "filename",<device>

SPRITE <sprite number> **SHAPE** <shape number>

Description: Assigns a shape to the given sprite. Space for 32 shapes are originally allocated in the sprite editor. Any sprite can select any one of the available shapes as well as share any shape with any other sprite.

Here is the formula for figuring out where a sprite shape is in memory:

First location of shape = (shapenumber + 31) * 64

NOTE: Shape 1 starts at memory location 2048. This shape data occupies the space immediately preceding your BASIC program, so care must be taken when POKEing data directly into this area.

Example:
(for illustration only)

```
5 A=2
10 SPRITE 1 SHAPE 4
20 SPRITE A SHAPE 4
30 SPRITE 8 SHAPE 16+3
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not in the range of 1 to 8.
2. <shape number> does not reference a valid location in memory.

Command Format: SPRITE <number> SHAPE <number>

SPRITE <sprite number> SPEED <x speed>,<y speed>

Description: Defines the <x speed> and <y speed> of the given sprite. Values of 0,0 will cause the sprite to remain still, while 5,5 will cause it to move rapidly to the upper-right.

NOTE: Movement will not actually take place until the SPRITE MOVE command is executed.

Examples: (for illustration only)

```
10 SPRITE 1 SPEED -1,0
10 SPRITE 1 SPEED -3,1
10 A=8
20 SPRITE 4 SPEED A,-A*6
```

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is out of range.
2. <x speed> or <y speed> is out of range. (127 to -128)

Demo Program: HATMAN RACE DEMO

SPRITE <sprite number> XYSIZE <x factor>,<y factor>

Description: Sets the horizontal and vertical size of any sprite. An <x factor>,<y factor> of 1,1 will set the sprite to its normal size, while a value of 2,2 will make it twice as large in horizontally and vertically.

Examples: (for illustration only)

```
10 SPRITE 3 XYSIZE 2,2
10 SPRITE 3 XYSIZE 1,2
```

Errors: ILLEGAL QUANTITY ERROR

1. <x factor> or <y factor> is not a 1 or 2.
2. <sprite number> is not in the range of 1 to 8.

XPOS (<sprite number>)

Description: Returns the x coordinate of the specified sprite. The returned value is affected by the ORIGIN and SCALE.

Example: 10 X=XPOS(2)

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not in the range 1 to 8.

Demo Program: AIRPLANE DEMO

YPOS(<sprite number>)

Description: Returns the y coordinate of the specified sprite. The returned value is affected by the ORIGIN and SCALE.

Example: 10 Y=YPOS(5)

Errors: ILLEGAL QUANTITY ERROR

1. <sprite number> is not in the range 1 to 8.

Demo Program: AIRPLANE DEMO

SOUND

SOUND CLEAR

Description: Resets the WAVE, VOLUME, TONE, and ADSR values for all three voices. This is useful at the beginning of programs to ensure that the sound generator has been reset.

Example: (for illustration only)

```
10 SOUND CLEAR
```

SOUND GO and SOUND FREEZE

Description: These two commands act as master controls for all automated sound playing. When the SOUND FREEZE command is executed, all automated sound sequences will stop until the SOUND GO command is executed. At this time, all pre-defined sound sequences which have been turned on will start to play.

This master control of automated sound is provided so that several voices may be synchronized. Otherwise, the various voices would begin playing their sequence as soon as their voice go command was given. Once the SOUND GO command has been executed, the VOICE <voice number> PLAY command will immediately take effect irrespective of what any other voices may be doing.

Demo Program:

3-PART SONG DEMO
3-VOICE DEMO

SOUND ON and SOUND OFF

Description: Turns the master volume down to zero. When the SOUND ON command follows it, however, the volume level is restored to its original value. If the SOUND ON command is executed before the SOUND OFF command, the master volume will be set to its default level of zero.

Example: (for illustration only)

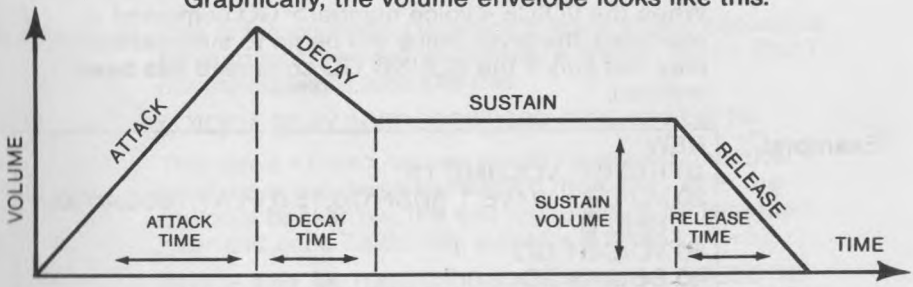
```
10 VOLUME 10
20 SOUND OFF

110 SOUND ON
```

Demo Program:

3-VOICE DEMO

VOICE <voice number> ADSR <attack>, <decay>,<sustain>,<release>

Description: The values given in this command determine the form of the volume envelope for the particular voice. Graphically, the volume envelope looks like this:



When the VOICE <voice number> ON command is executed, the attack will be triggered causing the volume of the voice to rise at the given <attack> rate. After it has reached peak volume, the volume will start to fall at a rate determined by the <decay> setting. The voice will then remain at the <sustain> volume setting until the VOICE <voice number> OFF command is executed. At this time, the volume level will fall at the pre-defined <decay> rate until it is no longer heard.

It should be noted that the <attack>,<decay>, and <release> values all determine the *rate* at which the volume level changes, while the <sustain> value is simply a *volume setting.* <attack>,<decay>, and <release> rates all vary from 0 for an instantaneous change to 15 which will cause the volume level to slowly change over a period of several seconds.

Example:

```
NEW
10 RESET : VOLUME 15
20 VOICE 1 WAVE 1 ADSR 6,0,15,0 TONE 10000
30 VOICE 1 ON
RUN
```

The command SOUND OFF will stop the sound. The command SOUND CLEAR will reset the ADSR values to zero.

Errors: ILLEGAL QUANTITY ERROR

1. <voice number> is out of range.
2. One of the ADSR settings is not in the range 0 to 15.

Demo Program:
BOUNCE BALL DEMO
3-PART SONG DEMO
ADSR DEMO

VOICE <voice number> GO and VOICE <voice number> FREEZE

Description: These two commands allow the automated playing of individual voice sequences to be turned on and off. When the VOICE <voice number> GO command is executed, the given voice will begin to automatically play, but *only* if the SOUND GO command has been entered.

Example:

```
NEW
10 RESET : VOLUME 15
20 VOICE 1 WAVE 1 ADSR 0,0,15,0 PLAY 1000,4000,←
   SPEED 10
30 VOICE 1 GO
40 SOUND GO
RUN
```

Stop the sound by typing SOUND FREEZE directly and type SOUND GO to turn it on again.

Errors: ILLEGAL QUANTITY ERROR

1. <voice number> is not in the range of 1 to 3.

VOICE <voice number> **PLAY** [CONT] <tone> [;<note duration>] [;<release time>] , <tone> [;<note duration>] [;<release time>], . . . [←] **[SPEED** <speed>]

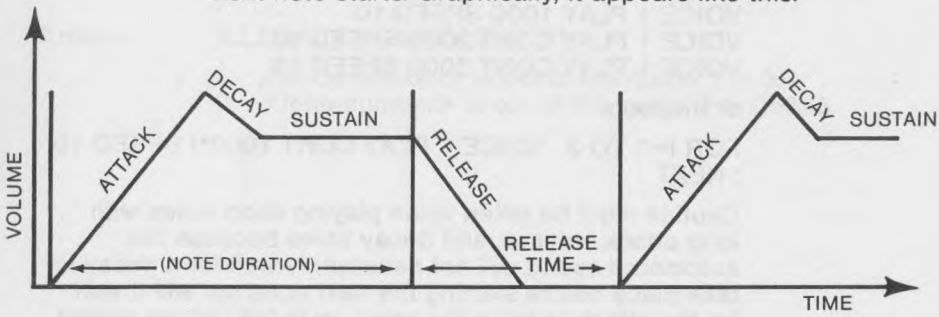
Description: This is the command in which a voice's tone sequence for automated sound is defined. A typical VOICE PLAY command might look like this:

VOICE 1 PLAY 1000,2000,3000,4000 SPEED 10

The listed <tone> values would be played one after another at the given speed and then stop. With the optional backarrow, the sequence would be repeated over and over. To do this, you would type

VOICE 1 PLAY 1000,2000,3000,4000, ← SPEED 10

By using the optional <note duration> and <release time> parameters, a rhythm can be created allowing certain notes to play longer than others. The <note duration> value defines the time between the moment the attack is started to the moment the release is *started.* The <release time> value defines the time from the moment the release is started to the moment the next note starts. Graphically, it appears like this:



You can figure out the number of seconds between the start of each note by using the following formula:

time = (<note duration>+<release time>+1)* (<speed>+1)/60

So, for example, to create a series of short notes one second apart, you would use the command:

VOICE 1 PLAY 2000;1;2, ← SPEED 14

With voice one set as follows:

VOICE 1 WAVE 1 ADSR 0,2,10,2

VOICE – PLAY (Continued)

Example: If you wished to make the notes different tones and lengths you could set up your automated sound table with this program:

```
NEW
10 SOUND CLEAR : VOLUME 15
20 VOICE 1 WAVE 1 ADSR 1,0,15,7
30 FOR I=1 TO 11
40 T=4000*(2↑(I/12) ) :REM CHROMATIC SCALE
50 D=INT (30*RND (1)+1)
60 VOICE 1 PLAY CONT T;D;1, ← SPEED 0
70 NEXT I
80 VOICE 1 GO : SOUND GO
RUN
```

Note that with the CONT inserted after the PLAY that the notes contained in the sequence may be defined using separate PLAY commands, or just one command contained within a loop as was done here. For example, the command

VOICE 1 PLAY 1000,2000,3000 SPEED 10

is functionally identical to the commands

VOICE 1 PLAY 1000 SPEED 10
VOICE 1 PLAY CONT 2000 SPEED 10
VOICE 1 PLAY CONT 3000 SPEED 10

or the loop:

FOR I=1 TO 3 : VOICE 1 PLAY CONT 1000*I SPEED 10 : NEXT

Caution must be taken when playing short notes with long attack, release, and decay times because the automated sound will not necessarily wait for a decay to take place before starting the next note, nor will it wait for the attack to bring the voice up to full volume before starting the release.

Errors: ILLEGAL QUANTITY ERROR

1. Number of notes to be played in any one voice exceeds 63.

Demo Program: 3-PART SONG DEMO
BOUNCE BALL DEMO

VOICE <voice number> **TONE** <tone number>

Description: Sets the tone of the given voice. Changing <tone number> will cause the voice to play different notes. For example, a <tone number> of 7382 will produce a 440 Hz tone which is the note A in concert pitch. The back of the manual contains a list of <tone number> values which correspond to various notes, as well as a formula for computing the <tone number> for any of the 12 semitones in any of the 7 available octaves.

Example:

```
NEW
10 VOLUME 15
20 VOICE 1 TONE 6000 ADSR 0,0,15,0
30 VOICE 1 ON
RUN
```

```
10 VOLUME 15
20 FOR I=1 TO 3
25 VOICE I WAVE TRIANGLE
30 VOICE I TONE 2000*2 ↑I
40 VOICE I ADSR 0,0,15,0 ON
50 NEXT I
RUN
```

Errors: ILLEGAL QUANTITY ERROR

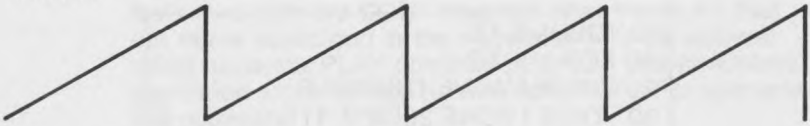
1. <voice number> is out of the range 1 to 3.
2. <tone number> is out of the range 0 to 65535.

VOICE <voice number> WAVE <wave number> [,<pulse width>]

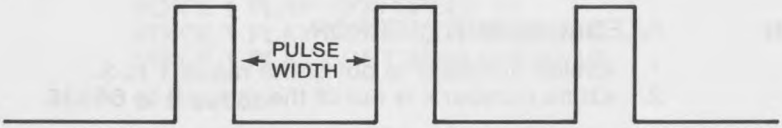
Description: Selects the waveform for a particular voice. Following are the available waveforms and their corresponding WAVE commands:

WAVE 1 or WAVE TRIANGLE
Triangle

WAVE 2 or WAVE SAW
Sawtooth



WAVE 3,<pulse width> or WAVE PULSE,<pulse width>
Pulse



WAVE 4 or WAVE NOISE
Noise

Example: (for illustration only)

```
10 VOICE 2 WAVE 1

10 W=4
20 VOICE 1 WAVE W

10 VOICE 1 WAVE 3,545
```

NOTE: The optional [,<pulse width>] is only valid when selecting WAVE 3. This value determines the duration of the pulse, and may range from 0 to 1024.

Errors: ILLEGAL QUANTITY ERROR

1. <voice number> is not a 1,2, or 3.
2. <wave number> is not in the range of 1 to 4.
3. <pulse width> is not in the range of 0 to 1024.

SYNTAX ERROR

1. A <pulse width> does not follow wave 3.

Demo Program: AIRPLANE DEMO
3-PART SONG DEMO
BOUNCE BALL DEMO

VOLUME <level>

Description: Acts as a master volume control for all three voices. Decreasing the volume level with this command has the same effect as turning down the volume on your TV or monitor. The <level> ranges from 0 to 15.

Example: (for illustration only)

```
10 VOLUME 0
```

```
10 VOLUME 10
```

```
10 FOR I=0 TO 15
20 VOLUME I
30 NEXT I
```

Errors: ILLEGAL QUANTITY ERROR

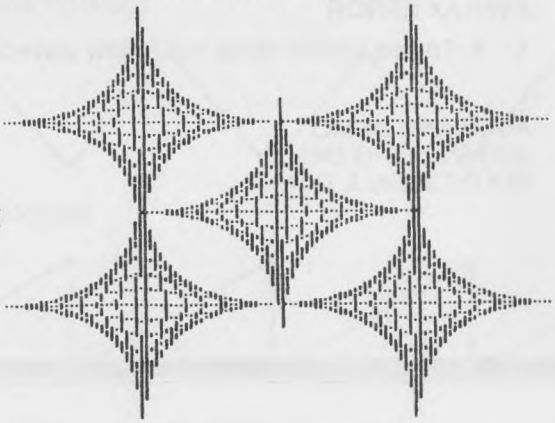
1. <level> is not in the range of 0 to 15.

Demo Program: 3-PART SONG DEMO
3-VOICE DEMO

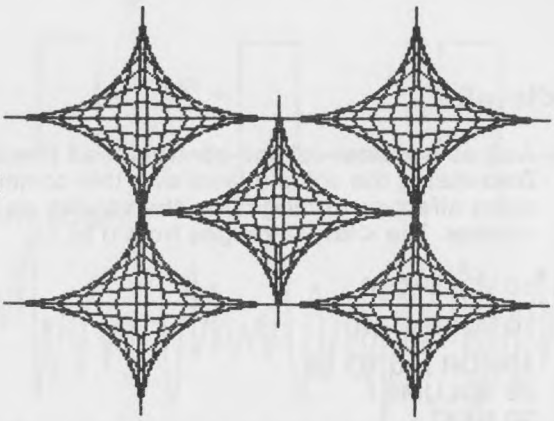
COPY HIRES/MULTI TO PRINTER

Description: Prints the present high resolution or multicolor image on the HIRES or MULTI screen on a 1525 Graphics Printer or Gemini series printer. Note that only graphics within the WINDOW will be printed. Pressing the RUN/STOP key stops the printing.

MULTI IMAGE



HIRES IMAGE



NOTE TO GEMINI PRINTER OWNERS: Before using the COPY HIRES/MULTI TO PRINTER command, type in the following number:

SYS 32512 <RETURN>

Examples: (for illustration only)

COPY HIRES TO PRINTER

COPY MULTI TO PRINTER

DIR [<device number>]

Description: Prints out the disk directory of the given device. If no device number is specified, device 8 is assumed. This command is equivalent to:

```
LOAD"$",<device #>
LIST
```

but it does not affect the program in memory. To slow the scrolling of the directory listing, hold down the CTRL key. Pressing the RUN/STOP key stops the scrolling and there is no continue function.

Examples:

```
DIR
DIR 9
```

If you were to read the directory of the Graphics BASIC disk, you would see several files listed. To distinguish between the various types of files the following conventions were used:

demo programs:	<demo name> DEMO
sprites:	<sprite data name> . SPR
character sets:	<char set name> . FONT
screen graphics:	<screen name> . GRPIC

DISK "<command string>" [,<device number>]

Description: Standard Commodore disk drive commands may be easily sent to the disk drive addressed by <device number> with this command. If no device number is given, device 8 is assumed.

The standard disk drive commands are

N0	for NEW
C0	for COPY
R0	for RENAME
S0	for SCRATCH
I	for INITIALIZE
OPEN	for OPEN
CLOSE	for CLOSE

See the section on disk commands in your disk drive manual for further explanations.

Examples:

```
DISK "S0:JAY'S DEMOS",9

DISK "N0:MASTER DISK, IL"

DISK A$
```

NOTE: The DISK command is equivalent to:

```
10 OPEN 37,<device number>,15
20 PRINT#37,<command string>
30 CLOSE 37
```

INPUT/OUTPUT

DISK [,<device number>]

Description: The error status for the disk drive addressed by <device number> is printed on the screen when this command is executed. It is equivalent to:

```
10 OPEN 37,<device number>,15
20 INPUT#37,N,N$,T,S
30 PRINT N,N$,T,S
40 CLOSE 37
```

The default device number is 8.

Examples:

```
DISK
DISK, 9
```

JOY(<joystick number>)

Description: This function returns a value corresponding to the present position of the given joystick. The values returned correspond to the various positions as follows:

```
1 ..... DOWN AND LEFT
2 ..... DOWN
3 ..... DOWN AND RIGHT
4 ..... LEFT
5 ..... NEUTRAL
6 ..... RIGHT
7 ..... UP AND LEFT
8 ..... UP
9 ..... UP AND RIGHT
```

Example:

```
NEW
10 N=JOY(1)
20 IF N<0 THEN PRINT"FIRE!!" : N=-N
30 PRINT"X,Y DIRECTION",
40 Y=INT( (N-4)/3 ) : X=N-Y*3-5
50 PRINT X;Y
60 GOTO 10
RUN
```

Errors: ILLEGAL QUANTITY ERROR

1. <joystick number> is not 1 or 2.

Demo Program: JOYSTICK DEMO

HIRES SAVE "<file name>" [,<device number>] and HIRES LOAD "<file name>" [,<device number>]

Description: These commands are used to save and retrieve the image on the high resolution screen. The default device number is 8.

The format of the file created is as follows:

Sequential File

1	Byte : "H"
8000	Bytes: High resolution screen
1000	Bytes: High resolution color screen

Examples: (for illustration only)

```
HIRES SAVE "PIE CHART"
HIRES LOAD "HES AD",9
HIRES LOAD A$
```

MULTI SAVE "<file name>" [,<device>] and MULTI LOAD "<file name>" [,<device>]

Description: These commands save and load the multicolor screen. The default device number is 8. The format of the file created is as follows:

Sequential File

1	Byte : "M"
8000	Bytes: Multicolor screen
1000	Bytes: Multicolor color screen
1000	Bytes: Text color memory

Examples: (for illustration only)

```
MULTI SAVE A$
MULTI LOAD "PICTURE"
```

NOTE: Two multicolor screen files which you can load are on the Graphics BASIC disk. Type:

```
10 RESET
20 MULTI LOAD "FUJ*" : MULTI
30 GOTO 30
```

or

```
10 RESET
20 MULTI LOAD "HES*" : MULTI
30 GOTO 30
```

Press the RUN/STOP key to stop the program, and press f7 to switch to the text screen.

Demo Program: HES BOUNCE DEMO

TEXT SAVE "<file name>" [,<device>]

TEXT LOAD "<file name>" [,<device>]

Description: With these commands, the text screen may be saved and loaded. The default device number is 8. You can design screens using text, such as tables or templates, and save them.

The format of the file created is as follows:

Sequential File

1	Byte : "T"
1000	Bytes: Text characters
1000	Bytes: Text color memory

Examples: (for illustration only)

```
TEXT SAVE "SPACE INVASION"
TEXT LOAD "TABLE",9
TEXT LOAD A$,D
```

CHANGE "<old string>" TO "<new string>"

Description: Finds all occurrences of <old string> and replaces them with <new string>. If <old string> contains a standard BASIC or Graphics BASIC command, it must be enclosed in single quotes rather than the double quotes as shown above. Slashes can also be used to separate the strings. Be sure to put one space before and after the word TO.

Example: (for illustration only)

CHANGE "RON'S" TO "TOM'S"

CHANGE /GOTO/ TO /GOSUB/

CHANGE 'MID$(' TO 'RIGHT$('

NOTE: CHANGE 'PRI' TO 'I' will not change all PRINT commands to INT commands. The whole command must be contained within the single quotes, that is, change 'PRINT' TO 'INT'.

See also
FIND

DO <procedure name> [(<variable1> , . . . , <variable n>)]

Description: Initiates the specified procedure and passes the given variables as parameters. If more variables are given than are defined in the specified procedure, they are ignored. When less than the required number of variables are passed, the remainder of the parameters are set to zero.

Example: (for illustration only)

DO POLYGON(6)

See the example under PROCEDURE.

Demo Program: HATMAN RACE DEMO

See also
PROCEDURE

ELSE <statement> or **ELSE** <line number>

Description: When used in conjunction with the IF . . . THEN command, control is transferred to the ELSE <statement> or <line number> when the IF . . . THEN condition is false. Note that ELSE is a command in itself and must be separated from other commands by a colon.

Examples: (for illustration only)

```
10 IF A=1 THEN PRINT "ONE" : ELSE PRINT "NOT ONE"
10 IF B$="RON" THEN PRINT "GILBERT" : GOTO 100
20 ELSE PRINT "TOM MCFARLANE" : GOTO 100
```

FIND "<search string>"

Description: Prints out all of the lines in your BASIC program containing <search string>. When searching for BASIC or Graphics BASIC commands, the entire command must be given and it must be enclosed in single quotes. The string may also be enclosed in slashes.

Examples: (for illustration only)

```
FIND "HES"
FIND 'PRINT'
FIND/GOSUB/
```

See also
CHANGE

GOTO <line number>

Description: This command has been enhanced to allow branching to line numbers specified by variables, as well as line numbers themselves.

Examples: (for illustration only)

```
GOTO A
GOTO B*4
```

NOTE: The RENumber command will not renumber variable line numbers.

This variable line number modification also works for these commands:

ON GOTO
GOSUB
ON GOSUB
IF . . . THEN
RESTORE and ELSE

KEY (<key number>) = <string>

Description: This command assigns a given function key the specified string. With this command, you can re-define what the function keys perform. No more than 32 characters may be assigned to any one function key.

Examples: (for illustration only)

```
KEY(1) = "DISK" + CHR$(13)
KEY(A) = STR$(A)
KEY (J+1) = "HESWARE IS GREAT"
```

See also
KEY ON
KEY OFF
KEY LOAD
KEY SAVE

KEY LIST

Description: Lists the current operations the function keys contain. To see the current functions type: KEY LIST <RETURN>. In Graphics BASIC, the function keys have been programmed as follows:

f1 RUN <RETURN>
f2 BACKGROUND
f3 LIST <RETURN>
f4 SPRITE OFF
f5 DIR <RETURN>
f6 KEY LIST
f7 TEXT <RETURN>
f8 DISK

NOTE: The **+CHR$(13)** following the assignment string adds a carriage return to the function. Pressing such a function key would be the equivalent of typing the function and pressing return.

You can change the function key assignments to suit your own needs using the KEY, KEY SAVE, and KEY LOAD commands.

KEY LOAD "<file name>" [,<device>]

KEY SAVE "<file name>" [,<device>]

Description: These commands will save and load the present function key assignments. You can change the function key assignments with the KEY command. The default device number is 8. The command KEY LIST displays the current function key assignments.

Examples: (for illustration only)

```
KEY LOAD "MY KEYS"
KEY SAVE A$,9
KEY LOAD "GRKEYS.RAT",1
```

See also
KEY ON
KEY OFF
KEY

KEY ON and KEY OFF

Description: These two commands control the function key string expansion feature of Graphics BASIC. After KEY ON is executed, the string assigned to each function key will be printed when a function key is pressed while in the immediate mode (i.e., while a program is *not* running). After KEY OFF is executed, the function keys behave as they normally do without Graphics BASIC installed.

LIST

Description: The list command in Graphics BASIC operates just as in normal BASIC, with one exception. When the Graphics BASIC LIST command is used within a BASIC program, the program execution is not terminated. In other words, the LIST command can be used anywhere within a program without halting execution. To slow the scrolling of a listing (when you use LIST in direct mode), hold down the CTRL key.

Example:

```
NEW
10 PRINT: PRINT "HERE IS MY PROGRAM"
20 LIST
30 PRINT: PRINT "DONE"
40 END
```

When you type RUN, the following will be printed on the screen:

```
HERE IS MY PROGRAM
10 PRINT: PRINT "HERE IS MY PROGRAM"
20 LIST
30 PRINT: PRINT "DONE"
40 END
DONE
```

ON ERROR GOTO <line number>

Description: Sets the line number to GOTO when any error occurs. Because this command must be executed in order for any errors to be trapped, it is a good idea to place it near the beginning of your programs. Also, it should be noted that when an error actually does occur, and the ON ERROR ON command has been executed, control will be transferred to <line number> and error trapping will be automatically turned off. This is to prevent an error-handling routine from causing an error and calling itself. Thus, after the error has been handled, the ON ERROR ON command needs to be executed if you want to trap more errors.

When an error occurs, the program jumps to the <line number> following GOTO. After control is transferred to <line number>, the variable ER contains the error number, and LI contains the line number in which the error occurred. By checking the values held in ER and LI, you can determine which error has been found and where it is.

If an error occurs during a FOR . . . NEXT loop, the FOR . . . NEXT loop is terminated and cannot be continued.

Example:

```
20 J = 1
30 FOR I = 1 to 999999
40 J = J*I*I*I
50 PRINT J
60 NEXT I
70 END
RUN
```

The line with the error should be listed. Add these lines and RUN the program again.

```
10 ON ERROR GOTO 80
80 PRINT "NUMBERS TOO LARGE"
90 END
```

See page 101 for a table of all possible errors and their corresponding error numbers.

See also
ON ERROR OFF
ON ERROR ON

ON ERROR OFF

Description: Turns off the most recent error found with the ON ERROR GOTO command. After it is executed, all errors will halt program execution and will be displayed on the screen as normal. In addition, Graphics BASIC prints the line on which the error occurred with a reversed arrow inserted at the point within the line which caused the error. This feature is especially useful for debugging.

See also
ON ERROR ON
ON ERROR GOTO

ON ERROR ON

Description: Activates error trapping and prevents your program from crashing. When an error occurs while the ON ERROR ON feature is activated, the program branches to the line number specified by the ON ERROR GOTO <line number> command. If an error occurs while error trapping is on, no error message will be printed. Instead, control will be transferred to the line number of your program which was set by the ON ERROR GOTO <line number> command.

NOTE: If an error occurs before the ON ERROR GOTO <line number> command has been executed, then an undefined statement error will occur. (This is assuming that ON ERROR ON was executed.)

Example:

```
NEW
10 DIM A(10)
20 ON ERROR GOTO 100
30 ON ERROR ON
40 FOR I = 1 TO 20
50 A(I) = I * 100
60 PRINT A(I)
70 NEXT I
80 END
100 PRINT "ERROR IN LINE";LI
110 PRINT "VARIABLE NOT DIMENSIONED HIGH
    ENOUGH"
120 END
RUN
```

See also
ON ERROR GOTO
ON ERROR OFF

PROCEDURE <procedure name> [(<variable 1>, . . . , <variable n>)] . . .

.
.
.

RETURN

Description: Defines a procedure to be initiated by the DO command. A procedure is a subroutine which has a name and the capability to pass variables. Using procedures in your programs eliminates the necessity of having numerous GOTOs or GOSUBs. When a procedure is required, you initiate it with the DO command followed by the name of the procedure. The command RETURN ends the procedure.

Procedure names cannot contain spaces, and procedures cannot be RUN. Procedures are initiated with the DO command.

Up to 10 variables may be passed and they must all be standard floating point variables. (i.e., no integer or string variables are allowed).

Examples:

```
10 PROCEDURE POLYGON (N)                          Type in
15 CLEAR                                           but
20 CIRCLE 100,100 XYSIZE 28,20 STEP 360/N        do not
30 RETURN                                        RUN it.

HIRES TO 15
DO POLYGON(5)                                     Type in
DO POLYGON(8)                                     directly
```

NOTE: (for advanced programmers) By passing an array, recursive procedures can be created.

Demo Program: HATMAN RACE DEMO

See also
DO

REN [<increment>],[<starting line number>]

Description: Resequences the line numbers of the entire program in memory, as well as modifies all line references (such as after the commands GOTO, GOSUB, ON GOSUB, IF . . . THEN, etc.). The default <increment> and <starting line number> values are both 10. Thus if just REN is typed, the program will be renumbered starting at 10 and counting up by 10 for each subsequent line.

Examples: (for illustration only)

REN

REN 1

REN 5,100

NOTE: When using variables for line numbers, care must be taken to ensure that they still address the proper line after the REN has been executed. Commands that can use variables for line numbers are: GOTO, GOSUB, THEN, ELSE, and RESTORE.

RESET

Description: Returns the graphics, sprites, and sound to normal. This command is useful to place at the beginning of a program, or to enter directly before running a program. The RESET command sets the following conditions:

Puts the screen into text mode.
Turns off all eight sprites.
Freezes all sprite animation and movement.
Clears all animation sequences.
Resets the origin to the lower left corner of the screen.
Resets the window to the screen size.
Sets all sprites XYSIZES to 1,1.
Sets all sprites to HIRES mode.
Freezes all automatic sound.
Turns off all three voices.
Clears all automatic sound sequences.
Clears the HIRES and MULTI screens.
Resets the BACKGROUND and BORDER colors.
Clears the TEXT screen.
Resets the SCALE.
Turns ON ERROR to off.
Resets scrolling window.
Resets memory.
Clears variables.
Resets to ROM character set.

Example: (for illustration only)

```
10 RESET
RESET
```

RESTORE [<line number>]

Description: Allows the pointer to the next DATA element to be set to the beginning of the specified <line number>.

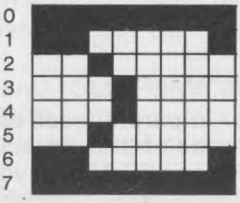
Examples: (for illustration only)

```
RESTORE 100
A=5
RESTORE A
```

NOTE: When the latter form is used as shown in the example above, the RENumber command will not adjust the line number accordingly.

CHAR(<ascii>) = <n>,<n>,<n>,<n>,<n>,<n>,<n>,<n>

Description: Redefines characters to create a RAM character set. The <ascii> number following CHAR is the screen display code of the character you want to redefine. For example, the letter A has the screen display code 1. To redefine the letter A, you would type CHAR(1) followed by the decimal values representing the redefined character. (Refer to pages 132-134 of your *Commodore 64 User's Guide* for a complete list of screen display codes.) A character consists of a block of 64 dots as pictured below where each dot is either on (filled in) or off (blank):



BINARY	DECIMAL
= 11111111	= 255
= 11000001	= 193
= 00100000	= 32
= 00010000	= 16
= 00010000	= 16
= 00100000	= 32
= 11000001	= 193
= 11111111	= 255

The binary number indicates which dots are on (1) and which dots are off (0). Each dot (bit position) has a power of 2. To calculate the decimal number, take the bit positions with the value of 1 and find the corresponding column number (the power-of-2 number). For example, the upper leftmost bit is equal to 128 or 2 to the 7th power. Add the results of each dot in the row to arrive at the decimal value for that row. When you are finished calculating the decimal values of the redefined character, you should have eight numbers between 0 and 255.

For example, to define A to be the character shown above, type:

```
CHAR(1) = 255,193,32,16,16,32,193,255
```

The 1 in parentheses assigns the given character its location (its screen display code) in the RAM character set. There are up to 255 locations.

Example: Use a redefined character in a program as follows:

```
10 COPY UPPERCASE TO RAM
20 CHAR RAM
30 CHAR(1)=255,193,32,16,16,32,193,255
40 PRINT "A"
RUN
```

The new character should replace the A everywhere it appears.

See also

CHAR (<ascii>,n)
CHAR SAVE
CHAR LOAD

CHAR (<ascii>,n) = "<an 8-character string>"

Description: This is an alternative way to define your own character set. This variation uses a visual representation of the actual character to be redefined. Each CHAR (<ascii>,n) command sets the bits (on or off) within one row of the specified ascii character. For example, the following eight commands would define a complete character in place of the letter A.

```
CHAR (1,1)="xxxxxxxx"
CHAR (1,2)="xx      x"
CHAR (1,3)="  x     "
CHAR (1,4)="   x    "
CHAR (1,5)="   x    "
CHAR (1,6)="  x     "
CHAR (1,7)="xx      x"
CHAR (1,8)="xxxxxxxx"
```

The first number in parentheses after CHAR is the screen display code of the character you want to redefine (see pages 132-134 of your *Commodore 64 User's Guide).* The second number indicates the row of the character. The x represents a bit that is on, and a space represents a bit that is off. (Any character except for a space will register a bit that is on.)

Example:

```
 10 COPY UPPERCASE TO RAM
 20 CHAR RAM
 30 CHAR (1,1)="xxxxxxxx"
 40 CHAR (1,2)="xx      x"
 50 CHAR (1,3)="  x     "
 60 CHAR (1,4)="   x    "
 70 CHAR (1,5)="   x    "
 80 CHAR (1,6)="  x     "
 90 CHAR (1,7)="xx      x"
100 CHAR (1,8)="xxxxxxxx"
RUN
```

The new character should replace the letter A everywhere it appears.

See also

CHAR (<ascii>)
CHAR LOAD
CHAR SAVE

CHAR RAM

Description: Switches to a character set in RAM located at $3800 through $4000. If there is no character data located there, (that is, you haven't redefined any characters) garbage will appear as characters. Character sets can be loaded into this area of RAM with the CHAR LOAD, CHAR (<ascii>) or CHAR (<ascii>,n) commands. You can create your own character sets using character editor software, or you can use the sample character sets included on the Graphics BASIC disk. Return to the standard Commodore character set by using the CHAR ROM command. The character set in RAM will not be destroyed when you type CHAR ROM. To clear the RAM of the character set, type COPY UPPERCASE TO RAM.

Example:

```
CHAR LOAD "OLD.FONT" (loads a complete font from
    the disk)
CHAR RAM (displays characters from RAM)
```

Type: "We have fonts, too!"

The characters should appear in old English style letters. Typing CHAR ROM switches back to the standard ROM character set, although the old English font will remain intact in RAM.

See also
- CHAR (<ascii>)
- CHAR LOAD
- CHAR SAVE
- CHAR ROM
- COPY LOWERCASE TO RAM
- COPY UPPERCASE TO RAM
- CHAR SET MEMORY
- CHAR RESET MEMORY

TEXT COMMANDS

CHAR ROM

Description: Displays the standard ROM character set, that is, the character set which appears when you first turn on the Commodore. CHAR ROM does not destroy a character set in RAM. You can get the character set in RAM back with the CHAR RAM command.

Example:

```
CHAR LOAD "CURSIVE.FONT"
CHAR RAM
Type: KEY LIST
CHAR ROM
All the characters should change back to normal.
```

See also
CHAR LOAD
CHAR RAM

HESWARE'S GRAPHICS BASIC
FOR YOUR COMMODORE 64 COMPUTER
(COMPUTER FONT)

HESWARE'S GRAPHICS BASIC
FOR YE OLDE COMMODORE 64 COMPUTER
(OLD ENGLISH FONT)

Heswane's Graphics Basic
You can even sign your name!
(CURSIVE FONT)

CHAR SAVE "<file name>" [,<device>]

CHAR LOAD "<file name>" [,<device>]

Description: These two commands are used to save and retrieve RAM character sets. The default device number is 8. You can load and save the characters you create with the CHAR (<ascii>) and CHAR (<ascii>,n) commands, whether you created them using a program or input them directly without line numbers. To save the character(s), type CHAR SAVE followed by a file name in quotes. To load the character(s), type COPY UPPERCASE TO RAM, CHAR RAM, and CHAR LOAD "filename". CHAR LOAD loads the entire character set, even if you only redefined one character.

The Graphics BASIC disk contains three complete character sets: Old English, computer, and cursive lettering. When loading one of these character sets, use the file names:

OLD.FONT	for Old English
COMPUTER.FONT	for computer style lettering
CURSIVE.FONT	for cursive lettering

Use the CHAR RAM command to switch to the character set once you have loaded it.

Return to the standard Commodore character set by using the CHAR ROM command.

WARNING: The data for RAM character sets occupies memory inside the BASIC workspace. If your BASIC program is large, the program may be destroyed.

Example:

```
CHAR LOAD"COMPUTER.FONT"
CHAR RAM
Type: "THE BYTE STOPS HERE."
```

Effect: All characters on the screen will appear in the customized font.

See also
- CHAR RAM
- CHAR ROM
- CHAR (<ascii>)
- COPY UPPERCASE TO RAM
- COPY LOWERCASE TO RAM

CHAR SET MEMORY

CHAR RESET MEMORY

Description: Reserves the memory needed for user-defined characters. BASIC now assumes you have less memory to work with, thus the area is protected from any BASIC actions. When either of the commands are used, all variables are destroyed. If you want to protect a RAM character set from damage by BASIC variable storage, one of these two commands must be initiated to reconfigure the memory, even if you plan to load another character set for use. The memory will remain configured for a RAM character set until the RESET or CHAR RESET MEMORY command is used. Thus, if a RAM character set is being used and the RESET or CHAR RESET MEMORY command is initiated, the character set is no longer protected from BASIC variables or program storage.

In a program where you are using a redefined character set, use the CHAR SET MEMORY command before you define any variables or dimension any arrays. Use CHAR RESET MEMORY command when you are finished with the character set or near the end of the program.

Example:
(for illustration only)

```
10 CHAR SET MEMORY
20 CHAR LOAD"COMPUTER.FONT"
30 CHAR RAM
40 GPRINT
   .
   .
   .
900 CHAR RESET MEMORY
```

COPY TEXT TO HIRES

Description: Copies the characters appearing on the text screen to the high resolution screen. All previous images on the graphics screen will be written over.

COPY TEXT TO PRINTER

Description: Prints the characters appearing on the text screen onto a standard Commodore compatible printer at device 4. Graphics characters will only appear properly when the Commodore 1525 printer is used.

This command will print program listings or text in a 40-column format. To print your program listings in an 80-column format, use the following sequence of commands:

```
OPEN 4,4 : CMD4 : LIST
```

Example:

```
10 HIRES
20 LINE 0,0 TO 120,120
COPY TEXT TO PRINTER
```

COPY UPPERCASE TO RAM

COPY LOWERCASE TO RAM

Description: These two commands copy character definition data from ROM into the reserved 2K RAM character set. You can then take the character definition data and redefine it to create customized characters (see CHAR (<ascii>) and CHAR (<ascii>,n).

Note that only uppercase *or* lowercase may be stored in the 2K RAM area at one time. Also, these two commands erase all variables and reconfigure BASIC workspace. See the memory map for details.

This command must be used before typing in CHAR RAM if you are redefining characters with CHAR (<ascii>) or CHAR (<ascii>,n). You do not need to use these commands with CHAR LOAD.

The commands COPY UPPERCASE TO RAM and COPY LOWERCASE TO RAM replaces (i.e. destroys) a character set currently in RAM.

Example:

```
COPY UPPERCASE TO RAM

CHAR RAM
CHAR (62) = 60,66,165,129,165,153,66,60
```

Effect: Each time you press >, a smiling face should appear, although the computer still treats the character as >.

See also

- CHAR SET MEMORY
- CHAR RESET MEMORY
- CHAR RAM
- CHAR ROM
- CHAR (<ascii>)
- CHAR (<ascii>,n)

TEXT COMMANDS

PRINT AT <x>,<y> [,] [<data>]

Description: Prints the data you supply at the specified cursor position on the text screen. The <x> and <y> values are referenced from 1,1 at the upper left corner of the screen. (Screen locations are different for TEXT and HIRES screens.)

Examples: PRINT AT 10,5 A$

PRINT AT X*4,Y+5"ATTACK OF THE MUTANT CAMELS!"

PRINT AT A,B,C

See also
GPRINT

TEXT

Description: Causes your Commodore 64 to display normal characters on the screen, as opposed to HIRES or MULTI graphics. This is the mode that your computer is in when you turn it on.

Example: 10 TEXT

See also
HIRES
MULTI
TEXT FROM/TO

TEXT COMMANDS

TEXT [FROM <firstline>] [TO <lastline>]

Description: Splits the screen with a TEXT screen visible from <firstline> to <lastline>. The remainder of the screen will be in the previous graphics mode. If FROM <firstline> is omitted, text will appear from the top of the screen down TO <lastline>. Similarly, if TO <lastline> is omitted, text will appear FROM <firstline> down to the bottom of the screen.

Examples: (for illustration only)

```
10 TEXT FROM 10 TO 10
```

```
10 TEXT FROM 20
```

```
10 A=4 : B=7
20 TEXT FROM A TO B
```

```
10 TEXT TO 20
```

Errors: ILLEGAL QUANTITY ERROR

1. One of the two lines given is not in the range of 1 to 25.

Command Format:

```
TEXT
TEXT TO <line>
TEXT FROM <line>
TEXT FROM <line> TO <line>
```

Demo Program: SPLIT DEMO

SCROLL <direction> <#characters> [WINDOW <min x>, <min y>,<max x,>,<max y>]

and

ROLL <direction> <#characters> [WINDOW <min x>, <min y>,<max x>,<max y>]

Description: These two commands scroll or roll the specified portion of the screen in the given <direction>. The acceptable directions are RIGHT, LEFT, UP, and DOWN. (Only the first character is required.) The WINDOW values specify the cursor positions for the upper left and lower right corners of the scrolling window referenced from 1,1 at the upper left corner of the screen. If the WINDOW option is omitted, the previous window setting is used, and if none has been previously set, the WINDOW is defaulted to the entire screen.

Examples: (for illustration only)

```
SCROLL RIGHT WINDOW 4,4,10,10
SCROLL LEFT 4
ROLL UP A*2
ROLL DOWN 6 WINDOW 1,1,15,15
```

Demo Program: SCROLL DEMO

NOTES

APPENDIX A: ERROR MESSAGES

Error Number	Message
1	Too Many Files
2	File Open
3	File Not Open
4	File not found
5	Device Not Present
6	Not input file
7	Not output file
8	Missing file name
9	Illegal device number
10	Next without For
11	Syntax
12	Return without Gosub
13	Out of data
14	Illegal Quantity
15	Overflow
16	Out of memory
17	Undefined statement
18	Bad subscript
19	Redimensioned array
20	Division by zero
21	Illegal direct
22	Type mismatch
23	String too long
24	File data
25	Formula too complex
26	Can't continue
27	Undefined function
28	Verify
29	Load
30	Break

See pages 150-151 of your *Commodore 64 User's Guide* for more details on these error messages.

NOTE: When Graphics BASIC encounters an error in your program, the line containing the error will be listed. The location of the error will be indicated by a double angle bracket < > (in reversed video). In making the corrections, be sure to remove the double angle bracket symbol.

APPENDIX B: DEMO PROGRAMS

Demo Programs on Diskette

ADSR DEMO
DIAMOND DEMO
ROTATE SQUARES
SHUTTLE DEMO
DRIVING DEMO
SPLIT DEMO
WINDOW DEMO
8BALLS DEMO
ORIGIN DEMO
HATMAN DEMO
MULTISTAR DEMO
DESIGN DEMO
MULTIHATMAN DEMO
MULTIBAR DEMO
ELLIPSE DEMO
JOYSTICK DEMO
AIRPLANE DEMO
GPRINT DEMO
3-VOICE DEMO
BALL&BOX DEMO
MULTICHANGE DEMO
3-PART SONG DEMO
BOUNCE BALL DEMO
PIE CHART DEMO
CIRCLE FILL DEMO
MODERN ART DEMO
SATURN DEMO
HATMAN RACE DEMO
SCROLL DEMO
SPACEMAN DEMO
WIND/CIRC DEMO
ANIMATE DEMO
PRIORITY DEMO
SETORIGIN DEMO
COLORWHEEL DEMO
PENCIL DEMO
HES BOUNCE DEMO

Graphics BASIC must be loaded into the computer before you can load a demo program. To load a demo, type:

LOAD"<demoname>",8

To run the demo, type:

RUN

Character Sets

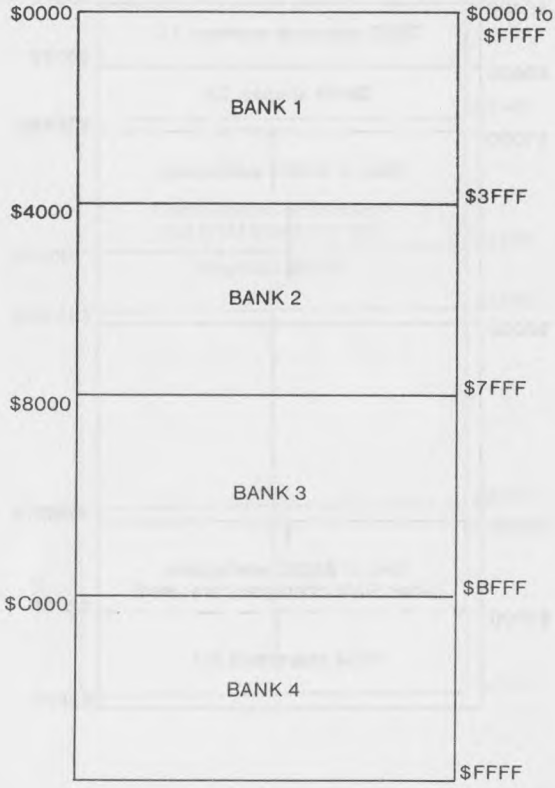
CURSIVE.FONT
OLD.FONT
COMPUTER.FONT

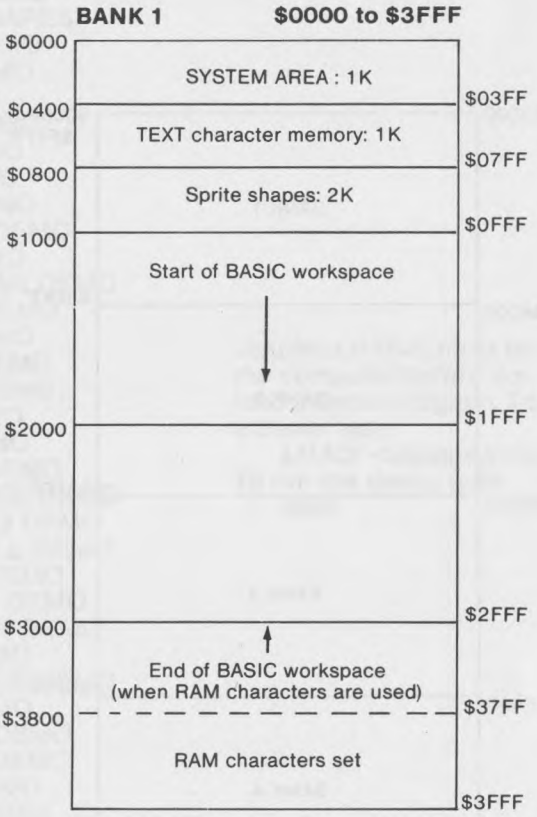
To load a character set, type

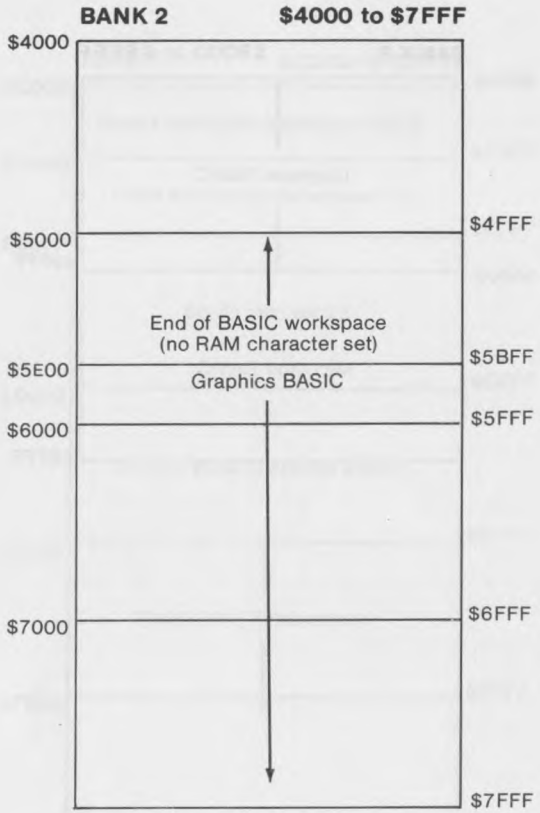
CHAR LOAD"<fontname>"
CHAR RAM

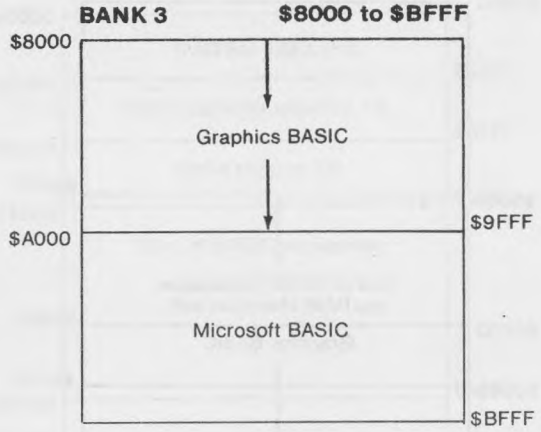
APPENDIX C: MEMORY MAP

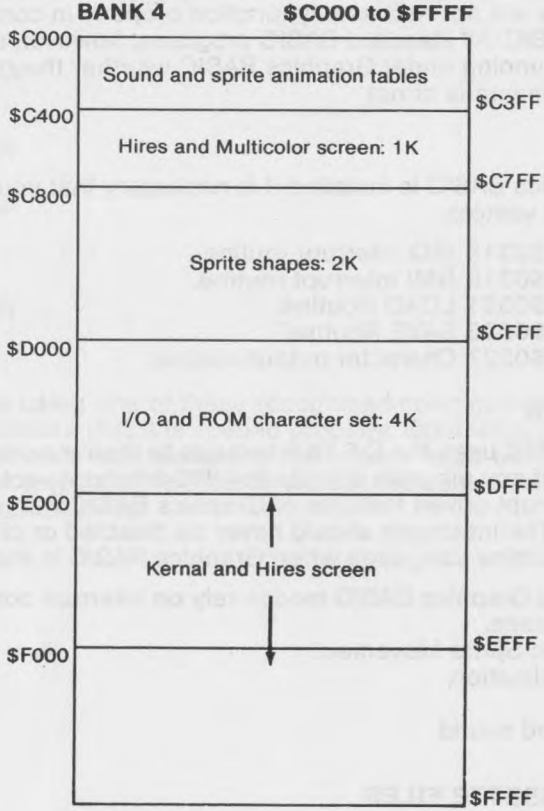
Graphics BASIC does some reconfiguration of the Commodore 64's memory. Following is a memory map of the Commodore 64 with Graphics BASIC installed.




BANK 1
$0000 to $3FFF
$0000
SYSTEM AREA : 1K
$03FF
$0400
TEXT character memory: 1K
$07FF
$0800
Sprite shapes: 2K
$0FFF
$1000
Start of BASIC workspace
$1FFF
$2000
$2FFF
$3000
End of BASIC workspace
(when RAM characters are used)
$37FF
$3800
RAM characters set
$3FFF


BANK 2
$4000 to $7FFF
$4000
$4FFF
$5000
End of BASIC workspace
(no RAM character set)
$5BFF
$5E00
Graphics BASIC
$5FFF
$6000
$6FFF
$7000
$7FFF


BANK 3
$8000 to $BFFF
$8000
Graphics BASIC
$9FFF
$A000
Microsoft BASIC
$BFFF



ZERO PAGE

Zero page remains the same while Graphics BASIC is installed with the exception of four memory locations:

$00FB – $00FE Used by Graphics BASIC

APPENDIX D: CAUTIONS IN GRAPHICS BASIC

GRAPHICS BASIC AND OTHER PROGRAMS

Graphics BASIC is an addition to the existing BASIC in your Commodore 64. If you have any programs that patch into the present BASIC, use memory locations used by Graphics BASIC or that do not check the existing memory configurations before POKEing data into memory, they will not necessarily function properly in conjunction with Graphics BASIC. All standard BASIC programs, however, should have no problem running under Graphics BASIC whether they use the extended commands or not.

VECTORS

When Graphics BASIC is installed it is necessary that you not change the following vectors:

$0314 - $0315 IRQ interrupt routine.
$0318 - $0319 NMI interrupt routine.
$0330 - $0331 LOAD Routine.
$0332 - $0333 SAVE Routine.
$0326 - $0327 Character output routine.

INTERRUPTS

Graphics BASIC uses the C-64's interrupts to their maximum capabilities. If any program adjusts the IRQ interrupt vector for its use, then all interrupt-driven features of Graphics BASIC will be inoperative. The interrupts should never be disabled or changed from BASIC or Machine Language when Graphics BASIC is installed.

The following Graphics BASIC modes rely on interrupt control:
Split Screens,
Automatic Sprite Movement,
Sprite Animation,
and
Automated sound

DISK or CASSETTE FILES

Disk or Cassette Files (used with the OPEN command) should never be used in Graphics BASIC when any interrupt-driven mode is engaged. To use disk or cassette files, first execute the RESET command. This will disable all interrupt-driven functions. If you forget, the computer will lock up. To free the computer, hit the RUN-STOP and RESTORE keys at the same time. This will cause Graphics BASIC to RESET. Your variables and program, however, will not be lost.

All LOAD and SAVE commands will operate properly with sprite animation or a split screen engaged. They simply halt all interrupt-driven functions until after the LOAD or SAVE is completed.

APPENDIX E: GRAPHICS BASIC COLORS

In Graphics BASIC you can use the actual color names in place of the color numbers. The following is a list of all the colors Graphics BASIC understands.

0 - BLACK
1 - WHITE
2 - RED
3 - CYAN
4 - PURPLE
5 - GREEN
6 - BLUE
7 - YELLOW
8 - PEACH
9 - BROWN
10 - PINK
11 - GRAY1
12 - GRAY2
13 - LGREEN
14 - SKY
15 - GRAY3

NOTE: When using one of these recognized color names, care must be taken to ensure that it is spelled properly. Otherwise, it will be interpreted as a variable name and its present value will be used as a color number. (e.g. COLOR GREAN will set the color to white when the variable GREAN=1.)

APPENDIX F: NOTE TABLE

Note	Octave	Tone number	Note number	
A	4	7382	0	440 Hz
Bb	4	7821	1	
B	4	8286	2	
C	4	8779	3	
C#	4	9301	4	
D	4	9854	5	
D#	4	10440	6	
E	4	11061	7	
F	4	11718	8	
F#	4	12415	9	
G	4	13253	10	
G#	4	13935	11	
A	5	14764	0	880 Hz
Bb	5	15642	1	
B	5	16572	2	
C	5	17557	3	
C#	5	18601	4	
D	5	19708	5	
D#	5	20879	6	
E	5	22121	7	
F	5	23436	8	
F#	5	24830	9	
G	5	26306	10	
G#	5	29871	11	
A	6	29528	0	1760 Hz

To calculate a tone number not in the list above, use the following formula:

<Tone number>=(1.059463157)↑(<note number>)*59059*2↑(<octave>−7)

Where <note number> is the value of the note you are looking for (for example, the note number for the note D in any octave is 5) and where <octave> is a number between 0 to 6 representing 7 full octaves available on the Commodore 64.

APPENDIX G: DISK INFORMATION

Formatting a Disk

Before you can use a disk for program storage, you need to set up "magnetic signposts" for the disk drive to read. You give the disk a name and a two-character ID number. This process is called formatting the disk.

CAUTION: Formatting a disk erases any files currently on that disk.

To format a disk, remove the Graphics BASIC disk and insert a blank disk. Use the DISK command in Graphics BASIC as follows: (Device number 8 is assumed.)

DISK "N0:<name>,ID"

For example:

DISK"N0:GRBASICPROGS,64" <RETURN>

The red light on the disk drive will come on for about a minute; then the red light will turn off, and the disk will be formatted.

Disk Care

To ensure long life and reliable operation from your disks, follow these guidelines:

- Keep disks away from magnetic objects—information on the disks may be erased or become hopelessly deranged. Magnetic objects include: your TV or monitor, disk drive, power supply, printer, telephone, even paper clips stored in a magnetized holder.
- Do not touch the exposed parts (shiny parts that show through the black disk liner) or the disk with anything. This includes fingerprints, coffee spills, cigarette ashes, paperclips, and dust.
- Store disks in their protective envelopes and in a vertical position when not in use. Keep the disk away from any direct heat source—sunlight, heaters, sleeping cats.

INDEX

Graphics BASIC commands are listed in capital letters. Page numbers in boldface indicate the page in the command reference section where the command is explained in detail.

ACKNOWLEDGEMENTS

The programmers would like to thank Charles and Barbara Landis for all their support and understanding in this project.

Special thanks to Jay Stevens and Tom Wahl for their suggestions during the development of this product, their numerous and creative demo programs, and their support in the preparation of this manual.

This manual has been thoroughly tested and to the best of our knowledge, the text is free of typographical and technical errors. However, in spite of our thorough testing procedures, a few errors may have crept in. If you catch any errors, we would appreciate hearing about them, as well as any other comments or suggestions on this manual.

Barbara Harvie
Product Documentation Manager
Human Engineered Software
150 North Hill Dr.
Brisbane, CA 94005