SUPER EXPANDER 64

commodore
COMPUTER

COPYRIGHT

DISCLAIMER

COMMODORE ELECTRONICS LTD. ("COMMODORE") MAKES NO WARRANTIES, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THE PROGRAM DESCRIBED HEREIN, ITS QUALITY, PERFORMANCE, MERCHANTABILITY, OR FITNESS FOR ANY PARTICULAR PURPOSE. THIS PROGRAM IS SOLD "AS IS". THE ENTIRE RISK AS TO ITS QUALITY AND PERFORMANCE IS WITH THE BUYER. SHOULD THE PROGRAM PROVE DEFECTIVE FOLLOWING ITS PURCHASE, THE BUYER (AND NOT THE CREATOR OF THE PROGRAM, COMMODORE, THEIR DISTRIBUTORS OR THEIR RETAILERS) ASSUMES THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION AND ANY INCIDENTAL OR CONSEQUENTIAL DAMAGES. IN NO EVENT WILL COMMODORE BE LIABLE FOR DIRECT, INDIRECT, INCIDENTAL OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT IN THE PROGRAM EVEN IF IT HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME LAWS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF IMPLIED WARRANTIES OR LIABILITIES FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY.

Your new SUPER EXPANDER 64 cartridge is a powerful extension of the BASIC language. It gives you the commands needed to easily access and implement Commodore's graphics, music, and sound capabilities. You will be amazed at how quickly and easily you can do the following:

- Plot points and lines
- Draw arcs, circles, ellipses, rectangles, triangles, octagons
- Paint shapes with your choice of colors
- Read game paddle, joystick, and light pen locations
- Display text or split screens to display both text and graphics
- Program your Commodore 64 function keys.

We sincerely hope you enjoy using SUPER EXPANDER 64. Here are several other Commodore software packages which you should know about:

The Commodore 64 Macro Assembler Development System

This package is designed for experienced Assembly language programmers. Everything you need to create, assemble, load, and execute 6500 series Assembly language code is included.

Disk Bonus Pack
Cassette Bonus Pack

As an introduction to your Commodore 64, both of these packages feature programming aids, music and video demonstration programs, and several educational and personal programs.

Screen Editor

The Screen Editor helps you design software by letting you create and edit your own screens. This programming tool is for users with some computer experience.

The Word Machine and The Name Machine

This is an easy-to-learn and easy-to-use wordprocessing package. Perfect for letters, address lists, memos, and notes, these programs let you overtype, insert, and delete text; personalize form letters; and print in draft, formal, or informal formats.

The Manager

This is a general data base for handling your files. THE MANAGER interfaces with wordprocessing, accumulates totals on the screen, and creates subfiles. THE MANAGER sorts from any field and features powerful report generating capabilities.

Easymail 64

With EASYMAIL 64, your address files are simple to manage. You can now easily keep track of names/addresses and print as many mailing labels as you desire. EASYMAIL 64 has the features you need for preparing special mailings (i.e., capabilities for searching for a specific category). Here are some of the EASYMAIL 64's many features: entry, change or deletion of labels by name or label number; printing on one or two abreast address labels; a complete printout of all of your data; and a HELP function.

Easyscript 64

EASYSCRIPT 64 is a powerful word processor with table producing capabilities. This package features comprehensive printer controls and easy document handling.

Preface

The SUPER EXPANDER 64 is a powerful extension of the BASIC language in your Commodore 64 computer. Previously, you had to Peek or Poke specific memory locations to access your computer's graphics and sound features. Now, the SUPER EXPANDER 64 provides new BASIC commands so you can easily access the Commodore 64's many features. Simply "plug" the SUPER EXPANDER 64 cartridge into your computer and turn on the power. With SUPER EXPANDER 64, you have the power to:

- Build high resolution graphic displays
- Create and animate sprites
- Create your own shapes or figures
- Draw points, lines, arcs, circles, and ellipses
- Draw polygons such as rectangles, triangles, octagons
- Combine text and high resolution graphics
- Fill shapes with your choice of colors
- Read game paddle, joystick, and light pen positions
- Create music and game sounds
- Define programmable function keys

Most BASIC programming tasks that you can do with the SUPER EXPANDER 64, can also be done in BASIC alone. However, when using the SUPER EXPANDER 64, your programs will generally be more compact, run much faster, and be easier to develop. Also, the SUPER EXPANDER 64 lets you program many tasks that simply cannot be done using BASIC alone.

This manual is intended for readers **who already know** some BASIC programming and **are familiar with** the many features of the Commodore 64. The SUPER EXPANDER 64 commands and functions are explained and accompanied by specific program examples for you to try. If anything in this manual is unclear, you should refer to the **Commodore 64 User's Guide** and the **Commodore 64 PROGRAMMER'S REFERENCE GUIDE** for additional information.

We are sure you will enjoy using Commodore's SUPER EXPANDER 64 cartridge, with its many graphics and sound features. Even readers with little "artistic" or "music" background will be able to easily incorporate bright, colorful, animated graphics and interesting sounds in their business, education, and game programs.

User Conventions

Here is a brief discussion of certain keys and symbols, and their respective use in the SUPER EXPANDER 64 manual. This will also help you to interpret the syntax of the commands and functions, including their optional features.

RETURN To continue on with a program after a line of input, press the RETURN key.

SHIFT To input the upper case convention of a letter, press and hold down the SHIFT key in conjunction with the desired key. Both keys should then be released at the same time.

C= To represent the "Commodore" key found on the lower left hand corner of your Commodore 64 keyboard (beside the SHIFT key)

< > Angled brackets indicate that the enclosed parameter is required information. However, the parameter itself may be of a variable nature.

[ ] Square brackets indicate that the enclosed parameter is optional and may be omitted from the command syntax.

.... Several consecutive periods, "ellipses", specify to repeat the preceding optional parameter.

A **command** is a keyword that may stand alone or be followed by one or more **parameters**. Each of these parameters is separated by a punctuation mark, such as a comma (,), semi-colon (;), number sign (#) or space.

A **function** is a keyword which is immediately followed by parentheses that enclose one or two **arguments.** Refer to your **Commodore 64 Programmer's Reference Guide** for a further explanation of commands and functions.

Command Summary

To enter GRAPHIC/TEXT mode:

```
GRAPHIC <mode> [,clear]
```

To enter Sprite Designer mode:

```
SPRDEF
```

Graphic Shape Generation and Color Selection:

Pixel Graphics and Color Selection

```
BOX [source] <,X1,Y1> [,[X2,Y2] [,[angle] [,fill] ] ]
CHAR [source] , <column,row> , <string> [,reverse]
CIRCLE [source] ,[X1,Y1] <,X-rad> [,[Y-rad] [,[start]
      [,[end] [,[angle] [,[inc] ] ] ] ] ]
COLOR [bgnd] [,[fgnd] [,[mcr1] [,[mcr2] [,ext] ] ] ]
DRAW [source] [,X1,Y1] [TO X2,Y2]...
GSHAPE <stringname> [,[X1,Y1] [,method] ]
LOCATE <X,Y>
PAINT [source] [,[X1,Y1] [,halt] ]
SCALE <n>
SCNCLR
SSHAPE <stringname>, <X1,Y1> [,X2,Y2]
```

Sprite Graphics and Programmable Collision Interrupts

```
COLINT <event> [,line-num]
MOVSPR <number> <,X1,Y1>
SPRCOL [smcr-1] [,smcr-2]
SPRITE <number> [,[on/off] [,[fgnd] [,[priority]
   [,[x-exp] [,y-exp] [,mode] ] ] ] ] ]
SPRSAV <origin>, <destination>
```

Built-in Functions and User Interface:

Graphic Functions

RCLR (<area>)

RDOT (<data>)

RGR (0)

Sprite Functions

RBUMP (<event>)

RSPCOL (<register>)

RSPPOS (<sprite> , <data>)

RSPR (<sprite> , <field>)

Game Port I/O and Programmable Function Keys

KEY [<keynum, string-expr>]
RJOY (<joystick>)

RPEN (<data>)

RPOT (<paddle>)

Creating Music and Sound with the SUPER EXPANDER 64:

Setting Up for Music and Sound

FILTER [freq], [,[low] [,[band] [,[high] [res]]]]
TEMPO < speed >
TUNE < env >, [,[atk] [,[dec] [,[sus] [,[rel] [,[form] [,[width]]]]]]

Music Elements

Element	Description
A,B,C,D,E,F,G,	Notes
#	Sharp
$	Flat
.	Dotted note
W	Whole note
H	Half note
Q	Quarter note
I	Eighth note
S	Sixteenth note
R	Rest

Playing Your Music and Sounds

SID Control	Description
< CTRL-F >	Enable/disable music playback (CHR$(6))
O	Octave
T	TUNE envelope
U	Volume
V	Voice
X	Filter

TABLE OF CONTENTS

1

Getting Started

The SUPER EXPANDER 64 is a cartridge based program that adds 21 new commands and 11 new built-in functions to the BASIC language in your Commodore 64. Follow these easy steps to start the SUPER EXPANDER 64 program:

- Turn OFF your computer (ALWAYS turn the computer power OFF when inserting or removing a cartridge.)
- Position the SUPER EXPANDER 64 cartridge so that the label is facing up and the open end is towards you.

INSERT PHOTO A

- Insert the SUPER EXPANDER 64 cartridge into the cartridge expansion slot. This is located just to the left of and behind the power light on your computer.
- Turn ON your computer.

You can now begin entering SUPER EXPANDER 64 commands directly from the keyboard (DIRECT mode) or you can incorporate these additional commands into your BASIC programs (PROGRAM mode). In DIRECT mode, simply enter the commands and press RETURN for immediate execution. In PROGRAM mode, the additional SUPER EXPANDER 64 commands are entered in your BASIC programs using standard line numbers. Type RUN for the program to execute.

Important Notes:

- Anytime your commands generate an unreadable display, such as white lettering on a white background, press the RUN/STOP and RESTORE keys simultaneously. This returns the computer to its original state, i.e., all sprites are turned off, all sound is stopped, and you are returned to standard text mode. The program in memory will remain intact.
- Programs created using SUPER EXPANDER 64 commands require that the cartridge is plugged in each time the program is run. Otherwise, the SUPER EXPANDER 64 commands will generate errors.
- In normal BASIC programs, keywords are reduced to single character "tokens" (or internal symbols) to conserve memory. The SUPER EXPANDER 64 keywords are reduced to two character tokens. Thus, when SUPER EXPANDER 64 commands are used in IF/THEN statements, a colon (:) MUST appear between the THEN and the SUPER EXPANDER 64 keyword.
- If any optional parameter is omitted on any SUPER EXPANDER 64 command, the parameter will default to the previously specified value. This is true even if the parameter was set in a previous program. Thus, we recommend that you do not omit optional parameters when using any SUPER EXPANDER 64 command for the first time in a program. Also, when you do omit a parameter in a SUPER EXPANDER 64 command, you must use a comma as a "place holder" if any parameters after it are explicitly stated.

2

TEXT and GRAPHICS

- Selecting Modes
- Selecting Colors
- Plotting Points and Lines
- Drawing Boxes, Circles, and Polygons
- Mixing Text with Graphics
- Saving and Replacing Shapes

TEXT and GRAPHICS

In this chapter, we will describe how to select the SUPER EXPANDER 64's graphic modes and how to choose colors for drawing. Ways to plot points and lines, draw boxes, circles and regular polygons are also discussed. We will show you how to mix text with your graphics displays and even define and move shapes on the screen.

Selecting Modes

The SUPER EXPANDER 64 program supports four graphic modes: Text, Multi-Color, High Resolution, and Split Screen. You can select the mode you want by using the GRAPHIC command as shown below:

GRAPHIC < mode > [,clear]

The required < mode > parameter is a number from 0 through 3 and is used for selecting one of the four graphic modes. When the optional [,clear] parameter is non-zero, the screen will be cleared to the background color (see the section on Selecting Colors) after the new mode is selected. The following list shows the four graphic modes and their corresponding < mode > values:

0	=	Standard Text Mode
1	=	Multi-Color (Bit Map) Mode
2	=	High Resolution (Bit Map) Mode
3	=	Split Screen Mode (Mixed High Res/Text)

Here are some examples of the GRAPHIC command:

GRAPHIC 1 Selects Multi-color Mode.

GRAPHIC 2,1 Selects High Resolution Mode and clears the Bit-Map Screen.

100 M = 3: C = 1: GRAPHIC M,C

110 REM Line 100 selects Split-Screen Mode and clears the screen

Another way to clear the screen, without using the GRAPHIC command, is to use the SCNCLR command. This command has the same effect as a non-zero [,clear] parameter with the GRAPHIC command. The SCNCLR command is used by itself, without parameters:

100 SCNCLR Clears the Screen in any Mode.

The RGR (0) function tells you which graphic mode you last selected by returning a number from 0 through 3. RGR (0) is used exactly as shown here. The number that is returned corresponds to the < mode > value used in the GRAPHIC command. Here are some examples:

```
20 GRAPHIC 3: PRINT RGR (0): END
30 REM LINE 20 PRINTS A "3" WHEN RUN

100 IF RGR (0) > 0 THEN: GRAPHIC 0,1
110 PRINT "SWITCH TO TEXT MODE"
```

The following is a brief description of each of the GRAPHIC modes and their different characteristics (refer to the **Commodore 64 PROGRAMMER'S REFERENCE GUIDE** for a more detailed discussion):

Standard Text Mode

Your computer is in Standard Text mode when you turn the power ON. In text mode, a maximum of 1000 characters can be displayed (25 lines of 40 characters each). Each character consists of an 8 by 8 dot region and may be any one of the 16 available colors.

High Resolution Mode

High Resolution mode is for creating graphics with the highest degree of precision and detail. In this mode, the screen is treated as a grid with 320 horizontal dots by 200 vertical dots. Therefore, your designs appear very sharp and crisp. Each 8 by 8 dot region of the screen (these regions correspond to text character locations) may have your choice of any two colors (Foreground and Background).

Multi-Color Mode

With Multi-Color mode, horizontal resolution is sacrificed for the ability to use more colors. The screen has 160 horizontal dots by 200 vertical dots. Each Multi-Color horizontal dot is twice as wide as each High Resolution horizontal dot. Multi-Color mode lets you display up to four different colors in each 8 by 8 dot region. The four colors you can set are Background, Foreground, Multi-Color 1 and Multi-Color 2.

Split Screen Mode

The Split Screen mode mixes both the High Resolution and Standard Text modes. In Split Screen mode, the top part of the screen is a 320 dot horizontal by 160 dot vertical High Resolution bit map. The bottom part of the screen is a "window" where you can display five (5) lines of Standard Text.

See the photographs on the back of this manual and Appendix III SAMPLE PROGRAMS for more examples of the GRAPHIC modes.

Selecting Colors

On your Commodore 64, you can independently set the Background, Border, and Foreground areas to any of the 16 colors. You may also set two multi-color "registers", though you will not see the effects of this unless you are using Multi-Color GRAPHIC mode. With the SUPER EXPANDER 64, you can use the COLOR command to select your choice of colors for these areas or "color sources". The syntax for the COLOR command is as follows:

```
COLOR [bgnd] [,[fgnd] [,[mcr1] [,[mcr2] [,ext] ] ] ]
```

The [bgnd] parameter specifies the Background color. The [fgnd] parameter specifies the Foreground color which is the color of the letters in text mode and most shapes drawn in High Resolution mode. The [mcr1] and [mcr2] parameters are the Multi-Color registers which are only viewable in Multi-Color mode. The [,ext] parameter specifies the Exterior Border color.

You can use any of the sixteen available colors on your Commodore 64 for any of the five parameters of the COLOR command. We will later refer to each of these parameters as "color sources". To select a color source for an area, simply specify its "color code" number in the appropriate position of the COLOR command. Here is a list of the colors and their corresponding color codes:

COLOR	CODE	COLOR	CODE
Black	0	Orange	8
White	1	Brown	9
Red	2	Light Red	10
Cyan	3	Dark Grey	11
Purple	4	Medium Grey	12
Green	5	Light Green	13
Blue	6	Light Blue	14
Yellow	7	Light Grey	15

Remember to use a comma as a position holder for fields which are omitted (these remain the same color as previously set). For example, to specify a black background, white lettering, and a light grey exterior border, use a COLOR command as follows:

COLOR 0, 1, , ,15 — Sets Background color to black, Foreground to white, and Border to light grey. This does not change the color of Multi-Color1 or Multi-Color2.

COLOR , , , , 3 — Sets Border (only) to Cyan. Other areas remain as previously set.

Notice that changing the Background color has no immediate effect unless you are in TEXT mode. In any GRAPHIC mode, the newly selected Background color is seen once you begin drawing or after a SCNCLR command.

See the photographs on the back of this manual and Appendix III SAMPLE PROGRAMS for more examples of the COLOR command.

To check what colors you last set, use the RCLR function. The syntax for this function is as follows:

RCLR (< area >)

The < area > argument is a number 0 through 4 that corresponds to one of the five areas of the screen whose colors are set by the COLOR command. For example, if < area > is 0, then RCLR will return the color code (a number 0 through 15) that was last specified for the background area of the screen.

Here is how to examine each of the screen color registers:

RCLR (0)	Returns Background color code
RCLR (1)	Returns Foreground color code
RCLR (2)	Returns Multi-Color1 color code
RCLR (3)	Returns Multi-Color2 color code
RCLR (4)	Returns Exterior Border color code

Here is a sample program you can RUN to see how RCLR works:

```
10  COLOR 0,7,2,3,4
20  FOR C = 0 TO 4
30  PRINT RCLR (C)
40  NEXT C
```

Plotting Points and Lines

Before you are ready to start drawing, there is an important concept that you should understand: the Pixel Cursor (PC). A pixel corresponds to a dot location on the screen. The pixel cursor is similar to the flashing cursor you see in Text mode which indicates where the next character will be displayed.

Although invisible, the pixel cursor indicates where the next dot will be placed in a High Resolution or Multi-Color bit map screen. In SUPER EXPANDER 64 commands where optional coordinates are omitted, the pixel cursor is also used as the default coordinate. Specific examples of how the PC is used as a default coordinate will be included in the following command discussions.

The LOCATE command lets you place the PC anywhere on the screen. The results of the LOCATE command will not be seen until you actually draw something.

The syntax for the LOCATE command is:

```
    LOCATE < X,Y >
```

The "X" part of the required < X,Y > parameter represents the horizontal distance across the screen in dot positions. When "X" equals 0, the PC is at the left edge of the screen. The "Y" part of the < X,Y > parameter represents the vertical distance down the screen in dot positions. When "Y" equals 0, the PC is at the top edge of the screen. In SUPER EXPANDER 64 commands, if an optional X-Y coordinate is omitted, the current PC location is used as a default location. As each of the SUPER EXPANDER 64 commands is discussed, we will specify which X-Y coordinates are optional.

The X and Y Coordinates can be specified as either absolute values or as an offset from the current PC location. By preceding the X or Y value with either a plus sign or minus sign (+ , –), the PC is moved in either a positive or negative direction relative to its current location. A plus sign before the X value moves the PC to the right and a minus sign moves the PC to the left. Likewise, a plus sign before the Y value moves the PC downward from its current location and a minus sign moves the PC upward. Anywhere that you explicitly state an X or Y coordinate in SUPER EXPANDER 64 commands, you can use either absolute values or a relative offset.

Here are some examples of the LOCATE command:

```
10   LOCATE 0,0   Places PC at the upper left corner of
                  the screen
50   LOCATE 160,100   Places PC at the exact center of
                       the High Resolution screen
80   LOCATE  – 40, + 20   Moves PC to the left 40 dots
                          and down by 20 dots
```

In these examples, the new PC locations were given as either an absolute X,Y coordinate or a relative offset. In this and other SUPER EXPANDER 64 commands, an alternate way to express the new location can be used. In the LOCATE command, as with several other commands that draw, you can specify a distance and angle relative to the current PC location by using a semi-colon in the place of the comma. For example:

```
LOCATE 50;45
```

The PC is moved from its current location by a distance of 50 dot positions at an angle of 45 degrees.

Other SUPER EXPANDER 64 commands such as DRAW, BOX, CIRCLE, etc., can also change the PC location. You can find out where the PC is at any time by using the RDOT function. RDOT will give you the X or Y coordinate, or the color source for the pixel at the current PC location. Here is the syntax for the RDOT command:

```
RDOT (<data>)
```

When the < data > argument is 0, the X-coordinate of the PC location is returned; when the < data > argument has a value of 1, the Y-coordinate of the PC location is returned; and when the <data > argument has a value of 2, a number from 0 through 3 is returned. This number represents the color source (as described in the COLOR command) for the dot at the current PC location and is shown in the following table:

0 = Background
1 = Foreground
2 = Multi-Color1
3 = Multi-Color2

The value of the color sources returned by RDOT (2) corresponds to the positions of the parameter fields in the COLOR command. Here are some examples using the RDOT function:

```
10 X = RDOT (0)   The horizontal position of the PC is put into
                  variable 'X'.
20 B = 1: PRINT RDOT (B)   Prints the vertical position of
                           the PC.
30 LOCATE X,Y: CT = RDOT (2)   Identifies the color source
                               for the dot at location X,Y.
```

You are now ready to use the DRAW command to draw simple, straight lines on the screen. The syntax for the DRAW command is as follows:

```
DRAW [source] [,X1,Y1] [TO X2,Y2] . . .
```

In the DRAW command, [source] is the color source for the line to be drawn and can have values from 0 through 3, corresponding to the preceding RDOT table. If [source] is omitted, the Foreground color source is used for drawing. Drawing begins at the [,X1,Y1] position. If this parameter is omitted, drawing begins at the current PC location. The line is drawn to [X2,Y2] if given; otherwise, a single dot is drawn at [,X1,Y1]. Drawing a line changes the PC location to the last point plotted on the line.

You can use absolute coordinate values or relative offsets for either set of X-Y locations. Also, you can use an alternate form of the DRAW command which lets you draw a line from the current PC at a given angle and distance (simply use a semi-colon to replace the comma). For example:

```
DRAW [source] [, <dist >; < angle> ] [TO < dist >;<angle> ]...
```

The line will be drawn for the given distance (in dot positions), at the specified angle (in degrees, where 0 degrees is straight up) from the current PC location. Both the distance and angle parameters must be specified.

Here are some sample DRAW commands (be sure to specify GRAPHIC 2,1, before trying these examples):

```
DRAW  1,100,50     Draws a dot in Foreground color.

DRAW  TO 200, 100     The PC is used to start the line at
                        100,150.

DRAW  ,10,10 TO 100,60

DRAW  3 TO  – 20, + 40    Uses relative offset for X2,Y2.

DRAW  2 TO 25; 30    The line is 25 dots long, at 30
                            degrees from the current PC.

DRAW  ,10,10 TO 10,60 TO 100,60 TO 10,10

DRAW  0,100,50     Erases the dot drawn by the first
                   example.
```

See the photographs on the back of this manual and Appendix III SAMPLE PROGRAMS for more examples of the DRAW command.

Drawing Boxes, Circles, and Polygons

The BOX command lets you draw a rectangle anywhere on the screen. The rectangle can be any size or proportion. Special features of the BOX command let you rotate the rectangle to any angle or automatically fill the box with color. The syntax for the BOX command is:

```
BOX [source] < ,X1,Y1 > [, [X2,Y2] [, [angle] [,fill] ] ]
```

In this syntax example, any of the parameters [X2,Y2] or [angle] or [,fill] may be omitted. However, you must be sure to use commas as place holders if any parameters follow the ones you leave out.

The [source] is the color source for the rectangle. If [source] is omitted, the Foreground color source is used. The BOX is drawn starting from [X2,Y2] if given (otherwise, from the current PC location), and drawn to the required < ,X1,Y1 > location. The final position of the PC will be at [X2,Y2] if the [angle] parameter is omitted or zero. The [angle] and [,fill] parameters are both optional. The [angle] parameter specifies **clockwise degrees of rotation** about the center of the rectangle. If the [,fill] parameter is non-zero, the box will be filled with the same color as the outline of the box (source color).

Here are some samples of the BOX command (be sure to specify GRAPHIC 2,1 before trying these examples):

BOX 1,10,10,60,60	Draws a square.
BOX 1, + 80, + 40	This box uses relative offset from current PC value.
BOX ,10,10,60,60,45,1	Rotates the square and fills it with color.
BOX 1,0,0,319,199	This draws a box around the Hi-Res screen, just inside the border.
BOX 1,30;45,,,1	Distance and angle from PC locates the opposite corner.

As you can see in the last example, BOX is another command you can alter to specify a distance and angle by simply using a semi-colon in place of a comma. The current PC location is used as the starting point to draw a rectangle. Then, the diagonally opposite corner is located by moving the PC for a distance of 30 at an angle of 45 degrees. This point is then used as the < X1,Y1 > coordinate.

With the SUPER EXPANDER 64 CIRCLE command, you can draw arcs, circles, ellipses, and even triangles and other polygons. The syntax for the CIRCLE command is as follows:

```
CIRCLE [source] ,[X1,Y1] < ,X-rad > [,[Y-rad] [,[start] [,[end]
[,[angle] [,[inc] ] ] ] ] ]
```

The [source] is the color source for the circle. The origin (or center) of the circle is located at [X1,Y1] (if omitted, the current PC is used). As with preceding commands, you can use absolute values, relative offsets, or distance and angle for the [X1,Y1] parameter in the CIRCLE command. The < ,X-rad > parameter is required and specifies the horizontal radius of the circle in dot positions. The optional [Y-rad] parameter specifies the vertical radius of the circle in dot positions (if omitted, the < X-rad > value is used).

The optional [start] and [end] parameters define the starting and ending points of an arc on the circumference of the circle. The default values for [start] and [end] are 0 and 360 respectively. These points are given in degrees (where 0 is straight up, 90 is right, 180 is down, etc.). The [angle] parameter specifies the clockwise degrees of rotation about the center of the circle. The default value for [angle] is 0 (i.e., no rotation).

The final PC location will be on the circumference of the circle at the [end] arc angle. Because of the difference between the number of horizontal and vertical dots on the screen, setting the X radius equal to the Y radius will draw an ellipse, rather than a circle. To draw a circle, the scaling of the radius values should be near the ratio of horizontal to vertical dots on the screen. This ratio will depend on the graphic mode you are using. More information will be presented on X-Y coordinate ratios in our discussion of the SCALE command.

The SUPER EXPANDER 64 actually draws circles by plotting a series of straight lines. This is done by calculating the next point on the circumference of the circle and then drawing a straight line from the last plotted point. The [,inc] parameter, increment value, specifies how many degrees around the circle (clockwise) the PC is moved before drawing the next line. The default value for the increment is 2 degrees. As the increment value becomes larger, the outline of the circle becomes coarser, until a point is reached where the circle appears as a regular polygon.

Here is a set of examples which illustrate the CIRCLE command (be sure to specify GRAPHIC 2,1 before trying these samples):

```
CIRCLE  1,160,100,100,75       Circle
CIRCLE  1,160,100,65,10        Ellipse
CIRCLE  1,160,100,65,10, , ,45 Rotated Ellipse
CIRCLE  ,60,40,20,18, , , ,45  Octagon
CIRCLE  ,260,40,20, , , , ,90  Diamond
CIRCLE  ,60,140,20,18, , , ,120 Triangle
```

The PAINT command lets you fill an outlined area with color. The syntax for PAINT is as follows:

```
PAINT [source] [,[X1,Y1] [,halt] ]
```

The [source] specifies the color source to paint with (Background Foreground, Multi-Color1, Multi-Color2). Painting will begin at the current PC location or at [X1,Y1] if specified. You can state the [X1,Y1] coordinate as absolute values, relative offsets, or distance and angle.

Painting continues around the [X1,Y1] location until an outline is encountered. The [,halt] parameter specifies what type of outline will stop the PAINT. If [,halt] is 0, PAINT will fill to an outline of the same color source as used in the PAINT command. If [,halt] is 1, PAINT will fill to any Foreground or Multi-Color outline.

The final PC location will be at [X1,Y1] when painting is completed. If the [X1,Y1] coordinate lies on a dot of the same color source as used with the PAINT command, then no painting is done. The first example Illustrates this:

```
CIRCLE 1,160,100,65,50: PAINT 1
```

In the examples below, lines 10 through 30 are executed prior to the PAINT commands:

```
10  COLOR 1,11: GRAPHIC 1,1
20  CIRCLE 1,80,100,50,40
30  CIRCLE 2,100,100,50,40
50  PAINT 1,110,100,1
60  PAINT 2,110,100,0
70  PAINT 0,110,100
```

Line 50 will fill the overlapping portions of the two circles with the current foreground color. Line 60 will fill all of the Multi-Color circle drawn by line 30. Using a color source value of 0 (Background color) effectively erases an area and its outline, as shown by line 70.

Up until now, our programming examples have been using X and Y coordinates that are the same as the X and Y coordinates recognized by the video controller chip in your computer. In addition to this "standard" coordinate system, the SUPER EXPANDER 64 provides an alternate scale for X and Y coordinates. You can switch from one scale to the other, by using the SCALE command. The syntax for the SCALE command is:

```
SCALE < n >
```

When < n > is 0, THE SUPER EXPANDER 64 uses the standard coordinate system. In SCALE 0, the X-Y coordinates for the screen boundaries depend on the particular GRAPHIC mode you are using.

When < n > is 1, the SUPER EXPANDER 64's special coordinate system is used. In SCALE 1, both X and Y coordinates range from 0 through 1023, regardless of which GRAPHIC mode you have selected. Here is a table showing the screen boundary coordinates for each of the Bit Map GRAPHIC modes in both scales:

Scale	**Mode**	**X-Coord.**	**Y-Coord.**
0	Multi-Color (GRAPHIC 1)	0 thru 159	0 thru 199
0	High Resolution (GRAPHIC 2)	0 thru 319	0 thru 199
0	Split Screen (GRAPHIC 3)	0 thru 319	0 thru 159
1	All GRAPHIC Modes	0 thru 1023	0 thru 1023

You can see from the preceding table that many programming tasks can be made easier using SCALE 1 since it gives you the same X and Y coordinates for **all** GRAPHIC modes. Here is an example using both scales in a program:

```
10   COLOR 1,9, , ,12: GRAPHIC 2,1
20   SCALE 0: CIRCLE 1,160,100,50,40
30   SCALE 1: COLOR ,6: CIRCLE 1,512,512,140,280
40   GOTO 40
```

This program draws a brown circle on the screen using SCALE 0, then draws a blue circle just inside the first one, using SCALE 1. The origin for both circles is the exact center of the screen.

You can see from the table of screen boundary coordinates that the ranges of X-Y coordinate values assume definite ratios to each other. These ratios vary depending on the particular graphic mode you are using.

Here is a table which shows the ratios of X-Y coordinate values from each of the Graphic modes. You can use these ratios in your programs to draw "true" circles, squares, etc. These ratios will make the proportions of the shapes you draw accurate, even though they may not appear "perfect" due to differences in video monitors.

Mode	**To Calculate X**	**To Calculate Y**
Multi-Color (GRAPHIC 1)	.8 * Y	1.25 * Y
High Resolution (GRAPHIC 2)	1.6 * Y	.625 * X
Split Screen (GRAPHIC 3)	1.6 * Y	.625 * X

See the photographs on the back of this manual and Appendix III SAMPLE PROGRAMS for more examples of the BOX, CIRCLE, PAINT, and SCALE commands.

Mixing Text with Graphics

In this section, we will discuss the CHAR command which lets you mix text (upper case and graphics characters only) with your Multi-Color and High Resolution Bit Map displays. In contrast with the PRINT command which can be used only in Standard Text mode or on the bottom five lines in Split Screen mode, the CHAR command can display text anywhere on the screen, **in any mode.**

The syntax for the CHAR command is:

CHAR [source] ,< column,row >, < string > [,reverse]

The [source] is the color source for the text. In High Resolution and Split Screen modes, only [source] values of 0 or 1 have any meaning. Using a [source] value of 0 (Background), you can "erase " text that you previously put on the screen. Using a [source] value of 1 will display text in the current Foreground color.

The < column, row > parameters are required. The < column > can range from 0 through 39, with column 0 being the leftmost side of the screen. The < row > can range from 0 through 24, with row 0 being the top line of the screen. These parameter ranges correspond with the 40 by 25 screen size in Standard Text mode.

For the < string > parameter, you can use a literal string inside double quote marks ("string") or the name of a string variable in your program. Just as with the PRINT statement, if your CHAR text string overflows the current character row on the screen, it will automatically be continued on the next row, starting in the first column.

The [,reverse] parameter can have a value of 0 or 1, and provides for displaying reverse video characters. The effects of the reverse parameter depend on the graphic mode you are using. In Standard Text mode, the [,reverse] parameter is ignored, and the CHAR string is displayed exactly as it would be seen using PRINT. **This includes acting on** (instead of printing) **control characters** for color selection, cursor movement, and reverse field.

In High Resolution or Split Screen modes, if the [,reverse] parameter has a value of 0, text will be displayed normally. If [,reverse] has a value of 1, the entire CHAR string is displayed in reverse video. **In these modes (and in Multi-Color mode), control characters in the string are ignored as "controls" and will be printed.**

In Multi-Color mode, the effects of the [,reverse] parameter depend on the value of the [source] parameter. **When [source] is 1, 2, or 3** and the [,reverse] parameter is zero, text is displayed normally; and when [,reverse] is 1, text is displayed in reverse video. In either case, the selected color source is used.

In Multi-Color mode, when [source] is zero, text will **always be displayed in Foreground color,** instead of the Background color which would be used in other modes. In addition, **when [,reverse] is also 0,** the text characters will be displayed with a Background of Multi-Color1; and **when [,reverse] is 1,** the characters will be displayed with a Background of Multi-Color2.

Here is a chart of how the values of [source] and [,reverse] affect how CHAR will display text in **Multi-Color** mode.

Source	**Reverse**	**Description of Text Displayed**
1, 2, 3	0	Normal video in selected color source
1, 2, 3	1	Reverse video in selected color source
0	0	Dot pattern of text is Foreground, background is Multi-Color1
0	1	Dot pattern or text is Foreground, background is Multi-Color2

To illustrate the preceding descriptions of CHAR, here is a sample program that shows the effects of the [source] and [,reverse] parameter values in both High-Resolution and Multi-Color modes:

```
 10  COLOR 1,0,6,9,15: GRAPHIC 2,1
 20  CHAR 0,1, 1,“SOURCE = 0  REVERSE = 0”,0
 30  CHAR 0,1, 3,“SOURCE = 0  REVERSE = 1”,1
 40  CHAR 1,1, 5,“SOURCE = 1  REVERSE = 0”,0
 50  CHAR 1,1, 7,“SOURCE = 1  REVERSE = 1”,1
 60  CHAR 2,1, 9,“SOURCE = 2  REVERSE = 0”,0
 70  CHAR 2,1,11,“SOURCE = 2  REVERSE = 1”,1
 80  CHAR 3,1,13,“SOURCE = 3  REVERSE = 0”,0
 90  CHAR 3,1,15,“SOURCE = 3  REVERSE = 1”,1
100  GET A$: IF A$ = “ ” THEN 100
110  IF RGR (0) = 2 THEN : GRAPHIC 1,1: GOTO 20
120  STOP
```

The program will first display several lines of text in High Resolution mode, then wait for you to press any key. The program will then change to Multi-Color mode and display the same text again. All possible combinations of the [source] and [,reverse] parameters are included in this program.

You can use CHAR with a color source of 0 (Background) to erase normal or reverse video of text with a color source of 1 (Foreground); or, to put text into an area which was painted or filled with a BOX command (in Foreground color). For more examples of the CHAR command, see the photographs on the back of this manual and Appendix III SAMPLE PROGRAMS.

Saving and Replacing Shapes

A powerful feature of the SUPER EXPANDER 64 is its ability to transfer graphic shapes from the screen into BASIC string variables or vice versa (from strings onto the screen). The SSHAPE command is used to transfer a rectangular area of a Bit Map screen (Multi-Color, High Resolution, or Split Screen mode) into a BASIC string variable.

The syntax for the SSHAPE command is as follows:

```
SSHAPE <stringname>, < X1,Y1 > [,X2,Y2]
```

The <stringname> is the name of the BASIC string variable to receive the shape data. Similar to the BOX command, the optional [,X2,Y2] parameter defines the starting point for saving the shape. If [,X2,Y2] is omitted, the current PC location is used. The required < X1,Y1 > parameter locates the diagonally opposite corner of the rectangular area to be saved. The SSHAPE command does not affect the location of the PC.

Because BASIC limits string lengths to a maximum of 255 characters, the size of the area you can save with SSHAPE is limited. You can calculate the size of the string needed to store a given shape by using one of the following formulas (these formulas assume that SCALE 0 is being used):

For Multi-Color Mode, String Size Equals:

```
INT ( (ABS (X1 – X2) + 1) / 4 + .99) * (ABS (Y1 – Y2) + 1) + 4
```

For High Resolution or Split Screen Modes, String Size Equals:

```
INT ( (ABS (X1 – X2) + 1) / 8 + .99) * (ABS (Y1 – Y2) + 1) + 4
```

If you are using SSHAPE in SCALE 1, divide the difference of "X1 – X2" by 3.2 and divide the difference of "Y1 – Y2" by 5.12. Then, substitute these results for the respective subtract operations in either of the preceding formulas. For example, if you are using SSHAPE in SCALE 1, you can determine the string size for High Resolution or split screen mode with the following steps:

```
A = INT (ABS(X1 – X2)/3.2 + 1)
B = INT (ABS (Y1 – Y2)/5.12 + 1)
SIZE = INT (A/8 + .99)*B + 4
```

The shape is transferred into the string, pixel row by pixel row. The last four bytes of the string will contain the column and row lengths. These are used by the GSHAPE command to put strings back onto the screen. Here are some examples to illustrate the SSHAPE command:

SSHAPE A$,0,0,50,50	This saves an area at the upper left corner of the screen
SCALE 0: SSHAPE B$, – 40, – 20,200,120	This saves an area near the center of the screen
SCALE 1: SSHAPE D$(6),512,512,640,580	This saves an area near the center of the screen
SSHAPE K$,480,480	Saves from the PC to 480,480

The GSHAPE command is the opposite of SSHAPE. It places the contents of a string variable onto a Bit Map screen display. The syntax for the GSHAPE command is:

GSHAPE < stringname > [, [X1,Y1] [,method]]

The < stringname > is the name of a BASIC string variable which is used as the source for drawing the shape on the screen. The optional [X1,Y1] parameter locates the upper left corner of the shape as it is drawn on the screen. If [X1,Y1] is omitted, the current PC location is used. The optional [,method] parameter specifies how the shape will be drawn on the screen. The GSHAPE command does not change the PC location.

The [,method] parameter allows you to place the shape in any of five different ways, combining shape data from the string with images already on the screen. This parameter can have a value from 0 through 4. Here is a table showing the values and their corresponding methods of placement; then each method is discussed in more detail:

0	Draw shape AS IS
1	Draw shape INVERTED
2	'OR' shape with screen
3	'AND' shape with screen
4	'XOR' shape with screen

AS IS: Draws the shape AS SAVED, overlaying the present display. The string shape replaces what had formerly been displayed.

INVERTED: Draws the shape as saved, changing the Foreground color areas of the string to Background color and vice versa.

'OR': Performs a logical OR of shape data in the string with the image data on the screen. This effectively "adds" the shape in the string to the screen. The result is that both shapes are merged together and all parts of both are displayed.

'AND': Performs a logical AND of shape data in the string with the image data on the screen. This effectively "erases" the parts of the two shapes that do not coincide. The result is that only the points where the two shapes coincided are displayed.

'XOR': Performs a logical XOR (Exclusive OR) of shape data in the string with the image data on the screen. The result is that only those portions of the two shapes that were different are now displayed.

These descriptions of the [,method] parameter apply to high resolution images. In Multi-Color mode, the effects of inversion and logical operations on the shapes may instead be a change of color. This depends on the color sources used for the two shapes since Multi-Color dots on the screen are two bits wide.

Here is an example of GSHAPE and SSHAPE:

```
10  SCNCLR
20  GRAPHIC 2,1:SCALE 1:COLOR 6,14, , ,6
30  CIRCLE ,530,530,18,18
40  PAINT 1,530,530,0
50  SSHAPE D$,512,512,640,580: REM SAVE COLORED CIRCLE
60  FOR I = 0 TO 1000: NEXT I
70  SCNCLR: FOR I = 0 TO 1000: NEXT I
80  GSHAPE D$,512,512: REM RESTORE COLORED CIRCLE
```

In addition to saving and restoring shapes to and from the screen, you can use GSHAPE and SSHAPE with SPRSAV (a sprite shape handling command) to transfer graphic shapes to sprites; or, from sprites to the screen. We will discuss this idea in greater detail in the chapter on Sprite Graphics.

To see a colorful illustration of the SSHAPE and GSHAPE commands, turn to the photographs on the back cover of this manual. To see the program listing which created this picture, turn to Appendix III SAMPLE PROGRAMS Listing 5.

3 SPRITE GRAPHICS

Sprite Graphics

- Sprite Designer Mode
- Defining Sprite Characteristics
- Saving Sprites
- Animating Sprites
- Handling Sprite Collisions

One of the most interesting features of the Commodore 64 is its ability to display movable objects called "sprites". Sprites are graphic images that you define and place anywhere on or off the screen. Sprites are especially suited for video graphics and arcade-type animation. You can display up to eight sprites on the screen at any given time. You can define each sprite as either a High Resolution or Multi-Color shape and can expand each in the X and/or Y directions.

Also, sprites can be combined with each other to create large and colorful graphic images. Each sprite can be assigned a "display priority" which makes it appear to move in front of or behind images in your Bit Map graphics display. This feature lets you create a three dimensional graphics effect. The SUPER EXPANDER 64 can even detect when any sprite bumps into or "collides with" another sprite or a shape in your Bit Map display.

Sprite Designer Mode

The SUPER EXPANDER 64 features a Sprite Designer Mode which makes it very easy for you to design and build sprite images. You can enter the Sprite Designer from either Direct Mode or from your BASIC program. While you are using the Sprite Designer, execution of your BASIC program is **suspended**; and several keyboard "function controls" for sprite design are provided by the SUPER EXPANDER 64. To enter the Sprite Designer Mode, use the **SPRDEF** command. This command has no parameters.

When you first enter the Sprite Designer, the screen is cleared and a large area for designing a sprite appears on the left portion of the screen. Just below this area, the prompt **"SPRITE NUMBER?"** is displayed. You should enter a number 0 through 7 which corresponds to the sprite you wish to define or modify. (Note, you do not have to press RETURN after the sprite number.) The current definition of the selected sprite is then displayed in the large design area, and also in its actual size on the right side of the screen.

At the top left corner of the design area, you will see a cursor that is either a single or double plus sign (+, + +) for High Resolution or Multi-Color sprites respectively. You can use your cursor control keys to move the cursor within the design area. The HOME key places the cursor at the upper left corner of the design area. The CLR key erases the design area to Background color and also places the cursor at the upper left corner.

There are four keys that are used as special "function controls" to make designing sprites more convenient. They are the **"A, M, X,** and **Y"** keys. Each of these keys "toggles" a specific function ON or OFF each time the key is pressed. These function keys are listed below:

KEY	FUNCTION
A	Automatic cursor movement
M	Toggles sprite display mode
X	Toggles sprite X expansion
Y	Toggles sprite Y expansion

While the Automatic Cursor function (A) is ON, when you fill a dot in the sprite design with color, the cursor is automatically moved to the next position. When this function is OFF, you must use the cursor control keys.

Each time the 'M' key is pressed, it toggles the "actual size" sprite, located at the right of the screen, from High Resolution to Multi-Color mode or vice versa. Also, the image displayed in the design area will change accordingly.

The 'X' key expands or reduces the horizontal size of the sprite each time the key is pressed. Similarly, the 'Y' key expands or reduces the vertical size of the sprite each time the key is pressed.

You can set the Foreground color of the sprite by using your Commodore 64's color control keys in the usual way. Pressing the CTRL key with a number key (1-8) selects color codes 0 through 7.

Pressing the Commodore key (C=) with a number key (1-8), selects color codes 8 through 15. This setting of Foreground color for the sprite is a temporary convenience for viewing the sprite while you are defining its shape. The SPRITE command (described later in this chapter is used to "permanently" set Sprite Foreground colors in your programs.

The Foreground color for sprites is defined **separately and independently from** the Bit Map for Text Foreground color. The same is true of multi-color selection for sprites. However, the Multi-Color1 and Multi-Color2 sprite colors can be set **only outside the sprite designer** using the SPRCOL command (described later in this chapter). Thus, if you will be designing multi-color sprites, you should use the SPRCOL command before entering the Sprite Designer.

When you are ready to begin filling the design area with color, the number keys 1 through 4 are used to select a **color source** for each dot in the design area. The following list shows these numbers and their associated color sources:

KEY	**COLOR SOURCE**
1	Screen Background color
2	Sprite Foreground color
3	Sprite Multi-Color1
4	Sprite Multi-Color2

Finally, **when you have finished designing a sprite and want to preserve it for later use,** simply press and hold the SHIFT key while you strike the RETURN key. Pressing the STOP key by itself cancels all of the changes that you have made to the design area of the screen, but the current definition of the sprite is left intact.

Whether the sprite is saved or not, the Sprite Designer returns to the **"SPRITE NUMBER?"** prompt. At this point, you can either enter a sprite number to continue defining sprites, or exit the Sprite Designer by pressing the RETURN key. If you had used the SPRDEF command in a BASIC program, and exit the Sprite Designer, the SUPER EXPANDER 64 will resume execution of your program.

Defining Sprite Characteristics

After you have defined your sprite images, and before using them in your programs, you must next define their display characteristics: colors, sizes, display priorities, and mode. The SPRCOL and SPRITE commands let you choose each of these characteristics for your sprites.

The SPRCOL command sets the Multi-Color1 and Multi-Color2 colors for **all sprites.** If you will be using the Sprite Designer to create Multi-Color mode sprites, you should use the SPRCOL command first; otherwise,the design area and the "sample sprite" may not be properly viewable.

These Multi-Color colors for sprites should not be confused with the Multi-Color1 and Multi-Color2 Bit Map colors as set by the COLOR command. The Sprite Multi-Color colors are separately defined by the SPRCOL command.

The syntax of the SPRCOL command is:

```
SPRCOL [smcr-1] [,smcr-2]
```

The [smcr-1] parameter sets Multi-Color1 for **all sprites**, and [,smcr-2] sets Multi-Color2 for **all sprites.** Either of these parameters may be any color code from 0 through 15, corresponding to the table shown for the COLOR command. If either color parameter is omitted, its current color value is left unchanged. Only sprites which are set to Multi-Color mode by the SPRITE command will display either of these colors. Here are some examples of setting sprite Multi-Color colors:

```
10  MA=2: MB=7: SPRCOL MA, MB: REM
                SET SPRITE MULTI-COLOR1 TO
                RED AND MULTI-COLOR 2 TO YELLOW
150  SPRCOL ,6: REM
                LEAVES MULTI-COLOR1 UNCHANGED
```

To check what sprite Multi-Color values you last set, use the RSPCOL function. This function returns the current color code value for the sprite Multi-Color "registers". The syntax of the RSPCOL function is:

```
RSPCOL (<register>)
```

The < register > argument may have a value of 0 or 1. When <register> is 0, RSPCOL returns the Sprite Multi-Color1 color code as a number from 0 through 15. Similarly, when < register > is 1, RSPCOL returns the color code for Sprite Multi-Color2. Here are some examples of how to use the RSPCOL function:

```
200   PRINT RSPCOL (0), RSPCOL (1): REM
                        PRINTS THE VALUES OF BOTH SPRITE
                        MULTI-COLORS
300   IF RSPCOL (0) = 6 THEN: SPRCOL 4: REM
                        SETS SPRITE MULTI-COLOR1 TO PURPLE,
                        IF IT IS NOW BLUE
```

For each of your sprites, you can use the SPRITE command to set certain characteristics. This includes turning sprites ON or OFF and setting a mode, Foreground color, and display priority. With the SPRITE command, you can also change the horizontal and/or vertical size of each sprite. Any parameters that you omit, will leave those characteristics unchanged from their last settings. The SPRITE command syntax is:

```
SPRITE < number > [, [on/off] [, [fgnd] [, [priority] [, [x-exp]
[, [y-exp] [,mode] ] ] ] ] ]
```

The required < number > parameter specifies which sprite the command refers to and its value must range from 0 through 7. The [on/off] field determines whether the sprite is displayed or not. When the value of this field is 1, the sprite is displayed (ON); and when this field is 0, the sprite is turned OFF. The [fgnd] parameter sets the Foreground color of the given sprite to a color code from 0 through 15.

The [priority] field gives the sprite a "display priority" that causes it to "appear" as if it moves in front of or behind your Bit Map display when you set the sprite in motion. If the [priority] value is 1, the sprite will appear "behind" the Bit Map display. This sets the sprite to "Low" priority. If the [priority] value is 0, the sprite wil appear "in front of" the display, setting the sprite to "High" priority. This is how you can create a three dimensional effect for your animated color graphics. With respect to each other, sprite priority is fixed with the lower numbered sprites being displayed in front of higher numbered sprites. Sprite 0 has the highest priority and sprite 7 has the lowest priority.

The sprite expansion parameters, [x-exp] and [y-exp], let you independently expand a sprite to twice its size in either or both the X-Y directions, or reduce the sprite to normal size. When the [x-exp] field is 1, the sprite becomes twice as large in the horizontal (X) direction; and when the [x-exp] field is 0, the sprite is displayed at normal width. Similarly, when the [y-exp] field is 1, the sprite becomes twice as large in the vertical (Y) direction; and when the [y-exp] field is 0, the sprite is displayed at normal height. When you expand a sprite, the number of dots that define the sprite's image does not change; instead, the size of each dot is doubled in the direction of your choice.

The [,mode] parameter determines whether a sprite will be displayed as a High Resolution or a Multi-Color shape. When the value of [,mode] is 0, the sprite is displayed as a High Resolution shape, 24 dots wide by 21 dots high. When the value of [,mode] is 1, the sprite is displayed as a Multi-Color shape, 12 dots wide by 21 dots high.

You can display High Resolution sprites in two colors: the Background color (as set by the COLOR command) and the Sprite Foreground color (as set with the [fgnd] field of this command). For Multi-Color sprites, you have two additional colors to use: Sprite Multi-Color1 and Sprite Multi-Color2 (as set by the SPRCOL command).

In SUPER EXPANDER 64 programs, you cannot display sprites in Standard Text mode. However, **in any of the Bit Map graphic modes,** you may use either or both High Resolution or Multi-Color sprites. If you press the STOP key or an error condition halts your program, the SUPER EXPANDER 64 will automatically return you to Standard Text mode and turn off all sprites and sound.

Here are some examples of the SPRITE command:

```
10 SPRITE 1,1,1,0,0,0,1:   REM SETS SPRITE 1 ON,
                           FOREGROUND WHITE, MULTI-COLOR
20 SPRITE 7,1,7,0,1,1,0:   REM SETS SPRITE 7 ON, FGND
                           YELLOW, X AND Y EXPANSION ON
                           HIGH RESOLUTION MODE
30 SPRITE 5, , , ,1:   REM EXPANDS SPRITE 5 IN X, ALL OTHER
                       CHARACTERISTICS ARE UNCHANGED
```

To check what characteristics were last set for each of your sprites, use the RSPR function. This function has **two arguments:** a sprite number and a number which represents a parameter position of the SPRITE command. The syntax of the RSPR function is as follows:

RSPR (< sprite > , < field >)

The < sprite > argument is a value from 0 through 7 and represents the number of the sprite you want information about. The < field > argument is a value from 0 through 5 that states which characteristic you are checking. Each of the < field > argument values for the RSPR function corresponds directly to a parameter position on the SPRITE command.

Thus, for the stated sprite number, the RSPR function returns the last setting of any of the parameters of the SPRITE command. Here is a table that shows the values for the < field > argument, the characteristic checked, and the values that can be returned:

Field	Characteristic	Values Returned
0	Sprite Display ON/OFF	0 = OFF; 1 = ON
1	Sprite Foreground color	0 thru 15 (color code)
2	Display Priority	0 = High; 1 = Low
3	Sprite Expanded in X	0 = No; 1 = Yes
4	Sprite Expanded in Y	0 = No; 1 = Yes
5	Sprite Display Mode	0 = Hi Res; 1 = Multi-Color

Here are two examples of the RSPR function:

```
10 IF RSPR (5,5)=1 THEN :SPRITE 5,,2
20 REM IF SPRITE 5 IS MULTI-COLOR MODE, THEN SET ITS
   FOREGROUND COLOR TO RED
```

```
10 FOR S=0 TO 7: FOR N=0 TO 5
20 PRINT RSPR (S,N);: NEXT N
30 PRINT: NEXT S
40 REM LINES 10 THROUGH 30 WILL PRINT A LIST OF ALL
   THE CURRENT CHARACTERISTIC VALUES FOR ALL
   SPRITES
```

For examples of the SPRDEF, SPRCOL, and SPRITE commands, see the photographs on the back cover of this manual and Appendix III SAMPLE PROGRAMS.

Saving Sprites

Just as you can transfer portions of the Bit Map screen into BASIC string variables (using the SSHAPE command), you can transfer sprite shapes into string variables using the SPRSAV command. SPRSAV also can transfer data in string variables into sprites, or from sprite to sprite. The syntax for the SPRSAV command is the following:

```
SPRSAV  < origin >, <destination>
```

Either parameter of the SPRSAV command can be a string variable name or a sprite number. Also, both parameters can be sprite numbers, but they cannot both be strings. To transfer a sprite image to a string, < origin > would be a sprite number and < destination > would be a string name. Conversely, to transfer string data to a sprite, < origin > would be a string name and < destination > would be a sprite number.

When both parameters are sprite numbers, the < destination > sprite assumes the same definition as the < origin > sprite (which is left unchanged). In this case, the resulting appearance of the two sprites may not be identical because their characteristic definitions may not be the same. This is because **only the dot pattern** of the <origin> sprite is copied to the <destination> sprite.

The dot pattern of a sprite is saved pixel row by pixel row, just as with the SSHAPE command. Likewise, the last four bytes of the <destination> string contains the column and row lengths. The format of the strings created by the SSHAPE and SPRSAV commands are identical. Thus, you can use SPRSAV to transfer a sprite to a string, and then use GSHAPE to draw the sprite as a shape on your Bit Map screen.

Similarly, you can use SSHAPE to transfer a shape from the Bit Map into a string, and then use SPRSAV to put the string into a sprite. When you transfer shapes to sprites in this manner, be sure that you save an area the same size as the sprite in dot positions (an area of 24 by 21 dots for High Resolution or an area of 12 by 21 dots for Multi-Color).

Here are a few examples of the SPRSAV command:

```
SPRSAV 1,A$: REM TRANSFERS THE DOT PATTERN OF
             SPRITE 1 TO THE STRING NAMED A$

SPRSAV B$,2: REM TRANSFERS THE STRING B$ INTO
             SPRITE 2

10 C=: D=4: IF RSPCOL(0)=2 THEN: SPRSAV C,D
20 REM IF THE SPRITE MULTI-COLOR1 IS RED, THEN
   TRANSFER THE DOT PATTERN OF SPRITE 1 TO SPRITE 4
```

Animating Sprites

One of the most interesting and powerful features of the SUPER EXPANDER 64 is the ability it gives you to position and set sprites in motion. For each sprite, you can use the MOVSPR command to set its position, start it moving, and stop it. The syntax for the MOVSPR command is as follows:

MOVSPR < number > <,X1,Y1>

The < number > is the sprite's number (0 through 7) whose position you want to set or change. The < ,X1,Y1 > coordinate is the new location for the sprite. To position a sprite, you can state < X1,Y1 > as absolute values or as a relative offset. However, when using a relative offset to position a sprite, you should be aware that the new position is calculated **from the current sprite position, instead of the current PC location.**

Sprites are positioned with respect to their upper left hand corner. There is a specific portion of your screen or "window" where sprites are visible. The coordinates that define this window are different from the coordinates that define the boundaries of a Bit Map screen. For example, the top left corner of a Bit Map screen is (0,0). To position the top left corner of a sprite to the same location, the sprite X-Y coordinates would be (24,50). Use the table of X-Y coordinate ratios (see SCALE) to calculate both absolute placement coordinates or relative move distances for sprites.

You can also use a special form of the < ,X1,Y1 > parameter of the MOVSPR command to set a sprite in motion or to stop it. In this case, the "X1" and "Y1" parts of the parameter are separated by a number sign (#) instead of a comma. The "X1" value specifies a clockwise angle in degrees for the direction the sprite will move. The "Y1" value is a number from 0 through 15 and specifies a constant speed for the sprite (where 0 stops motion of the sprite and 15 is the fastest speed). The speed of a sprite is measured by the number of dot positions (in the indicated direction) that the sprite will be moved in a period of time. Sprite motion is actually a series of instantaneous relative offsets which your eye recognizes as smooth motion.

You can independently set any or all sprites in motion at varying speeds and directions simultaneously. When display of a sprite is turned OFF by the SPRITE command, its motion is stopped automatically. When the sprite is turned back ON, its motion will resume automatically at the former speed and direction.

Here are some examples of the MOVSPR command:

```
MOVSPR 1,160,100:   REM PUTS THE UPPER LEFT CORNER
                    OF SPRITE 1 AT THE CENTER OF THE
                    HIGH RESOLUTION SCREEN
MOVSPR 1,+40,-60:   REM USES RELATIVE OFFSET TO
                    MOVE SPRITE 1 BY 40 DOTS TO THE
                    RIGHT AND 60 DOTS UPWARD
MOVSPR 2,90#8:      REM SETS SPRITE 2 IN MOTION AT A
                    90 DEGREE ANGLE AND A SPEED OF 8
```

You can check the position and speed of sprites by using the RSPPOS function. This function has **two arguments**: a sprite number and a number which requests the X position, Y position, or speed of the sprite. The syntax of the RSPPOS function is as follows:

```
RSPPOS( <sprite> , <data> )
```

The < sprite > argument identifies which sprite you are checking. The < data > argument specifies what information is to be returned. When < data > is 0, the current X position of the sprite is returned, and when < data > is 1, the current Y position of the sprite is returned, and when < data > is 2, the current speed of the sprite is returned as a number from 0 through 15.

NOTE THAT RSPPOS ALWAYS RETURNS SCALE 0 COORDINATES.

Here are some examples of the RSPPOS function:

```
RSPPOS (4,0): REM RETURNS THE CURRENT X POSITION OF
              SPRITE 4
RSPPOS (7,1): REM RETURNS THE CURRENT Y POSITION OF
              SPRITE 7
```

```
10 S=3: D=2: PRINT RSPPOS(S,D)
20 REM PRINTS THE CURRENT SPEED VALUE OF SPRITE 3
```

Handling Sprite Collisions

The SUPER EXPANDER 64 program gives you the ability to detect when any of your moving sprites "collide" into each other or images in your Bit Map display. You can detect sprite collisions by using the COLINT command and determine which sprites collided by using the RBUMP function. This gives you the ability to create animated graphic displays in BASIC programs.

The COLINT command detects three types of events: collisions between sprites, collisions between sprites and Bit Map images, and light pen activation. When one of these events occurs, your program finishes the current statement and then the SUPER EXPANDER 64 transfers control to the first line number of your collision handling subroutine.

In other words, your BASIC program is "interrupted" and sent to your collision subroutine. We will later refer to any of these events as "collision interrupts". After a RETURN statement in your collision subroutine is executed, control is returned to the statement which follows the one that was just interrupted.

The syntax for the COLINT command is as follows:

```
COLINT < event > [,line-num]
```

The < event > parameter can have a value of 0 through 2 and specifies which type of event should cause a collision interrupt. When < event > is 0, sprite to sprite collisions are detected; when <event > is a 1, sprite collisions with the Bit Map display are detected; and when < event > is 2, light pen activation causes an "interrupt".

The [,line-num] parameter is the first line number of the subroutine in your BASIC program where control will be transferred when a collision interrupt of the stated < event > type occurs. When the optional [,line-num] parameter is specified, then collision detection is turned ON (enabled) for the given type of < event >. When the [,line-num] parameter is omitted, collision detection is turned OFF (disabled) for the specified type of < event> .

A sprite to sprite collision occurs when any part of a sprite that is not Background color occupies the same location as a non-Background portion of any other sprite. A sprite cannot cause a collision interrupt when it is completely off the screen (not visible). A sprite to Bit Map collision occurs when any part of a sprite that is not Background color occupies the same location as any non-Background image on the screen (i.e., Foreground, Multi-Color1, or Multi-Color2). Sprites that have been turned OFF (disabled) using the SPRITE command do not cause collision interrupts.

Here are some examples of the COLINT statement:

```
50 COLINT 0,500: REM START DETECTING SPRITE TO SPRITE
                 COLLISIONS
60 COLINT 1,600: REM START DETECTING SPRITE TO BIT
                 MAP COLLISIONS
500 COLINT 0: COLINT 1: REM DISABLE SPRITE TO SPRITE
                        AND SPRITE TO BIT MAP
                        COLLISIONS
```

In the preceding examples, line 50 causes control to be transferred to a subroutine at line 500 whenever a sprite to sprite collision occurs. Similarly, line 60 causes control to be transferred to a subroutine at line 600 when a sprite to Bit Map collision is detected. In line 500, detection of any further sprite collisions is disabled while the current collision is handled.

You can have any/all types of < event > detection active at the same time; but, **only one collision can be handled at a time.** Therefore, you should always **disable further detection of collisions** as the first step in your collision handling subroutines. This prevents further interrupts from occurring while you are processing the present one. Also, the last step you should take in your collision interrupt subroutine is to reenable collision detection.

To check which sprites have collided, you can use the RBUMP function. This function returns information about which sprites have collided with other sprites, or which sprites have collided with the Bit Map display. You do not have to have collision interrupts ON in order to use RBUMP. The syntax for the RBUMP function is:

```
RBUMP ( < event > )
```

The RBUMP < event > argument corresponds directly to the <event> type of the COLINT command. When < event > is 0, the RBUMP function returns information about sprite to sprite collisions; when < event > is 1, RBUMP returns information on which sprites collided with the Bit Map display.

In either of the preceding cases, RBUMP returns a number from 0 through 255. The bit positions (0-7) in the number returned by RBUMP correspond to sprite numbers 0 through 7. When a bit is turned ON (has a value of 1), the sprite in that bit position was involved in a collision. In the event of multiple collisions occurring simultaneously, you should also use the RSPPOS function previously discussed to determine which sprite collided with what object.

The RBUMP function reads the hardware sprite collision registers in the video controller chip of the Commodore 64. These registers are automatically set to zero whenever they are read, whether you use the RBUMP function or the PEEK function to read them directly. Thus, if you need to refer to the information returned by RBUMP more than once, you must assign the value to a variable name.

Here is an example of the RBUMP function:

```
1000 A = RBUMP(0): B = RBUMP(1)
1010 REM LINE 1000 READS BOTH THE SPRITE TO SPRITE
     AND SPRITE TO BIT MAP COLLISIONS
```

In the next example, line 530 uses the logical AND operator to check whether or not sprite 1 has collided.

```
500 MOVSPR 1,160,100: REM PUTS SPRITE 1 AT SCREEN
                           CENTER
510 FOR X = 1 TO 50: REM LOOP 50 TIMES
520 MOVSPR 1,+5,+5: REM MOVE RIGHT 5 DOTS AND
                           DOWN 5 DOTS
530 IF RBUMP (0) AND 2 THEN GOSUB 800
540 REM WHEN SPRITE 1 COLLIDES, THEN EXECUTE THE
    SUBROUTINE AT LINE 800
550 NEXT X: REM END THE LOOP
```

For examples of the SPRSAV, MOVSPR, and COLINT commands, see the photographs on the back cover of this manual and Appendix III SAMPLE PROGRAMS.

4

GAME CONTROLS and FUNCTION KEYS

- Game Port Input/Output
- Programmable Function Keys

Game Port Input/Output

The SUPER EXPANDER 64 provides three functions that let you easily read the positions of one or two joysticks, up to four game paddles, or light pen coordinates from the Commodore 64 game control ports. These include the RJOY function for joysticks, RPOT for game paddles, and RPEN for the light pen. The syntax for the RJOY function is:

RJOY (< joystick >)

The < joystick > argument can have a value of 1 or 2 and specifies whether the joystick attached to Control Port 1 or Control Port 2 is read. A number is returned by RJOY that indicates which direction the joystick is being pushed toward. When the "fire" button on the joystick is pressed, a value of 128 is added to the direction value of the number returned. The illustration below shows the values which correspond with each of the joystick directions:

```
      FIRE BUTTON
           1
      8         2
 7         0         3
      6         4
           5
```

Joystick Direction Values

Here are some examples of the RJOY function. Turn to Appendix III SAMPLE PROGRAMS for a sample program that further illustrates the use of the RJOY function.

```
10 D = RJOY(1): IF D > 127 THEN GOSUB 750
20 REM IF JOYSTICK 1 FIRE BUTTON IS PRESSED, THEN
   EXECUTE THE SUBROUTINE AT LINE 750

50 D = RJOY(2): REM READ JOYSTICK 2 DIRECTION
60 IF D= 6 THEN X = X-1: Y = Y + 1: GOTO 50
70 REM IF JOYSTICK IS PRESSED TO THE LEFT AND DOWN,
   THEN ADJUST X-Y COORDINATES ACCORDINGLY
```

Using the RPOT function, you can read the current position of up to four game paddles (or potentiometers). Paddles 0 and 1 are read from Control Port 1 and paddles 2 and 3 are read from Control Port 2. The syntax of the RPOT function is as follows:

RPOT(< paddle >)

The < paddle > argument can have a value of 0 through 3 and specifies which paddle will be read. The RPOT function returns a number from 0 through 255 which indicates the position of the control knob on the paddle. When the paddle "fire" button is pressed, a value of 256 is added to this number. Thus, when the value returned from RPOT is greater than 255, the fire button is being pressed.

Here is an example of the RPOT function. Refer to Appendix III SAMPLE PROGRAMS for an illustration of using the RPOT function in an animated game.

```
800 X = RPOT(0): Y = RPOT(1)
810 REM LINE 800 READS BOTH PADDLES ON CONTROL
    PORT 1
820 IF X > 255 OR Y > 255 THEN GOSUB 1000
830 REM LINE 820 CHECKS FOR FIRE BUTTONS PRESSED
    AND EXECUTES SUBROUTINE AT LINE 1000 IF YES
840 MOVSPR 4,X,Y: REM UPDATE SPRITE 4 POSITION
850 GOTO 800
```

The RPEN function returns the value of the X or Y coordinate of the light pen from its last activation. The collision interrupt feature does not have to be active to use the RPEN function. The coordinates returned by RPEN always use the "standard" SCALE 0 coordinate system. The syntax for the RPEN function is:

RPEN(< data >)

The < data > argument can have a value of 0 or 1 and states whether the X or Y coordinate, respectively, of the light pen is returned. The RPEN values returned can vary from one system to another. As a result, the X position is returned as an even number, ranging from approximately 60 through 380; and the Y position ranging from approximately 50 through 250.

A value of 0 is returned from RPEN when the light pen is off screen and has not triggered an interrupt since it was last read. A white (or very bright) screen background is usually required to stimulate the light pen. Note that only Control Port 1 is capable of reading a light pen.

Here is an example of the RPEN function:

```
50 X = RPEN(0): Y = RPEN(1)
60 REM LINE 50 READS THE X AND Y COORDINATES OF THE
                  PEN
70 IF X = 0 and Y = 0 THEN 50
80 REM LOOP IF LIGHT PEN IS NOT ACTIVATED
90 SPRITE 7,0: REM TURN OFF SPRITE 7 WHEN PEN IS
                  TRIGGERED
```

Programmable Function Keys

The SUPER EXPANDER 64 lets you **take full advantage** of the power of the Function Keys by programming your own definitions for them in your BASIC programs. By programming the Function Keys with frequently used commands, you can save yourself a lot of time and effort, not to mention mistakes. The SUPER EXPANDER 64 initially gives you the following definitions for the eight function keys:

f1 = "GRAPHIC"	f5 = "RUN"
f2 = "SAVE"	f6 = "CIRCLE"
f3 = "SPRDEF"	f7 = "LIST"
f4 = "LOAD"	f8 = "SPRITE"

You can use the KEY command to either program the Function Keys or check their current meanings. The syntax for the KEY command is:

```
KEY [ <keynum, string-expr> ]
```

The KEY command used without parameters will LIST the current function key definitions to the screen (output device number 3). The CMD command can be used to re-direct output to other devices such as the printer or disk unit.

When the [<keynum, string-expr>] parameters are present, the function key specified by < keynum > will be re-defined. The < keynum > value may range from 1 through 8. The < ,string-expr > may be a literal string inside double quotes ("string"), a BASIC string variable name, or a string expression formed by concatenation (linking strings together) using the '+' operator. All terms of the expression must also be strings.

After you have programmed the function keys, you can use them in both Direct mode and Program mode. In Direct mode, the string that you have programmed for that key will be printed on the Standard Text screen (this may not be visible in Bit Map graphic modes). Then, when you press the RETURN key, the string is executed as a Direct mode command (or series of commands separated by colons ':'). If a RETURN (CHR$(13)) is included as the last character of the string, simply press the function key to execute its associated command or commands.

In Program mode, you can use function keys in response to the INPUT or GET commands to fill string variables in your programs. For the INPUT command, pressing a function key prints its current definition just behind the INPUT command's question mark (?) prompt. When you press RETURN, the string is transferred to the variable of the INPUT command. As before, if a RETURN (CHR$(13)) is included as the last character of the string, simply press the function key to transfer the function key string to the variable.

The maximum combined string length for all of your Function Key definitions is 255 characters. Generally, this number of characters is more than enough for all practical purposes, since you will not usually program long command sequences for the Function Keys. To turn “OFF” one or more of the Programmable Function Keys, use the CHR$(n) function to redefine each key to its original value. These would be the character values returned by the GET command on systems not equipped with the SUPER EXPANDER 64.

For example, to turn OFF the f1 key, specify **KEY 1,CHR$(133).** Here is a short program to use or modify for setting up your own Function Key definitions:

```
10 KEY 1, “RUN” + CHR$(13)
20 KEY 2, “SAVE” + CHR$(34) + “VSPX.1” + CHR$(34) +
          “,8” + CHR$(13)
30 KEY 3, “LOAD” + CHR$(34) + “VSPX.1” + CHR$(34) +
          “,8” + CHR$(13)
40 KEY 4, “X = 100: Y = 80: GOTO 1000” + CHR(13)
50 KEY 5, “COLOR 1,11” + CHR (13)
60 KEY 6, “CIRCLE”
70 KEY 7, “LIST” + CHR$(13)
80 KEY 8, “KEY” + CHR$(13)
```

In addition to what we have specified in the preceding Function Key strings, we can also use control characters for defining the following:

- Color selection
- Cursor movement
- Reverse field video and
- Playing music strings

5

SOUND EFFECTS and MUSIC

- Defining Sound Effects
- Playing Your Sound Effects

Defining Sound Effects

The SUPER EXPANDER 64 lets you easily use the Commodore 64's Sound Interface Device (SID) chip to generate game sound effects and music. The musical scale that the SUPER EXPANDER 64 provides approximates concert pitch (where note A in octave 4 has a frequency of 440 hz). Commands are provided to determine the characteristics of the sounds to be produced. Also, you can play music in Direct mode or in Program mode. Refer to the **Commodore 64's Programmer Reference Guide** for more details on sound.

The TEMPO command is used to set the speed at which music is played. The TUNE command is used to define the waveforms and "envelopes" of the sounds. The FILTER command is used to create resonance effects and to enhance or suppress selected frequency ranges in the sounds.

The syntax for the TEMPO command is:

```
TEMPO < speed >
```

The < speed > parameter governs the relative duration of notes as they are played. The value of < speed > can range from 0 through 255. At 0, the note will sound continuously. As the < speed > value increases, the duration of the note becomes shorter. The initial value of TEMPO is set to 8.

You can calculate the actual duration of a whole note by the formula (duration = 19.22 / < speed >) in seconds. Here are some examples of setting TEMPO:

```
10 TEMPO 180: REM FAST TEMPO SOUNDS ARE GOOD FOR
                  GAMES
20 TEMPO 6: REM FOR SLOW CHAMBER MUSIC
```

The syntax for the TUNE command is:

```
TUNE < env >, [,[atk] [,[dec] [,[sus] [,[rel] [,[form] [,width] ] ] ] ] ]
```

The <env > parameter specifies the TUNE envelope number and can have a value from 0 through 9. The [atk], [dec], [sus], and [rel] parameters may have values ranging from 0 through 15 and are used together to define the Attack, Decay, Sustain, and Release (ADSR) characteristics of the waveform envelope.

The Attack is the time during which the sound builds from nothing to peak volume. The Decay is the time it takes for the sound to fall from peak volume to the sustain level. The Sustain is the volume level at which the note will be held for most of its duration. The Release is the time the sound requires to fall from the Sustain level to nothing.

The [form] parameter specifies the waveform of the sound and may have a value from 0 through 4. The following list shows the [form] parameter values and their corresponding waveforms:

Values	**Waveforms**
0	Triangle
1	Sawtooth
2	Pulse
3	Random Noise
4	Ring Modulation

The [,width] parameter has meaning only when the waveform selected is the "pulse" waveform. This parameter can have a value from 0 through 4095. A width of 2048 produces a square wave.

The SUPER EXPANDER 64 lets you define up to ten TUNE envelopes. These are set to the initial values as defined in the chart below:

Env	**Atk**	**Dec**	**Sus**	**Rel**	**Form**	**Width**	**Instrument**
0	0	9	0	0	2	1536	Piano
1	12	0	12	0	1		Accordion
2	0	0	15	0	0		Calliope
3	0	5	5	0	3		Drum
4	9	4	4	0	0		Flute
5	0	9	2	1	1		Guitar
6	0	9	0	0	2	512	Harpsichord
7	0	9	9	0	2	2048	Organ
8	8	9	4	1	2	512	Trumpet
9	0	9	0	0	0		Xylophone

Here are some examples of the TUNE command:

```
40 TUNE 0,,,,,,1280: REM CHANGES PULSE WIDTH FOR THE
                     PIANO ENVELOPE

50 TUNE 4,12,6,6: REM INCREASES ATTACK, DECAY, AND
                  SUSTAIN VALUES FOR THE FLUTE
                  ENVELOPE
```

The FILTER command is used to dynamically vary other tonal qualities of the sounds produced. You can do this by setting a filter cutoff to suppress selected ranges of frequencies. You can also specify a resonance effect which emphasizes notes with frequencies near the cutoff frequency of the filter. The syntax for the FILTER command is:

```
FILTER [freq] [,[low] [,[band] [,[high] [,res] ] ] ]
```

The [freq] parameter is the cutoff frequency for the filter in the SID chip and may range in value from 0 through 2048. To determine the actual cutoff frequency in Hz, multiply this value by 5.8 and add 30.

The [low], [band] and [high] parameters are used together to determine which parts of the audio spectrum are passed onto the output of the SID chip unaltered, and which parts are suppressed by the filter. Each of these parameters can have a value of 0 (suppress) or 1 (pass). You can set any/all of these parameters to either value.

The [,res] parameter can range from 0 through 15. This determines the resonance (i.e., how strongly the peaking effect of sounds near the cutoff frequency are emphasized). Here are some examples of the FILTER command:

```
10 FILTER 2048,,,1 Sets filter cutoff and high pass filter mode.
30 FILTER 1024,,,,10 Sets the resonance control.
70 FILTER 700,1,0,1 Set the filter for a "notch reject" mode of
                    operation to suppress sounds that are
                    nearest to the selected cutoff frequency
```

Playing Your Sound Effects

With the SUPER EXPANDER 64, music is composed using string characters. You can play music by entering these characters from your keyboard in Direct mode or by including them in PRINT strings in your programs. Notes are specified by the letters A through G. The durations of notes are indicated by the letters W (whole), H (half), Q (quarter), I (eighth), and S (sixteenth). Every note that follows one of the "duration" letters is played at the same length until you change the duration. The letter R specifies a rest for the duration of one note.

Notes that are preceded by a '#' sign are played as sharps and notes preceded by a '$' are played as flats. Notes preceded by a '.' are played as "dotted notes", at one and a half times the normal duration. Here is a chart which summarizes these music elements:

Element	**Description**
A,B,C,D,E,F,G	Notes
#	Sharp (precedes note)
$	Flat (precedes note)
	Dotted (precedes note)
W	Whole notes will follow
H	Half notes will follow
Q	Quarter notes will follow
I	Eighth notes will follow
S	Sixteenth notes will follow
R	Rest (for one note duration)

Similarly, letters are used to define certain SID chip control values. You can set the master volume control by using the letter U, followed by a number 0 through 9. The TUNE envelope is selected by the letter T, followed by a number 0 through 9. These values correspond to the envelope numbers as set by the TUNE command.

One or more of the SID chip's three voices can be active at the same time. The letter V followed by a number from 0 through 2 selects which voices will play music. The filter in the SID chip is turned ON by 'X1' and is turned OFF by 'X0'. Since there is only one filter, its current settings apply to all voices that are enabled.

The particular octave for a note is selected by the letter 'O' followed by a number from 0 through 6. The < CTRL-F > character is used as a toggle to enable and disable the actual playing of music. Here is a chart that summarizes the SID control values:

SID Control	**Description**
U	Volume (0 -9)
T	TUNE envelope (0 - 9)
V	Voice (0 - 2)
X	FILTER (0 = OFF, 1 = ON)
O	Octave (0 - 6)
< CTRL-F >	Enable/disable music playback (CHR$(6))

The initial default values for the musical parameters are: TUNE 0, Voice 0, Octave 4, Volume 9, Filter 0 (OFF), and Whole note duration. As with other features of the SUPER EXPANDER 64, once a value has been set for a music element, that value becomes the new default for later operations.

Here is an example of music strings that you can try:

```
10 PRINT “PLAYING HALF NOTES
         <CTRL-F> HCDEFGAB <CTRL-F>
         AND NOW SIXTEENTHS...”
20 PRINT CHR$(6);“SCDEFGAB”
```

Now, enter these lines and let’s “Boogie”!

```
1  POKE53280,0:POKE53281,0:PRINT CHR$(147)
10  FILTER 1200,1,0,1,9: TEMPO 13
20  PRINT CHR$(6)“O1T0X1U9”;
30  PRINT“V1IO1CO2CV2.QO4CV0O3EV1IO1EO2E”;
35  PRINT“V2.QO4CV0O3EV1IO1GO2GO1AO2A”;
40  PRINT“O1$BV2.QO4CV0O3EV1IO2$BO1AO2A”;
45  PRINT“V2.QO4CV0O3EV1IO1GO2GO1EO2E”;
50  PRINT“V1IO1CO2CV2.QO4CV0O3EV1IO1EO2E”;
55  PRINT“V2.QO4CV0O3EV1IO1GO2GO1AO2A”;
60  PRINT“O1$BV2.QO4CV0O3EV1IO2$BO1AO2A”;
65  PRINT“V2.QO4CV0O3EV1IO1GO2GO1EO2E”;
70  PRINT“V1IO1FO2FV2.QO4CV0O3$EV1IO1AO2A”;
75  PRINT“V2.QO4CV0O3$EV1IO2CO3CO2DO3D”;
80  PRINT“O2$EV2.QO4CV0O3$EV1IO2$EO2DO3D”;
85  PRINT“V2.QO4CV0O3$EV1IO2CO3CO1AO2A”;
90  PRINT“V1IO1FO2FV2.QO4CV0O3$EV1IO1AO2A”;
95  PRINT“V2.QO4CV0O3$EV1IO2CO3CO2DO3D”;
100  PRINT“O2$EV2.QO4CV0O3$EV1IO2$EO2DO3D”;
105  PRINT“V2.QO4CV0O3$EV1IO2CO3CO1AO2A”;
110  PRINT“V1IO1CO2CV2.QO4CV0O3EV1IO1EO2E”;
115  PRINT“V2.QO4CV0O3EV1IO1GO2GO1AO2A”;
120  PRINT“O1$BV2.QO4CV0O3EV1IO2$BO1AO2A”;
125  PRINT“V2.QO4CV0O3EV1IO1GO2GO1EO2E”;
130  PRINT“V2.QO4DV0O3FV1IO1GO2GV1IO1B”;
135  PRINT“V2.QO4DV0O3FV1IO2BO2DO3DO2EO3E”;
140  PRINT“V1IO1FO2FV2.QO4CV0O3$EV1IO1AO2A”;
145  PRINT“V2.QO4CV0O3$EV1IO2CO3CO2DO3D”;
150  PRINT“V1IO1CO2CV2.QO4CV0O3EV1IO1EO2E”;
155  PRINT“V2.QO4CV0O3EV1IO1GO2GO1AO2A”;
160  PRINT“O1$BV2.QO4CV0O3EV1IO2$BO1AO2A”;
165  PRINT“V2.QO4CV0O3EV1IO1GO2GO1EO2E”;
170  GOTO30
```

APPENDICES

Appendix I PROGRAMMERS NOTES

In this section, we will present additional information about the operation of the SUPER EXPANDER 64. You will find some important notes concerning the following areas:

- How Memory is Used
- Bit Map Graphic Displays
- Sprites and Collision Interrupts
- I/O and Error Handling

How Memory is Used

The SUPER EXPANDER 64 program is an auto-start cartridge which resides in memory at $8000 — $9FFF (hexadecimal) or 32768 — 40959 (decimal). The amount of BASIC program space available for use is reduced by 8192 bytes. The SUPER EXPANDER 64 also uses the RAM memory from $C000 — $CBFF (49152 — 52223). Although the rest of the memory up to $CFFF (53247) is not presently used, it is reserved for possible future system software expansion.

The SUPER EXPANDER 64 program also uses the RAM memory that exists "beneath" it and the BASIC language ROM at $8000 — $BFFF. This memory is used for screen displays, sprite patterns, programmable character definitions, etc. You can use the POKE command to change the contents of this RAM memory, but a PEEK function returns the contents of the ROM memory instead. Here is a chart which summarizes the SUPER EXPANDER 64's use of the RAM memory in these areas:

Hex Addr.	Decimal	Description
$8000	32768	Coordinate stack area for PAINT
$8A00	35328	Sprite image patterns (8 sprites)
$8C00	35840	Color control RAM for Bit Map displays
$9000	36864	Character generator ROM (thru $9FFF)
$A000	40960	Bit Map Screen (thru $BFFF)
$C000	49152	Temporary data storage and work areas
$C400	50176	RAM for programmable characters (used by CHAR and Split Screen Text)
$CC00	52224	Reserved for expansion (thru $CFFF)

A 2048 byte program used to drive the Commodore IEEE-488 interface is ordinarily stored in RAM memory starting at $C000. During system initialization, when the SUPER EXPANDER 64 detects the presence of this device, it relocates the IEEE-488 program. The IEEE-488 program is then automatically moved to $7800 (30720). This reduces the amount of program space by another 2048 bytes.

Bit Map Graphic Displays

When using the PRINT command in Multi-Color GRAPHIC mode, it is impossible to prevent the Screen Editor from updating the color control RAM. Consequently, if you print to the Text screen behind the Bit Map display, you may see some colors change. The net effect is that images displayed in Multi-Color2 will become Foreground color.

In GRAPHIC modes 2 and 3 (High Resolution and Split Screen), lines are drawn one dot wide. You can toggle between this and bold lines that are two dots wide by using the POKE command. POKE 49168,0 sets line width to one dot, and POKE 49168,1 sets line width to two dots.

In Split Screen mode, the cursor is visible only when it is somewhere on the bottom five lines of the screen. Using the HOME key or CRSR controls can move the cursor to a position **behind the Bit Map portion** of the screen where it becomes "lost" until you CRSR down.

The SSHAPE and SPRSAV commands transfer shapes to strings pixel row by pixel row. Four bytes are appended to the string which contain the column (X) and row (Y) lengths to the shape. The first two bytes are the number of X-positions less one, and the last two bytes are the number of Y-positions less one. Both numbers are stated as 16-bit "address pointers" in low-byte, high-byte format.

Sprites and Collision Interrupts

You may set the Foreground color of a sprite the same as the Background color for the Bit Map screen and the sprite will then be invisible. However, if enabled, the sprite will still cause collision interrupts with other sprites or Bit Map displays. Although you can only have eight sprite definitions active at the same time you can define more sprites by storing the definitions in strings.

When you set sprites in motion, the speed parameter of MOVSPR is used as the actual number of dot positions that the sprite moves in each time interval. In other words, the movement is a series of "instantaneous" relocations. Thus, a sprite moving at high speed can miss colliding with a thin line or shape, or another small sprite by "jumping" past it.

I/O and Error Handling

While displaying sprites, the SUPER EXPANDER 64 changes certain address pointers used to handle system interrupts. Therefore, before attempting any cassette tape I/O, you must turn OFF all sprites to restore the normal system interrupt handling.

The SUPER EXPANDER 64 reports command errors through the BASIC interpreter and thus adds no unique error messages of its own. Instead, bad statement syntax causes a "SYNTAX ERROR" message and a parameter value out of range is an "ILLEGAL QUANTITY" error. Missing parameters or arguments are reported by an "OUT OF DATA" error message. A PAINT command which runs out of coordinate stack space results in a "FORMULA TOO COMPLEX" error message.

When you halt a BASIC program by pressing the STOP key, or when BASIC halts the program on an error condition, the SUPER EXPANDER 64 will automatically select Standard Text mode and turn sprites OFF. However, any automatic motion is left active. Use the MOVSPR command to set speed to zero or use the STOP/RESTORE keys to halt sprite motion. Color selections and Bit Map display data are left unchanged. When a program terminates with an END or STOP statement, Text mode is not selected automatically.

Pressing the STOP key will not interrupt the PAINT command or playback of a music string or the drawing of a line. Pressing the RUN/STOP and RESTORE keys will reset the SUPER EXPANDER 64 to all of its initial default values, except for Function Key definitions. Typing CONT to continue a halted program will not switch back to the intended graphic mode, nor restart music playback, sprite movement, or collision detection.

Appendix II COMMAND REFERENCE GUIDE

BOX [source] <,X1,Y1> [,[X2,Y2] [,[angle] [,fill]]]

Parameter	Description
[source]	Color source for rectangle (0 – 3)
<,X1,Y1>	Specified corner coordinate
[X2,Y2]	Corner coordinate opposite < ,X1,Y1> ; (default is PC)
[angle]	Rotation in clockwise degrees (default is 0)
[fill]	Fill shape with same color as source (default is 0)

CHAR [source] , <column,row> , <string> [,reverse]

Parameter	Description
[source]	Color source (0 – 3; default 1)
<column,row>	Character column, row location
<string>	String to be displayed
[,reverse]	Reverse field, except Text mode (default is 0)

CIRCLE [source] ,[X1,Y1] < ,X-rad > [,[Y-rad] [,[start] [,[end] [,[angle] [,inc]]]]]

Parameter	Description
[source]	Color source (0 – 3); (default is 1)
[X1,Y1]	Center coordinate (default is PC)
< ,X-rad>	Radius in X
[Y-rad]	Radius in Y (default is X-rad)
[start]	Starting arc (default is 0 degrees)
[end]	Ending arc (default is 360 degrees)
[angle]	Rotation in clockwise degrees (default is 0 degrees)
[,inc]	Increment in degrees (coarseness) (default is 2)

COLINT < event > [,line-num]

Parameter	Description
<event>	Type of interrupt (0 – 2) 0 — Sprite to sprite collisions 1 — Sprite to Bit Map display collisions 2 — Light pen activation
[,line-num]	Line number of subroutine to handle the particular type of collision < event >

COLOR [bgnd] [,[fgnd] [,[mcr1] [,[mcr2] [,ext]]]]

Parameter	Description
[bgnd]	Background color (0 - 15)
[fgnd]	Foreground color (0 - 15)
[mcr1]	Multi-Color register 1 (0 - 15)
[mcr2]	Multi-Color register 2 (0 - 15)
[,ext]	External border color (0 - 15)

(Defaults will not change current color)

COLOR	CODE	COLOR	CODE	COLOR	CODE
Black	0	Green	5	Light Red	10
White	1	Blue	6	Dark Grey	11
Red	2	Yellow	7	Med. Grey	12
Cyan	3	Orange	8	Light Green	13
Purple	4	Brown	9	Light Blue	14
				Light Grey	15

DRAW [source] [,X1,Y1] [TO X2,Y2]...

Parameter	Description
[source]	Color source (0 - 3); (default) is 1)
[,X1,Y1]	Move PC to this location and draw a dot
[TO X2,Y2]	Draw a line from PC to this location

FILTER [freq] [,[low] [,[band] [,[high] [,res]]]]

Argument	Description
[freq]	Filter cutoff frequency (0 - 2048)
[low]	Low pass filter; 0 - OFF; 1 - ON
[band]	Band pass filter; 0 - OFF; 1 - ON
[high]	High pass filter; 0 - OFF; 1 - ON
[,res]	Resonance (0 - 15)

GRAPHIC <mode> [,clear]

<mode> Value	Mode Description
0	TEXT
1	Multi-Color Graphic
2	High Resolution Graphic
3	Split Screen

The [,clear] parameter defaults to 0; if specified non-zero, screen is cleared.

GSHAPE < stringname > [,[X1,Y1] [,method]]

Parameter	Description
< stringname >	Shape to be drawn
[X1,Y1]	Location to draw shape (default is PC)
[,method]	Placement of shape (0 - 4) 0 — Draw shape AS IS 1 — Draw shape INVERTED 2 — 'OR' shape with screen 3 — 'AND' shape with screen 4 — 'XOR' shape with screen

KEY[< keynum, string-expr >]

Argument	Description
<keynum>	Function Key number (1 - 8)
<,string-expr>	A literal string within double quotes; a BASIC string variable; or a string expression

(KEY with no parameters lists all commands assigned to the Function Keys)

LOCATE < X,Y >

Parameter	Description
<X,Y>	Coordinate location to place PC

MOVSPR <number> <,X1,Y1>

Parameter	Description
< number >	Sprite number
<,X1,Y1>	Location to place sprite

PAINT [source] [,[X1,Y1] [,halt]]

Parameter	Description
[source]	Color source (0 - 3); default is 0
[X1,Y1]	Start painting at this location (default is PC)
[,halt]	End painting (0 - 2) 0 — Paint to border same as color source (default) 1 — Paint to border of any Foreground color

RBUMP(<event>)

Argument	Description
<event>	Type of collision (0 - 1) 0 — Information on sprite to sprite collision 1 — Information about sprite to background collision

RCLR(<area>)

Argument	Description
<area>	One of five areas (0 - 4) whose colors (0 - 15) are set by the COLOR command 0 — Returns Background color code 1 — Returns Foreground color code 2 — Returns Multi-Color1 color code 3 — Returns Multi-Color2 color code 4 — Returns Exterior Border color code

RDOT(<data>)

Argument	Description
<data>	The X or Y coordinate, or color source for the pixel at current PC location 0 — X coordinate of PC 1 — Y coordinate of PC 2 — Color (register 0 - 3) of dot at PC

RGR (0)

Argument	Description
0	GRAPHIC mode you selected (0 - 3)

RJOY(<joystick>)

Argument	Description
<joystick>	Which joystick's position to read 1 — Read position of joystick attached to Control Port 1 (0 - 8) 2 — Read position of joystick attached to Control Port 2 (0 - 8)

(When the "Fire" button is pressed, 128 is added to position value)

RPEN(<data>)

Argument	Description
<data>	To return X or Y coordinate of lightpen 0 — X coordinate 1 — Y coordinate

RPOT(<paddle>)

Argument	Description
<paddle>	Which of four paddles to read (0 - 3); A value returned (0 - 255) indicates the position of paddle control knob 0 — Position of paddle 1 1 — Position of paddle 2 2 — Position of paddle 3 3 — Position of paddle 4

(When "Fire" button is pressed, 256 is added to position value)

RSPCOL(<register>)

Argument	Description
<register>	Sprite color information (0 - 1) 0 — Sprite Multi-Color1 color code (0 - 15) 1 — Sprite Multi-Color2 color code (0 - 15)

RSPPOS(<sprite>,<data>)

Argument	Description
<sprite>	Sprite number (0 - 7)
<data>	Information about the sprite 0 — Current X position 1 — Current Y position 2 — Current speed

RSPR(<sprite>,<field>)

Argument	Description
<sprite>	Sprite number (0 - 7)
<field>	A value (0 - 5) that specifies which < field > characteristic you are checking 0 — Sprite Display; 0 - OFF; 1 - ON 1 — Sprite Foreground color (0 - 15) 2 — Display Priority; 0 - High, 1 - Low 3 — Sprite Expanded in X; 0 - No; 1 - Yes 4 — Sprite Expanded in Y; 0 - No; 1 - Yes 5 — Sprite Display mode; 0 - HiRes; 1 - Multi-Color

SCALE < n >

Parameter	Description
<n>	Scale toggle 0 — Standard coordinate system, boundaries depend on GRAPHIC mode being used 1 — SUPER EXPANDER 64 coordinate system, boundaries range from 0 through 1023 for any mode

SCNCLR

Parameter	Description
None	To clear the screen in any mode

SPRCOL [smcr-1] [,smcr-2]

Parameters	**Description**
[smcr-1]	Sprite Multi-Color 1 color (0 - 15)
[,smcr-2]	Sprite Multi-Color 2 color (0 - 15)

(Defaults do not change color)

SPRDEF

User Input	Description
0 - 7	Selects destination sprite (prompted)
A	Automatic cursor movement toggle
CRSR keys	Moves cursor
RETURN key	Moves cursor to start of next line
RETURN key	Exits Sprite Designer mode (prompted)
HOME key	Moves cursor to top left of grid
CLR key	Erases entire grid
1 - 4	Selects color source
<CTRL> 1 - 8	Selects sprite Foreground color (0 - 7)
C= 1 - 8	Selects sprite Foreground color (8 - 15)
STOP key	Cancels changes and returns to prompt
SHIFT RETURN	Saves sprite and returns to prompt
X	Expands sprite in X toggle
Y	Expands sprite in Y toggle
M	Multi-Color sprite toggle

SPRITE < number > [,[on/off] [,[fgnd] [,[priority] [,[X-exp] [,[Y-exp] [,mode]]]]]]

Parameter	Description
< number >	Sprite number (0 - 7)
[on/off]	Sprite enabled (1), sprite disabled (0)
[fgnd]	Sprite Foreground color (0 - 15)
[priority]	Sprite priority (0 - 1) 0 — Sprite priority over screen data 1 — Screen data priority over sprite
[X-exp]	Sprite expansion in X direction 0 — X-expansion OFF 1 — X-expansion ON
[Y-exp]	Sprite expansion in Y direction 0 — Y-expansion OFF 1 — Y-expansion ON
[,mode]	Sprite mode 0 — Sprite displayed as High Resolution 1 — Sprite displayed as Multi-Color

(Defaults will not change the current parameters)

SPRSAV < origin> , < destination >

Parameter	Description
< origin >	Sprite number (0 - 7) or string name
< destination >	String name or sprite number

SSHAPE < stringname > , < X1,Y1> [,X2,Y2]

Parameter	Description
< stringname >	Variable to save shape
< X1,Y1 >	Locates the diagonally opposite corner of rectangular area to be saved
[,X2,Y2]	Starting point for saving shape; default is current PC

TEMPO < speed >

Argument	Description
< speed >	The relative duration of notes (0 - 255) (Default is 8)

TUNE < env > ,[,[atk] [,[dec] [,[sus] [,[res][,[form][,width]]]]]]

Argument	Description
< env >	Envelope number (0 - 9)
[atk]	Attack rate (0 - 15)
[dec]	Decay rate (0 - 15)
[sus]	Sustain rate (0 - 15)
[rel]	Release rate (0 - 15)
[form]	Waveform of sound (0 - 4) 0 — Triangle 1 — Sawtooth 2 — Pulse 3 — Random Noise 4 — Ring Modulation
[width]	Has meaning only when [form] is 2 for Pulse waveform; (0 - 4095)

Appendix III SAMPLE PROGRAMS

The following programs will give you practice in using the SUPER EXPANDER 64 commands. Several of these programs correspond to photographs on the back cover of this manual.

Listing 1: Commodore Logo

```
10 COLOR 3,6,,,4: GRAPHIC 2,1: POKE 49168,1: SCALE 0
20 CIRCLE 1,158,102,60,52,160,19,,1
30 CIRCLE 1,160,102,36,32,150,30,,1
40 DRAW 1,177,54 TO 177,74: DRAW 1,178,130 TO 178,151
50 COLOR ,2: DRAW 1,182,122 TO 182,104 TO 202,104 TO 226,122 TO
   182,122
60 COLOR ,1: DRAW 1,182,81 TO 182,99 TO 202,99 TO 226,81 TO
   182,81
70 POKE 49168,0: COLOR ,6: PAINT ,102,102
80 COLOR ,2: PAINT ,184,118: COLOR ,1: PAINT ,184,90
90 GOTO 90
```

Listing 2: Split Screen Exercise

```
10 COLOR 12,11,,,4: GRAPHIC 3,1
20 SCALE 0: POKE 49168,0: Q$ = CHR$(34):R$ = CHR$(18):
   V$ = CHR$(146)
30 PRINT “see note below”;
40 PRINT “       THIS IS MY “R$” SPLIT-SCREEN “V$” MODE.”
50 PRINT “        YOU CAN PRINT 5 LINES OF”
60 PRINT “      UPPER-CASE TEXT AT THE BOTTOM.”
70 PRINT “    YOU CAN ALSO USE MY “R$” CHAR “V$”
   STATEMENT”
80 PRINT “    TO PUT TEXT ANYPLACE ON THE SCREEN.”;
90 COLOR ,1: CHAR 1,2,1,“YOU CAN MAKE SIMPLE LINES”
100 CHAR 1,2,3,“OR DRAW COMPLEX FIGURES”
110 CHAR 1,2,5,“WITH A SINGLE STATEMENT”
120 COLOR ,0: LOCATE 100,150: DRAW 1
130 DRAW TO 260,60: FOR J = 1 TO 6
140 GOSUB 190: XA = X: YA = Y
150 GOSUB 190: XB = X: YB = Y
160 DRAW 1,XA,YA TO XB,YB: NEXT J
170 DRAW 1, 20,100 TO 80,100 TO 30,130 TO 50,80 TO 70,130 TO
    20,100
180 GOTO180
190 X = INT(RND(0)*300): IF X < 96 THEN 190
200 Y = INT(RND(0)*150): IF Y < 60 THEN 200
210 RETURN
```

Note: To produce the reverse characters which should appear in quotes on Line 30, press the SHIFT and CLR/HOME key followed by 20 CRSR down keys.

Listing 3: Polygons

```
10 COLOR 15,14,,,3: GRAPHIC 2,1: SCALE 0: POKE 49168,1
20 BOX 1,318,158,0,0: BOX 1,316,156,2,2: BOX 1,314,154,4,4
30 COLOR ,11: CHAR 1,2,20,“YOU CAN USE MY ‘BOX’ COMMAND
   TO DRAW”
40 CHAR 1,4,22,“RECTANGLES OF ANY SIZE OR SHAPE—”
50 CHAR 1,3,24,“AND ROTATE OR FILL THEM WITH COLOR.”
60 POKE 49168,0: COLOR ,1: BOX 1,12,12,64,32,0,1
70 COLOR ,7: BOX 1,108,48,,,1
80 COLOR ,11: BOX 1,120,16,300,48
90 XA=12: XB=28: YA=46: YB=136: POKE 49168,1
100 FOR J=1 TO 8: COLOR ,J*2-2+(J>5)
110 BOX 1,XA,YA,XB,YB,0,1: XA=XA+24
120 XB=XB+24: YA=YA+10: YB=YB-10: NEXTJ
130 COLOR ,12: BOX ,128,112,160,144,45: BOX ,128,112,160,144
140 COLOR ,5: BOX 1,128,24,196,40,,1
150 COLOR ,0: BOX ,224,16,256,48,45
160 XA=240: YA=60: XB=261: YB=140: D=30: N=150: COLOR ,0
170 POKE 49168,0: FOR Z=0 TO N STEP D
180 BOX ,XA,YA,XB,YB,Z: NEXT Z
190 GOTO190
```

Listing 4: Arcs, Circles, and Ellipses

```
10 COLOR 7,9,,,14: GRAPHIC 2,1: SCALE 0: POKE 49168,1
20 CHAR 1,1,1,“MY ‘CIRCLE’ COMMAND LETS YOU EASILY”
30 CHAR ,1,3,“DRAW ARCS, CIRCLES,”
40 DRAW ,20,80 TO 100,80: DRAW ,60,80 TO 60,40
50 R=40: CIRCLE ,60,50,R,R,130,230
60 CIRCLE ,40,80,R,R,0,110
70 CIRCLE ,80,80,R,R,250,0
80 XA=160: YA=68: FOR R=6 TO 30 STEP 6
90 CIRCLE ,XA,YA,R*1.35,R: NEXT R
100 CHAR 1,21,3,“ROTATED ELLIPSES —”
110 POKE 49168,0: FOR R=0 TO 150 STEP 30
120 CIRCLE 1,260,72,32,12,,,R: NEXT R
130 CHAR ,1,13,“OR ANY REGULAR POLYGON”
140 POKE 49168,1: XA= -20: YA=140: D=360
150 FOR J=3 TO 7: XA=XA+54+J: N=J: IF J=7 THEN N=8
160 IF J=7 THEN N=8: YA=YA-4
170 CIRCLE ,XA,YA+(J AND 1)*32,30,25,,,,D/N: NEXT J
180 GOTO 180
```

Listing 5: Saving Shapes and Getting Shapes

```
10 COLOR 1,11,,,5: GRAPHIC 2,1: SCALE 1: POKE 49168,0
20 CHAR 1,3,1,“MY ‘SSHAPE’ COMMAND FILLS A STRING”
30 CHAR 1,5,2,“FROM A RECTANGLE-SHAPED AREA OF”
40 CHAR 1,7,3,“YOUR BIT-MAP SCREEN DISPLAY.”
50 X=96: Y=192: W=64: L=248: N=135: D=45: G=192: H=128
60 FOR K=0 TO 4: COLOR ,K*3
70 IF K=1 THEN: COLOR,2: POKE 49168,1
80 FOR J=0 TO N STEP D
90 BOX 1,X,Y,X+W,Y+L,J: NEXT J
100 SSHAPE A$(K),X,Y,X+W,Y+L
110 X=X+G: Y=Y+H: NEXT K: X=96: Y=124
120 COLOR ,11: CHAR 1,13,5,“THEN ‘GSHAPE’ PLACES THE”
130 CHAR 1,15,6,“STRING ANYWHERE ON THE”
140 CHAR 1,16,7,“SCREEN — IN FIVE WAYS.”
150 FOR K=0 TO 4: COLOR ,K*3
160 IF K=0 THEN: CHAR 1,1,11,“AS IS”
170 IF K=1 THEN: COLOR ,2: CHAR 1,3,14,“INVERTED”
180 IF K=2 THEN: CHAR 1,2,17,“‘OR’ WITH SCREEN”
190 IF K=3 THEN: CHAR 1,8,20,“‘AND’ WITH SCREEN”
200 IF K=4 THEN: CHAR 1,15,23,“‘XOR’ WITH SCREEN”
210 GSHAPE A$(K),X,Y,K: X=X+G: Y=Y+H: NEXT K
220 GOTO 220
```

Listing 6: Drawing and Painting Shapes

```
10 COLOR 13,6,2,4,10: GRAPHIC 1,1
20 SCALE 0: POKE 49168,0
30 XA=30: YA=60: D=360: FOR J=3 TO 5
40 CIRCLE 1,XA,YA,20,32,,,,D/J
50 XA=XA+45: NEXT J: YA=130: XA=30
60 FOR J=10 TO 6 STEP -2: CIRCLE 1,XA,YA,20,32,,,,D/J
70 XA=XA+45: NEXT J
80 CIRCLE ,75,90,60,40: PAINT 1,75,94
90 PAINT 2,30,60,1: PAINT 3,75,40,1: PAINT 2,120,60,1
100 PAINT 3,30,130,1: PAINT 2,75,140,1: PAINT 3,120,130,1
110 PAINT 3,40,75,1: PAINT 2,75,60,1: PAINT 3,110,80,1
120 PAINT 2,40,120,1: PAINT 3,75,120,1: PAINT 2,110,120,1
130 GOTO 130
```

Listing 7: Floating Sprites

```
10 COLOR 1,1,,,13: GRAPHIC 2,1: POKE 49168,1: SCALE 0
20 CHAR 1,1,17,“I HAVE SIX COMMANDS + FOUR FUNCTIONS”
30 CHAR ,1,19,“SPECIALLY DESIGNED TO HELP YOU EASILY”
40 CHAR ,1,21,“CREATE, ANIMATE AND CONTROL SPRITES.”
50 CHAR 1,0,0,“                  ”,1
60 CHAR 1,0,1,“ 1 2 3 4 5 6 7 8 ”,1
70 CHAR 1,0,2,“                  ”,1: Q=24
80 X=0: Y=2: FOR J=0 TO 7: SPRITE J,0
90 SSHAPE A$(J),X,Y,X+23,Y+20
100 SPRSAV A$(J),J: MOVSPR J,X+Q,Y+50
110 SPRITE J,1,J*4,1: X=X+Q: NEXT J
120 FOR Z=1 TO 64: FOR J=0 TO 7
130 MOVSPR J,1;150: NEXT J,Z
140 FOR Z=0 TO 499: NEXT Z
150 FOR J=0 TO 7: A=10*(INT(RND(0)*36))
160 SPRITE J,,,J AND 1: MOVSPR J,A#4: NEXT J
170 GOTO 170
```

Listing 8: Joystick Demonstration

This program lets you create High-Resolution designs using a Joystick plugged into Control Port 1. Touch the “FIRE” button to draw or erase lines. Hold down the “FIRE” button to draw with dotted lines.

```
10 GOTO100
20 Y = Y – K:RETURN
30 X = X + K:Y = Y – K:RETURN
40 X = X + K:RETURN
50 X = X + K:Y = Y + K:RETURN
60 Y = Y + K:RETURN
70 X = X – K:Y = Y + K:RETURN
80 X = X – K:RETURN
90 X = X – K:Y = Y – K:RETURN
100 COLOR14,6,,,4:GRAPHIC0,1
110 PRINT CHR$(147)“   WHEN IT COMES TO GAMES I GET
    SERIOUS”
120 PRINT“    MY /RJOY(N)/ FUNCTION LETS YOU EASILY”
130 PRINT“    READ JOYSTICKS FROM MY GAME PORTS.”
140 PRINT“    THE JOYSTICK DIRECTIONS ARE READ AS:”
150 PRINT“  ”
160 PRINT“     WHEN ‘FIRE’ BUTTON            1
170 PRINT“     IS PRESSED, ADD            8     2
180 PRINT“     128 TO THESE            7     0     3
190 PRINT“     DIRECTION VALUES           6     4
200 PRINT“                                   5
210 PRINT“   HIT SPACE BAR FOR JOYSTICK DEMONSTRATION”
220 PRINT“    PRESS ANY OTHER KEY TO EXIT”
230 GETA$:IFA$ = “ ”THEN230
240 COLOR1,12,,,9:PRINT CHR$(147):IFA$< > “ ”THEN340
250 GRAPHIC2,1:SCALE0:POKE49168,1
260 X = 160:Y = 100:K = 2:C = 1:DRAWC,X,Y
280 J = RJOY(1):IFJ = > 128THEN320
290 ON J GOSUB 20,30,40,50,60,70,80,90
300 IF J THEN:DRAW C TO X,Y
310 GOTO280
320 C = 1 – C:FORN = 0TO49:NEXT
330 J = J – 128:DRAWC,X,Y:GOTO290
340 GRAPHIC0,1:END
```

Note: In lines 110 through 210 inclusive, insert a reverse character (in the space provided after the beginning quote) by pressing SHIFT and the CRSR down key. Place two of these characters in the quotes at line 150.

Listing 9: SUPER EXPANDER 64 Pong

This program illustrates the use of SUPER EXPANDER 64 in developing graphics for games and recreation. In our example, we use Paddle 1 in Control Port 1.

```
0 GRAPHIC1,1:SCALE 0:COLOR14,6,2,4,11
1 CIRCLE2,100,100,4,6:PAINT2,100,100,1
2 SSHAPEA$,95,90,106,110:SPRSAVA$,1:SCNCLR
3 BOX2,95,90,106,106:PAINT2,100,91,1
4 SSHAPEB$,95,90,106,110:SPRSAVB$,3:SCNCLR
20 SCALE1:CIRCLE2,300,300,50,50,,,:PAINT2,300,300,1
30 BOX3,750,345,650,245,45:PAINT3,700,300,1
40 BOX,345,750,245,650,,1
50 DRAW2,650,650TO750,650TO700,750TO650,650:PAINT2,700,675,1
55 S=8:XX=S:YY=S:X=156:Y=100:C=4
60 SPRITE1,1,13,,,,1:SCALE0
70 X=X+XX:Y=Y+YY:MOVSPR1,X,Y
73 IFC<4THEN76
74 C=4
75 COLINT1,200:COLINTO,300
76 C=C+1
80 IFY=> 280THEN700
83 IFY=< 50THENYY= -YY
85 IFX=< 24THENXX= -XX
87 IFX=> 320THENXX= -XX
120 J=RPOT(1):SPRITE3,1,0, ,1,,1
130 MOVSPR3,287-J,230:GOTO70
200 COLINT1:R=INT(RND(0)*3):C=0
210 IFR=0THEN:XX= -XX:YY= -YY:MOVSPR1,X+XX,Y+YY:
                      RETURN
211 IFR=1THEN:XX= -XX:MOVSPR1,X+XX,Y+YY:RETURN
212 IFR=2THEN:YY= -YY:MOVSPR1,X+XX,Y+YY:RETURN
300 COLINTO:YY= -YY:MOVSPR1,X+XX,Y+YY:C=0:RETURN
700 GRAPHIC2:CHAR1,13,1,“ YOU MISSED!  ”:FORN=1
    TO1000:NEXT
710 CHAR0,13,1“                 ”:GRAPHIC1:GOTO55
```

Listing 10: Moire Pattern

```
10 COLOR 0,1: GRAPHIC 2,1: SCALE 0
20 X=320: Y=200: C=X*RND(0): D=Y*RND(0)
30 A=0: B=0: J=0: K=1: N=2: P=1
40 DRAW K,A,J TO C,D, TO A,Y
50 DRAW J,A+K,J TO C,D TO A+K,Y
60 A=A+N: IF A<X THEN 40
70 DRAW K,J,B TO C,D TO X,B
80 DRAW J,J,B+K TO C,D TO X,B+K
90 B=B+N: IF B<Y THEN 70
100 GOTO 100
```

Listing 11: Hypnotic Pattern

```
10 GRAPHIC1,1: XC=79: YC=99: Z=0: P=3.14: K=P/60
20 FOR N=0 TO 4: C(N)=INT(RND(1)*16): NEXT N
30 COLOR C(0),C(1),C(2),C(3),C(4)
40 FOR J=0 TO 1999: NEXT: N=K
50 XD=INT(RND(1)*XC): YD=INT(RND(1)*YC): C=INT(RND(1)*5)
60 XA=XC+XD*COS(N): XB=XC-XD*COS(N)
70 YA=YC+YD*SIN(N): YB=YC-YD*SIN(N)
80 DRAW C,XA,YA TO XB,YB
90 N=N+K: IF N<P THEN 60
100 FOR J=0 TO 1999: NEXT: GOTO 20
```

Note: For variations on the Hypnotic Pattern, try changing the divisor for variable P in Line 10. Also, in Line 20, you can change the range of colors to be selected by the random function (RND).

INDEX

commodore
COMPUTER

Commodore Business Machines, Inc.
1200 Wilson Drive • West Chester, PA 19380

Commodore Business Machines, Limited
3370 Pharmacy Avenue • Agincourt, Ontario, M1W 2K4

Printed in Hong Kong