Radio Shack®

"The biggest name in little computers"

TRS-80®
Computer Graphics

CUSTOM MANUFACTURED IN THE USA BY RADIO SHACK, A DIVISION OF TANDY CORPORATION

A Special Note on Model III Computer Graphics...

Be sure to use the GCLS command ("clear the Graphics Screen") at TRSDOS READY when you first turn on your computer. Otherwise, random graphics may appear on the Screen.

Thank You

Radio Shack
A DIVISION OF TANDY CORPORATION
FORT WORTH, TEXAS 76102

8759223

TERMS AND CONDITIONS OF SALE AND LICENSE OF RADIO SHACK COMPUTER EQUIPMENT AND SOFTWARE
PURCHASED FROM A RADIO SHACK COMPANY-OWNED COMPUTER CENTER, RETAIL STORE OR FROM A
RADIO SHACK FRANCHISEE OR DEALER AT ITS AUTHORIZED LOCATION

LIMITED WARRANTY

I. CUSTOMER OBLIGATIONS

A. CUSTOMER assumes full responsibility that this Radio Shack computer hardware purchased (the "Equipment"), and any copies of Radio Shack software included with the Equipment or licensed separately (the "Software") meets the specifications, capacity, capabilities, versatility, and other requirements of CUSTOMER.

B. CUSTOMER assumes full responsibility for the condition and effectiveness of the operating environment in which the Equipment and Software are to function, and for its installation.

II. RADIO SHACK LIMITED WARRANTIES AND CONDITIONS OF SALE

A. For a period of ninety (90) calendar days from the date of the Radio Shack sales document received upon purchase of the Equipment, RADIO SHACK warrants to the original CUSTOMER that the Equipment and the medium upon which the Software is stored is free from manufacturing defects. THIS WARRANTY IS ONLY APPLICABLE TO PURCHASES OF RADIO SHACK EQUIPMENT BY THE ORIGINAL CUSTOMER FROM RADIO SHACK COMPANY-OWNED COMPUTER CENTERS, RETAIL STORES AND FROM RADIO SHACK FRANCHISEES AND DEALERS AT ITS AUTHORIZED LOCATION. The warranty is void if the Equipment's case or cabinet has been opened, or if the Equipment or Software has been subjected to improper or abnormal use. If a manufacturing defect is discovered during the stated warranty period, the defective Equipment must be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer for repair, along with a copy of the sales document or lease agreement. The original CUSTOMER'S sole and exclusive remedy in the event of a defect is limited to the correction of the defect by repair, replacement, or refund of the purchase price, at RADIO SHACK'S election and sole expense. RADIO SHACK has no obligation to replace or repair expendable items.

B. RADIO SHACK makes no warranty as to the design, capability, capacity, or suitability for use of the Software, except as provided in this paragraph. Software is licensed on an "AS IS" basis, without warranty. The original CUSTOMER'S exclusive remedy, in the event of a Software manufacturing defect, is its repair or replacement within thirty (30) calendar days of the date of the Radio Shack sales document received upon license of the Software. The defective Software shall be returned to a Radio Shack Computer Center, a Radio Shack retail store, participating Radio Shack franchisee or Radio Shack dealer along with the sales document.

C. Except as provided herein no employee, agent, franchisee, dealer or other person is authorized to give any warranties of any nature on behalf of RADIO SHACK.

D. Except as provided herein, **RADIO SHACK MAKES NO WARRANTIES, INCLUDING WARRANTIES OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE.**

E. Some states do not allow limitations on how long an implied warranty lasts, so the above limitation(s) may not apply to CUSTOMER.

III. LIMITATION OF LIABILITY

A. EXCEPT AS PROVIDED HEREIN, RADIO SHACK SHALL HAVE NO LIABILITY OR RESPONSIBILITY TO CUSTOMER OR ANY OTHER PERSON OR ENTITY WITH RESPECT TO ANY LIABILITY, LOSS OR DAMAGE CAUSED OR ALLEGED TO BE CAUSED DIRECTLY OR INDIRECTLY BY "EQUIPMENT" OR "SOFTWARE" SOLD, LEASED, LICENSED OR FURNISHED BY RADIO SHACK, INCLUDING, BUT NOT LIMITED TO, ANY INTERRUPTION OF SERVICE, LOSS OF BUSINESS OR ANTICIPATORY PROFITS OR CONSEQUENTIAL DAMAGES RESULTING FROM THE USE OR OPERATION OF THE "EQUIPMENT" OR "SOFTWARE". IN NO EVENT SHALL RADIO SHACK BE LIABLE FOR LOSS OF PROFITS, OR ANY INDIRECT, SPECIAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF ANY BREACH OF THIS WARRANTY OR IN ANY MANNER ARISING OUT OF OR CONNECTED WITH THE SALE, LEASE, LICENSE, USE OR ANTICIPATED USE OF THE "EQUIPMENT" OR "SOFTWARE".

NOTWITHSTANDING THE ABOVE LIMITATIONS AND WARRANTIES, RADIO SHACK'S LIABILITY HEREUNDER FOR DAMAGES INCURRED BY CUSTOMER OR OTHERS SHALL NOT EXCEED THE AMOUNT PAID BY CUSTOMER FOR THE PARTICULAR "EQUIPMENT" OR "SOFTWARE" INVOLVED.

B. RADIO SHACK shall not be liable for any damages caused by delay in delivering or furnishing Equipment and/or Software.

C. No action arising out of any claimed breach of this Warranty or transactions under this Warranty may be brought more than two (2) years after the cause of action has accrued or more than four (4) years after the date of the Radio Shack sales document for the Equipment or Software, whichever first occurs.

D. Some states do not allow the limitation or exclusion of incidental or consequential damages, so the above limitation(s) or exclusion(s) may not apply to CUSTOMER.

IV. RADIO SHACK SOFTWARE LICENSE

RADIO SHACK grants to CUSTOMER a non-exclusive, paid-up license to use the RADIO SHACK Software on **one** computer, subject to the following provisions:

A. Except as otherwise provided in this Software License, applicable copyright laws shall apply to the Software.

B. Title to the medium on which the Software is recorded (cassette and/or diskette) or stored (ROM) is transferred to CUSTOMER, but not title to the Software.

C. CUSTOMER may use Software on one host computer and access that Software through one or more terminals if the Software permits this function.

D. CUSTOMER shall not use, make, manufacture, or reproduce copies of Software except for use on **one** computer and as is specifically provided in this Software License. Customer is expressly prohibited from disassembling the Software.

E. CUSTOMER is permitted to make additional copies of the Software **only** for backup or archival purposes or if additional copies are required in the operation of **one** computer with the Software, but only to the extent the Software allows a backup copy to be made. However, for TRSDOS Software, CUSTOMER is permitted to make a limited number of additional copies for CUSTOMER'S own use.

F. CUSTOMER may resell or distribute unmodified copies of the Software provided CUSTOMER has purchased one copy of the Software for each one sold or distributed. The provisions of this Software License shall also be applicable to third parties receiving copies of the Software from CUSTOMER.

G. All copyright notices shall be retained on all copies of the Software.

V. APPLICABILITY OF WARRANTY

A. The terms and conditions of this Warranty are applicable as between RADIO SHACK and CUSTOMER to either a sale of the Equipment and/or Software License to CUSTOMER or to a transaction whereby RADIO SHACK sells or conveys such Equipment to a third party for lease to CUSTOMER.

B. The limitations of liability and Warranty provisions herein shall inure to the benefit of RADIO SHACK, the author, owner and/or licensor of the Software and any manufacturer of the Equipment sold by RADIO SHACK.

VI. STATE LAW RIGHTS

The warranties granted herein give the **original** CUSTOMER specific legal rights, and the **original** CUSTOMER may have other rights which vary from state to state.

Contents

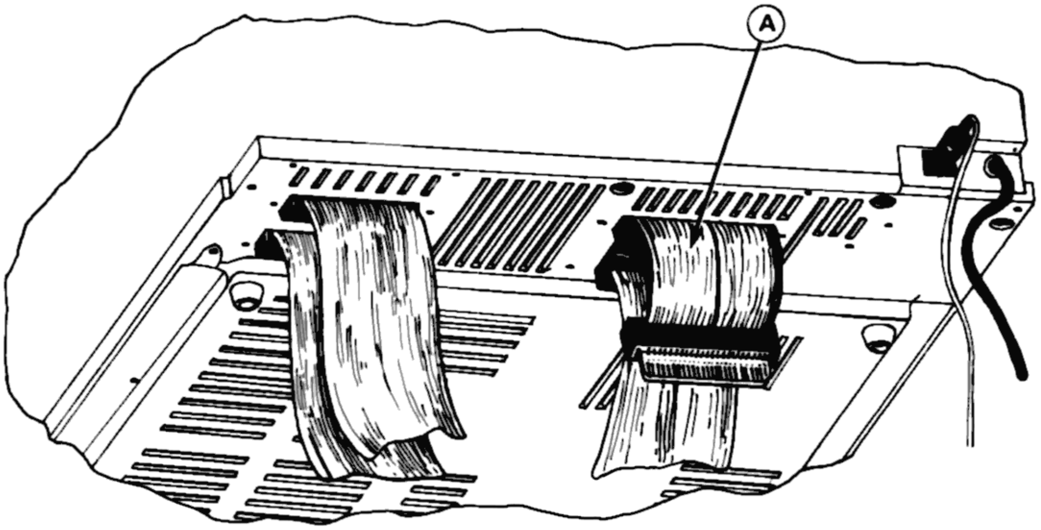
To Our Customers . . .

The TRS-80® Computer Graphics package revolutionizes your Model III by letting you draw intricate displays from simple program instructions. With the highly-defined Graphics Screen, the list of practical applications is nearly endless!

The TRS-80 Computer Graphics package includes a:

- Computer Graphics Diskette
- Computer Graphics Operation Manual

However, before you can use this package, your Model III must have 48K of RAM (Random Access Memory) and one disk drive. Your computer must also be modified by a qualified Radio Shack service technician. The only difference you'll notice is a cable which protrudes from the bottom of the Model III case. Do not attempt to disconnect this cable! This cable is provided to allow you to attach peripheral devices (such as a hard disk) to the I/O Bus Jack of the Model III. The cable connector which is attached directly to the I/O Bus Jack (see Point A in the figure below) must be firmly attached for the Computer Graphics package to work.



Included on the Graphics diskette are:

- TRSDOS 1.3
- Disk BASIC
- Graphics BASIC (BASICG)
- Graphics Subroutine Library (GRPLIB)

- Graphics Utilities
- Sample Programs in BASICG and FORTRAN.

To print graphic displays, you can use any Radio Shack printer that has graphic capabilities such as Line Printer VII (26-1167), Line Printer VIII (26-1168), DMP-100 (26-1253), DMP-200 (26-1254), DMP-400 (26-1251), or DMP-500 (26-1252).

You can also utilize the Graphics Subroutine Library with several languages, including, but not limited to FORTRAN (26-2200).

About This Manual . . .

For your convenience, we've divided this manual into five sections plus appendixes:

- Computer Graphics Overview
- Graphics BASIC (BASICG) Language Description
- Graphics Utilities
- FORTRAN Description
- Programming the Graphics Board
- Appendixes

This package contains two separate (but similar) methods for Graphics programming:

- Graphics BASIC (BASICG)
- Graphics Subroutine Library

If you're familiar with Model III TRSDOS™ and BASIC, you should have little trouble in adapting to Graphics BASIC. If you want to review BASIC statements and syntax, see your **Model III Operation and BASIC Language Reference Manual** and **Model III Disk System Owner's Manual**. Then read Chapters 1, 2 and 3, along with Appendixes A, B, E, and F of this manual.

If it's Graphics applications in FORTRAN you're after, refer to the TRS-80 FORTRAN manual. Then read Chapters 1, 2, 3, and 4 as well as Appendixes C, D, E, and F of this manual.

Note: This manual is written as a reference manual for the TRS-80 Computer Graphics package. It is not intended as a teaching guide for graphics programming.

Notational Conventions

The following conventions are used to show syntax in this manual:

CAPITALS	Any words or characters which are uppercase must be typed in exactly as they appear.
lowercase underline	Fields shown in lowercase underline are variable information that you must substitute a value for.
<ENTER>	Any word or character contained within brackets represents a keyboard key to be pressed.
...	Ellipses indicate that a field entry may be repeated.
filespec	A field shown as filespec indicates a standard TRSDOS file specification of the form: filename/ext.password:d Note that with TRSDOS 1.3, d (Drive) can be any number from 0-3.
punctuation	Punctuation other than ellipses must be entered as shown.
delimiters	Commands must be separated from their operands by one or more blank spaces. Multiple operands, where allowed, may be separated from each other by a comma, a comma followed by one or more blanks, or by one or more blanks. Blanks and commas may not appear within an operand.

1/ Computer Graphics Overview

Graphics is the presentation of dimensional artwork. With TRS-80 Computer Graphics, the artwork is displayed on a two-dimensional plane -- your computer screen. Like an artist's easel or a teacher's blackboard, the screen is a "drawing board" for your displays.

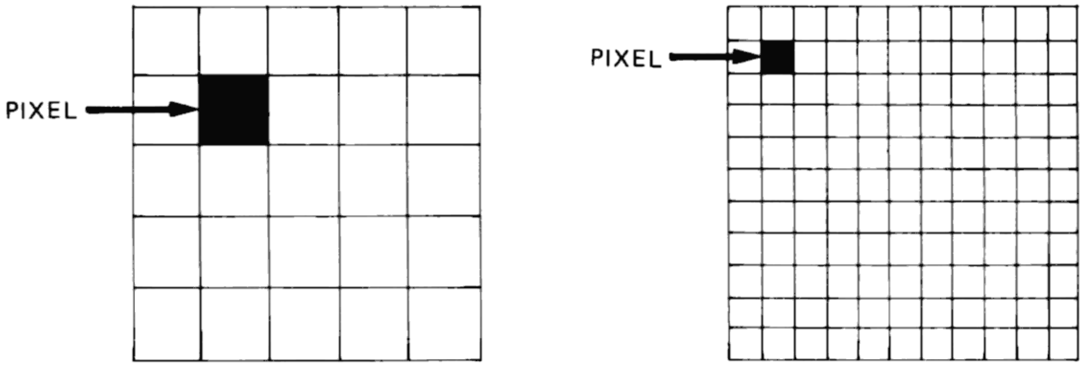
TRS-80 Computer Graphics has two colors:

- Black (OFF)
- White (ON)

Graphics programming is different from other types of programming because your ultimate result is a pictorial display (bar graph, pie chart, etc.) rather than textual display (sum, equation, etc.). This is an important distinction. After working with graphics for a while, you'll find yourself thinking "visually" as you write programs.

In computer-generated graphics, displays can include tables, charts, graphs, illustrations and other types of artwork. Once they're created, you can "paint" displays with a variety of styles and shapes, or even simulate animation.

The Computer Graphics program uses a "high-resolution" screen. The more addressable points or dots (called "pixels") on a computer's screen, the higher the resolution. A lower resolution screen has fewer addressable pixels.



Lower resolution

Higher resolution

Figure 1. Resolution

Since the TRS-80 has high-resolution -- 640 pixels on the X-axis (0 to 639) and 240 pixels on the Y-axis (0 to 239) -- you can draw displays that have excellent clarity and detail.

How TRS-80 Computer Graphics Works

The concept of graphics is fairly simple. Each point on the screen can be turned ON (white) or OFF (black).

When you clear the Graphics Screen, all graphic points are turned OFF.

Therefore, by setting various combinations of the pixels (usually with a single command) either ON or OFF, you can generate lines, circles, geometric figures, pictures, etc.

The Graphics Subroutine Library, which is part of the Computer Graphics package, contains subroutines which provide the same capabilities, as well as similar names and parameters, as the commands and functions in Graphics BASIC. The main difference between the Subroutine Library and BASICG is the manner in which coordinates are specified (e.g., BASICG coordinates are specified as arguments for each command while the Subroutine Library specifies coordinates with a separate subroutine call). Another difference concerns the names of a few routines (e.g., LINE vs. LINEB vs. LINEBF, etc.). All of these differences will be described in detail in the appropriate sections of this manual.

The Graphics Screen

TRS-80 Computer Graphics has two "screens" -- Text and Graphics. (We'll call them screens, although they are really modes.) Both screens can act independently of each other and make use of the computer's entire display area.

The Text Screen, also referred to as the "Video Display," is the "normal" screen where you type in your programs. The Graphics Screen is where graphic results are displayed. Both screens can be cleared independently. Note: The Graphics Screen will not automatically be cleared when you return to TRSDOS. It will be cleared when you re-enter BASICG.

The Graphics Screen cannot be displayed at the same time as the Text Screen.

While working with Computer Graphics, it might be helpful to imagine the screen as a large Cartesian coordinate plane (with a horizontal X- and a vertical Y-axis). However, unlike some coordinate systems, TRS-80 Computer Graphics' coordinate numbering starts in the upper-left corner -- (0,0) -- and increases toward the lower-right corner -- (639,239). The lower-left corner is (0,239) and the upper-right corner is (639,0).

Since the screen is divided into X-Y coordinates (like the Cartesian system), each pixel is defined as a unique position. In TRS-80 Computer Graphics, you can directly reference these coordinates as you draw.

About Ranges...

Some TRS-80 Computer Graphics commands accept values within the Model III integer range (-32768 to 32767), instead of just 0 to 639 for X and 0 to 239 for Y. Since most of the points in the integer range are off the screen, these points are part of what is called Graphics "imaginary" Cartesian system.

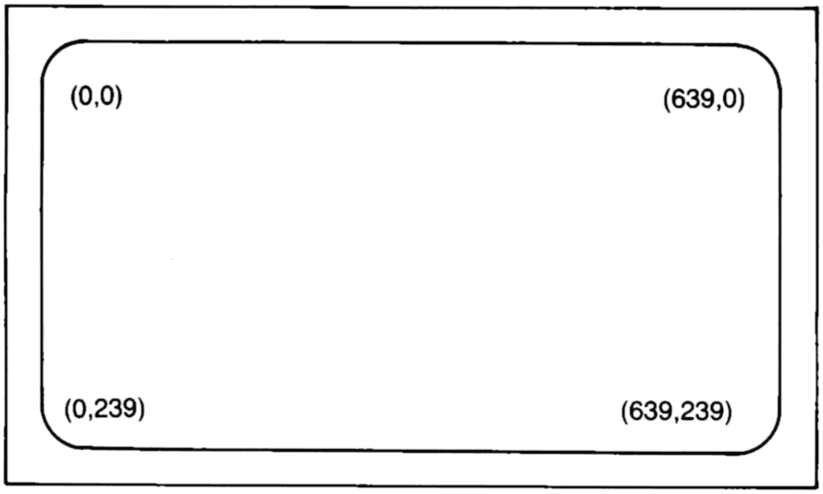


Figure 2. Graphics Visible Screen

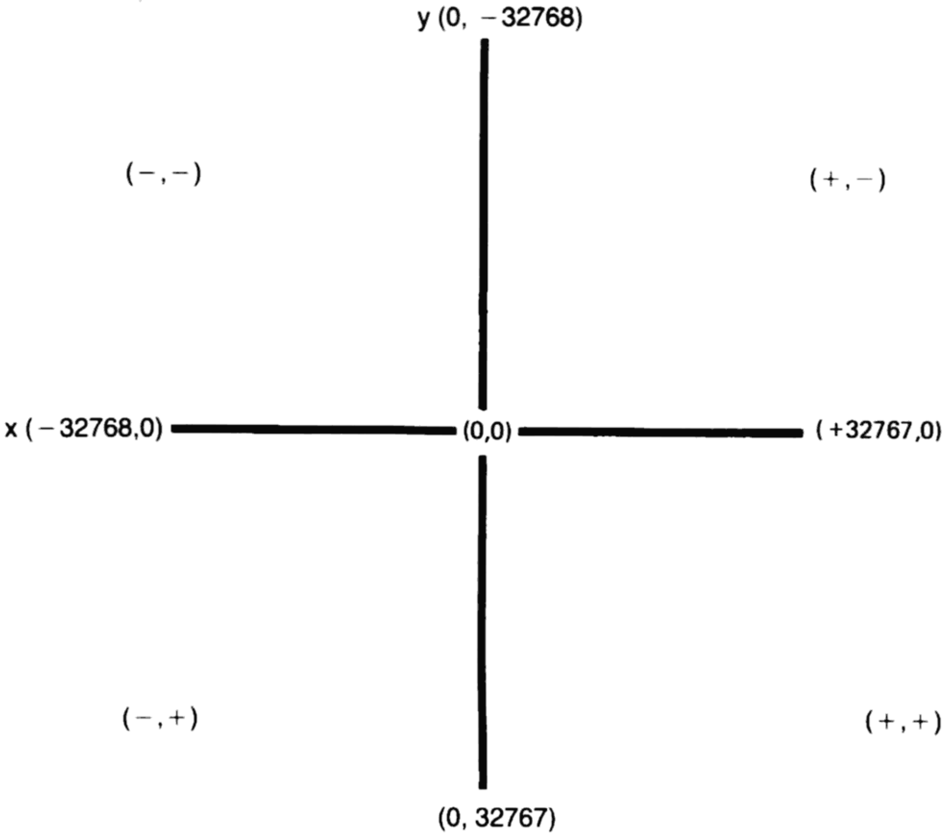


Figure 3. Graphics "Imaginary" Cartesian System

2/ Graphics BASIC

Graphics BASIC (BASICG) vs. BASIC

The Graphics BASIC file on the supplied diskette is named BASICG.

Program files created under BASICG are not directly loadable with BASIC files (and vice versa). If you attempt to load a BASIC file in compressed format from BASICG (and vice versa), an NB error may occur. See Appendix B for a list of BASICG error messages.

If you want to load a file from one BASIC to the other to the other, we recommend that you first save the file in ASCII format (**SAVE"<u>filename/ext</u>",A**).

You can then load and run a BASIC file from either BASICG or BASIC. You cannot run programs that contain BASICG statements while in BASIC.

Important Note: Because of memory limitations, some programs (i.e., some application programs) will not run in BASICG. BASICG uses approximately 6.5K more memory than BASIC. When you enter BASIC with 0 files, there are 39,282 bytes free. When you enter BASICG with 0 files, there are 32,675 bytes free. Some Graphics Commands use Free Memory. This means that the larger your BASIC programs are, the more limitations on your Graphics capabilities.

Each Graphics program statement has a specific syntax and incorporates a Graphics BASIC command or function.

Table 1 gives a brief description of the BASICG commands; Table 2 lists the BASICG functions. This section of the manual will describe each statement and function in detail.

BASICG Commands

Command	Description
CIRCLE	Draws a circle, arc, semicircle, etc.
CLR	Clears the Graphics Screen.
GLOCATE	Sets the Graphics Cursor and the direction for putting characters on the Graphics Screen.
GET	Reads the contents of a rectangle on the Graphics Screen into an array for future use by PUT.
LINE	Draws a line from the startpoint to the endpoint in the specified line style and color. Also creates a box.
PAINT	Paints an area, starting from a specified point. Also paints a specified style.
PRESET	Sets an individual dot (pixel) OFF (or ON).
PRINT #-3	Writes characters to the Graphics Screen.
PSET	Sets an individual dot (pixel) ON (or OFF).
PUT	Stores graphics from an array onto the Graphics Screen.
SCREEN	Selects the Graphics or Text Screen.
VIEW	Creates a viewport which becomes the current Graphics Screen.

Table 1

BASICG Functions

Function	Description
&POINT	Returns the OFF/ON color value of a pixel.
&VIEW	**Returns the current viewport coordinates.**

Table 2

Starting-Up

Before using the diskette included with this package, be sure to make a "safe copy" of it. See your **Model III Disk System Owner's Manual** for information on BACKUP.

To load BASICG:

1. Power up your System according to the start-up procedure in your **Model III Disk System Owner's Manual.**
2. Insert the backup diskette into Drive 0.
3. Initialize the System as described in the "Operation" section of the **Model III Disk System Owner's Manual.**
4. When TRSDOS Ready appears, type:

```
BASICG <ENTER>
```

The Graphics BASIC start-up prompts, followed by the READY prompt (>), appear and you are in Graphics BASIC. You can now begin BASICG programming.

Remember that Model III numeric values are as follows:

Model III Numeric Values

Numeric Type	Range	Storage Requirement	Example
Integer	-32768, 32767	2 bytes	240, 639, -10
Single-Precision	$-1*10^{38},-1*10^{-38}$ $+1*10^{38},+1*10^{-38}$ Up to 7 significant digits (Prints six)	4 bytes	22.50,3.14259 -100.001
Double-Precision	$-1*10^{38},-1*10^{-38}$ $+1*10^{38},+1*10^{-38}$ Up to 17 significant digits (Prints 16)	8 bytes	1230000.00 3.1415926535897932

Table 3

With each BASICG command or function, there are various options which you may or may not include in a program statement (depending on your needs). Each option is separated from the previous option by a delimiter, usually a comma. When you do not specify an available option (e.g., you use the default value) and you specify subsequent options, you must still enter the delimiter or a Syntax Error will result. (See your **Model III Operation and BASIC Language Reference Manual** for more information).

Because you are dealing with two distinct screens, the Graphics Screen and the Text Screen, we strongly urge you to read the description of the **SCREEN** command before continuing.

CIRCLE
Draws Circle, Semicircle, Ellipse, Arc, Point

CIRCLE (x,y),r,c,start,end,ar

(x,y) specifies the centerpoint of the figure. x and y are integer expressions.

r specifies the radius of the figure in pixels and is an integer expression.

c specifies the OFF/ON color of the figure and is a integer expression of either 0 (OFF/black) or 1 (ON/white). c is optional; if omitted, 1 is used.

start specifies the startpoint of the figure and is a numeric expression from 0 to 6.283185. start is optional; if omitted, 0 is used.

end specifies the endpoint of the figure and is a numeric expression from 0 to 6.283185. end is optional; if omitted, 6.283185 is used.

ar specifies the aspect ratio of the circle, is a single-precision floating-point number > 0.0 (to 1*10^{38}) and determines the major axis of the figure. ar is optional; if omitted, .5 is used and a circle is drawn.

The CIRCLE command lets you draw five types of figures:

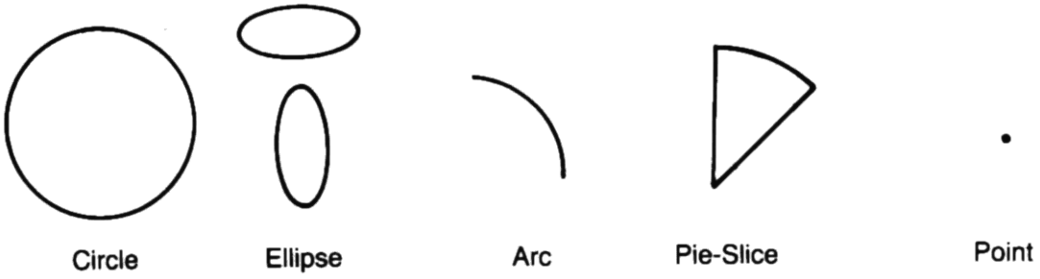


Figure 4. Types of Displays with CIRCLE

With CIRCLE, you can enter values for PI (and 2 x PI) up to 37 significant digits without getting an overflow error.

3.141592653589793238462643383279502884l
6.2831853071795864769252867665590057682

However, you'll probably only be able to visually detect a change in the circle's start and end when PI is accurate to a few significant digits (e.g., 3.1, 6.28, etc.). The start and end values can't be more than 2 x PI (e.g., 6.2832 will not work) or an Illegal Function Call error will occur.

(x,y)
Centerpoint

The (x,y) coordinates in the CIRCLE statement specify the centerpoint of the figure. x and y are numeric expressions in the integer number range.

Example

```
CIRCLE (x,y),r

CIRCLE (320,120),r
```

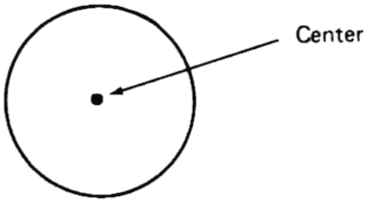


Figure 5. Center of Circle

r
Radius

The radius of a circle is measured in pixels and is a numeric expression in the integer range. Radius is the distance from the centerpoint to the edge of the figure. Although a negative value will be accepted by BASICG, the results of using a negative value are unpredictable.

The radius is either on the X-axis or Y-axis, depending on the aspect ratio (see ar). If the aspect ratio is greater than 1, the radius is measured on the Y-axis. If the aspect ratio is less than or equal to 1, the radius is measured on the X-axis.

Example

```
10 CIRCLE(320,120),100
```

This example draws a circle. The radius is 100 and the centerpoint is (320,120).

c
Color

You can set the ON/OFF (white/black) color of a figure's border and radius lines (see start/end) by specifying a numeric value of 1 or 0.

If you omit color, BASICG uses 1 (ON/white).

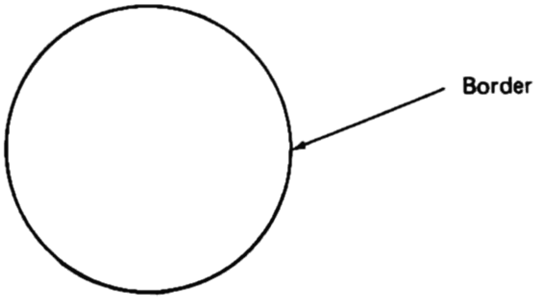


Figure 6. Border of Circle

start/end
Startpoint/Endpoint of Circle

The range for start and end is 0 to 6.283185 (2 x PI).

If you do not enter start and end, the default values of 0 and 6.28 respectively, are used.

A negative start or end value will cause the respective radius to be drawn in addition to the arc (i.e., it will draw a "piece of the pie"). The actual start and endpoints are determined by taking the absolute value of the specified start and endpoints. These values are measured in radians.

Note: Radius will not be drawn if start or end is -0. To draw a radius with start or end as 0, you must use -0.000...01.

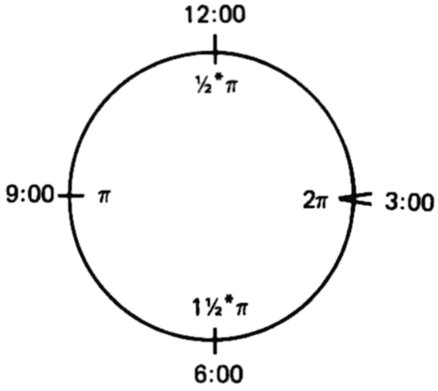


Figure 7. Clock/Radian Equivalents

Degrees	Radians	Clock Equivalent
0	0	3:00
90	1.57	12:00
180	3.14	9.00
270	4.71	6:00
360	6.28	3:00

Table 4. Degree/Radians/Clock Equivalents

You can draw semicircles and arcs by varying start and end. If start and end are the same, a point (one pixel) will be displayed instead of a circle.

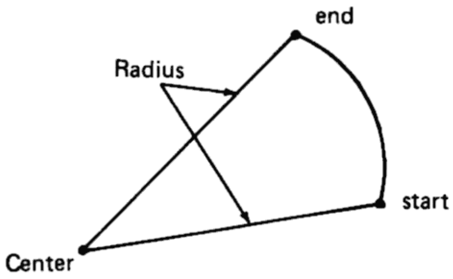


Figure 8. CIRCLE's (-) start, (-) end

You can have a positive start and a negative end (or vice versa) as well as negative starts and ends. In these cases, only one radius line is drawn.

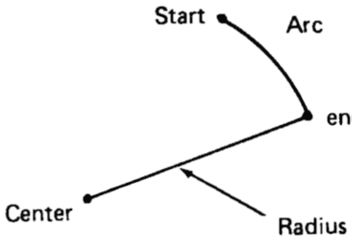


Figure 9. CIRCLE's (+) start, (-) end

Hints and Tips about start and end:

- When using the default values for start and end, you must use commas as delimiters if you wish to add more parameters.
- If you use PI, it is not a reserved word in BASICG and must be defined in your program.

ar
Aspect Ratio

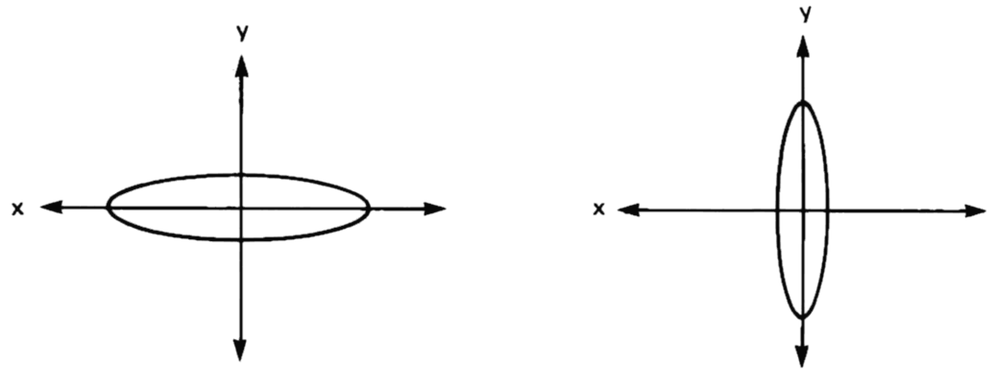
You can draw ellipses by varying the aspect ratio from the default value (.5) for a circle (and semicircle).

Every ellipse has a "major axis" which is the ellipse's longer, predominant axis. With an ellipse (as with a circle), the two axes are at right angles to each other.

The mathematical equation for determining the aspect ratio is:

ar = length of Y-axis/length of X-axis

- If the aspect ratio is .5, a circle is drawn.
- If the ratio is less than .5, an ellipse with a major axis on the X-axis is drawn.
- If the ratio is greater than .5, an ellipse with a major axis on the Y-axis is drawn.



X-Axis Ellipse (*ar* < .5) **Y-Axis Ellipse (*ar* > .5)**

Figure 10. CIRCLE's Ellipse

The range for aspect ratio is a single-precision floating-point number greater than 0.0 (to $1*10^{38}$). Although a negative value will be accepted by BASICG, the results of using a negative value are unpredictable.

Hints and Tips about aspect ratio:

- Entering .5 as the ratio produces a circle.
- Numbers between 0 and .5 produce an ellipse with a major axis on X.
- Numbers over .5 generate an ellipse with a major axis on Y.
- Even though you can enter large aspect ratios, large numbers may produce straight lines.

Examples

```
CIRCLE (320,120),90,1
```

This example draws a white-bordered circle with the centerpoint of (320,120) and radius of 90.

```
CIRCLE (320,120),90,1,,,.7
```

This statement draws a white-bordered ellipse with an origin of (320,120) and radius of 90. The major axis is the Y-axis.

```
CIRCLE (320,120),90,1,-6.2,-5
```

This statement draws an arc with a vertex ("origin") of (320,120) and radius of 90. start is 6.2 and end is 5. Radius lines are drawn for start and end.

```
CIRCLE (320,120),90,1,,-4
```

This example draws an arc with a vertex of (320,120) and radius of 90. start is 0 and end is 4. A radius line is drawn for end.

```
10 PI=3.1415926
20 CIRCLE (320,120),100,1,PI,2*PI,.5
```

A semicircle is drawn.

```
10 CIRCLE (150,100),100,1,-5,-1
20 CIRCLE (220,100),100,1,5,1
```

Two arcs are drawn with the same start and end point. The arc with the negative start and end has two radius lines drawn to the vertex. The arc with a positive start and end has no radius lines.

```
CIRCLE (320,120),140,,-4,6.1
```

This statement draws an arc with a vertex at (320,120) and a radius of 140. Start is 4 and end is 6.1. A radius line is drawn for start.

```
CIRCLE (320,120),140,1,0,1,.5
```

This example draws an arc with a vertex of (320,120) and radius of 140.

Sample Program

```
4 SCREEN 0
5 CLR
10 FOR X=10 TO 200 STEP 10
20 CIRCLE (300,100),X,1,,,.9
30 NEXT X
40 FOR Y=10 TO 200 STEP 10
50 CIRCLE (300,100),Y,1,,,.1
60 NEXT Y
70 FOR Z=10 TO 200 STEP 10
80 CIRCLE (300,100),Z,1,,,.5
90 NEXT Z
100 GOTO 5
```

A set of 20 concentric ellipses is drawn with a major axis on Y, a set of 20 concentric ellipses is drawn with a major axis on X, and a set of 20 concentric circles is drawn. The ellipses and circles in each of the three groups are concentric and the radius varies from 10 to 200.

CLR
Clears the Graphics Screen

CLR

CLR clears the Graphics Screen.

Example

```
10 SCREEN 0
20 CIRCLE(320,120),100,1
```

This program line will draw a circle. Now type:

```
CLR <ENTER>
```

and the Graphics Screen will be cleared but the Text Screen will remain unchanged. This can be seen by typing:

```
SCREEN 1
```

GET

Reads the Contents of Rectangular Pixel Area into Array

GET(x1,y1)-(x2,y2),array name

(x1,y1) are coordinates of one of the opposing corners of a rectangular pixel area. x1 is an integer expression from 0 to 639. y1 is an integer expression from 0 to 239.

(x2,y2) are coordinates of the other corner of a rectangular pixel area. x2 is an integer expression from 0 to 639. y2 is an integer expression from 0 to 239.

array name is the name you assign to the array that will store the rectangular area's contents. array name must be specified.

Important Note: BASICG recognizes two syntaxes of the command GET -- the syntax described in this manual and the syntax described in the **Model III Operation and BASIC Language Reference Manual.** BASIC recognizes only the GET syntax described in the **Model III Operation and BASIC Language Reference Manual.**

GET reads the graphic contents of a rectangular pixel area into a storage array for future use by PUT (see PUT).

A rectangular pixel area is a group of pixels which are defined by the diagonal line coordinates in the GET statement.

The first two bytes of array name are set to the horizontal (X-axis) number of pixels in the pixel area; the second two bytes are set to the vertical (Y-axis) number of pixels in the pixel area. The remainder of array name represents the status of each pixel, either ON or OFF, in the pixel area. The data is stored in a row-by-row format. The data is stored 8 pixels per byte and each row starts on a byte boundary.

Array Limits

When the array is created, BASICG reserves space in memory for each element of the array. The size of the array is limited by the amount of memory available for use by your program -- each real number in your storage array uses four memory locations (bytes).

The array must be large enough to hold your graphic display and the rectangular area must include all the points you want to store.

Your GET rectangular pixel area can include the entire screen (i.e., GET(0,0)-(639,239),array name), if the array is dimensioned large enough.

To determine the minimum array size:

1. Divide the number of X-axis pixels by 8 and round up to the next higher integer.
2. Multiply the result by the number of Y-axis pixels. When counting the X-Y axis pixels, be sure to include the first and last pixel.
3. Add four to the total.
4. Divide by four (for real numbers) or two (for integers) rounding up to the next higher integer.

The size of the rectangular pixel area is determined by the (x,y) coordinates used in GET:

Position:	upper-left corner = startpoint = (x1,y1) lower-left corner = endpoint = (x2,y2)

Size (in pixels):	width = x2-x1+1
	length = y2-y1+1

Example

```
GET(10,10)-(80,50),V
```

This block is 71 pixels wide on the X-axis (10 through 80) and 41 long on the Y-axis (10 through 50).

- For real: 71/8 = 9 * 41 = 369 + 4 = 373/4 = 94
- For integer: 71/8 = 9 * 41 = 369 + 4 = 373/2 = 187

Depending on the type of array you use, you could set up your minimum-size dimension statement this way:

- Real DIM V(93)

or

- Integer DIM V%(186)

Examples

```
10 DIM V(249)
20 CIRCLE (65,45),20,1
30 GET (10,10)-(120,80),V
```

An array is created, a circle is drawn and stored in the array via the GET statement's rectangular pixel area's parameters (i.e., (10,10)-(120,80)).

Calculate the dimensions of the array this way:

Rectangular pixel area is 111 x 71. That equals:

111/8= 14 * 71 =994 + 4 = 998/4 = 250

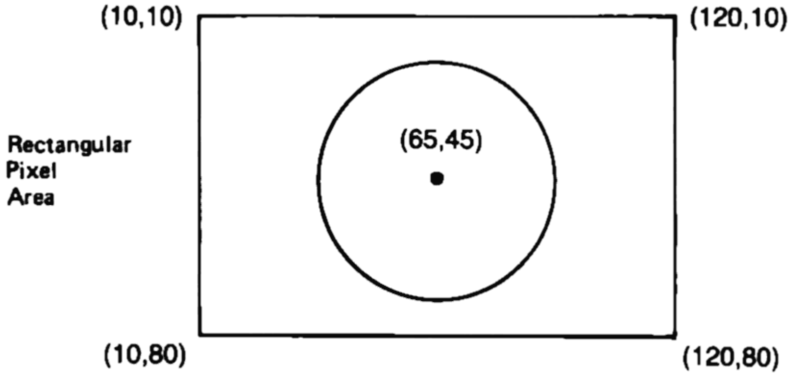


Figure 11

```
10 DIM V(30,30)
20 CIRCLE (50,50),10
30 GET (10,10)-(80,80),V
```

A two-dimensional array is created, a circle is drawn and stored in the array via the GET statement's rectangular pixel area's parameters (i.e., (10,10)-(80,80)).

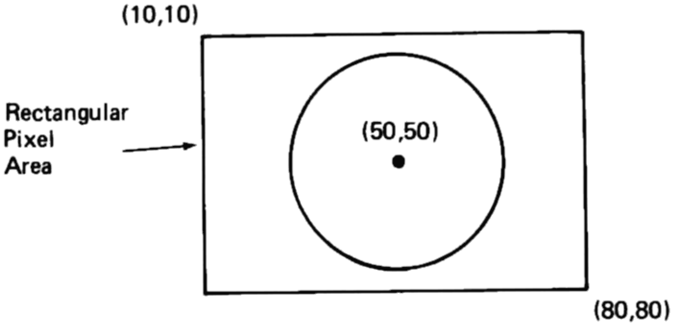


Figure 12

```
10 DIM V%(564)
20 CIRCLE (65,45),50,1,1,3
30 GET(10,10)-(120,80),V%
```

A one-dimensional integer array is created, an arc is drawn and stored in the array via the GET statement's rectangular area's parameters.

GLOCATE

Sets the Graphics Cursor

GLOCATE (x,y), direction

(x,y) specifies the location of the Graphics Cursor and is a pair of integer expressions.
direction specifies the direction that the characters will be written to the Graphics Screen and has an integer value of 0, 1, 2, or 3. direction is optional; if omitted, 0 is used.

Since the Text Screen and the Graphics Screen cannot be displayed at the same time, you need an easy way to display textual data on the Graphics Screen. GLOCATE provides part of this function by allowing you to specify where on the Graphics Screen to start displaying the data, (x,y), and which direction to display it -- direction.

The allowable values for direction are:

0 - zero degree angle
1 - 90 degree angle
2 - 180 degree angle
3 - 270 degree angle

Examples

```
10 GLOCATE (320,120),0
```

This program line will cause characters to be displayed starting in the center of the screen in normal left-to-right orientation.

```
100 GLOCATE (320,10),1
```

This program line will cause characters to be displayed starting in the center of the top portion of the screen in a vertical orientation, going from the top of the screen to the bottom of the screen.

```
200 GLOCATE (630,120),2
```

This program line will cause characters to be displayed upside down starting at the right of the screen and going towards the left.

```
300 GLOCATE (320,230),3
```

This program line will cause the characters to be displayed vertically, starting at the center of the lower portion of the screen going towards the top of the screen.

LINE

Draws a Line or Box

LINE (x1,y1)-(x2,y2), c, B or BF, style

(x1,y1) specifies the starting coordinates of a line and is a pair of integer expressions. (x1,y1) is optional; if omitted, the last ending coordinates of any previous command are used as the startpoint. If a command has not been previously specified, (0,0) is used.

(x2,y2) specifies the ending coordinates of a line. (x2,y2) is a pair of integer expressions.

c specifies the color and is a numeric expression of either 0 or 1. c is optional; if omitted, 1 is used.

B or BF specifies drawing and/or shading (solid white or solid black) a box. B draws a box and BF fills a box with shading. B/BF is optional; if omitted, only a line is drawn.

style is the setting for the pattern of a line and is a numeric value in the integer range. style is optional; if omitted, -1 (solid line) is used. style must be omitted if BF is used.

LINE draws a line from the starting point (x1,y1) to the ending point (x2,y2).

If the starting point is omitted, either (0,0) is used if a previous end coordinate has not been specified or the last ending point of the previous command is used. If one or both parameters are off the screen, only the part of the line which is visible is displayed.

With over 65,500 line styles possible, each style is slightly different. You'll find it's almost impossible to detect some of the differences since they are so minute.

LINE with Box Option

The start and end coordinates are the diagonal coordinates of the box (either a square or rectangle). When you don't specify the **B** or **BF** options, the "diagonal" line is drawn. When you specify the **B** option, the perimeter is drawn but not the diagonal line. When you specify the **BF** option, the perimeter is drawn, and the area bounded by the perimeter is shaded in the specified color (c).

```
LINE(140,80)-(500,200),1,B
```

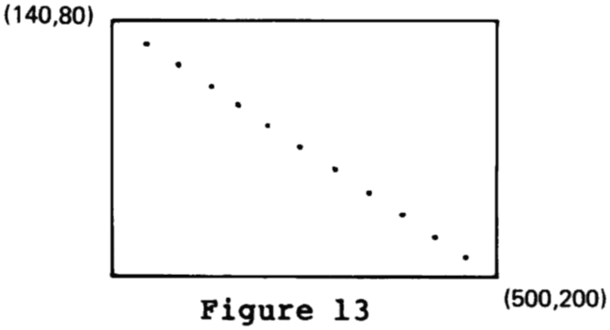


Figure 13

style

style sets the pixel arrangement in 16-bit groups.

For example, 0000 1111 0000 1111 (binary), 0F0F (hex), or 3855 (decimal).

style can be any number in the integer range (negative or positive). Using hexadecimal numbers, you can figure the exact line style you want. There will always be four numbers in the hexadecimal constant.

To use hexadecimal numbers for style:

1. Decide what pixels you want OFF (bit=0) and ON (bit=1).
2. Choose the respective hexadecimal numbers (from the **Base Conversion Chart,** Appendix E).

Example

```
0000 1111 0000 1111 = &H0F0F
```

Creates a dashed line.

Type	Binary Numbers	Hex Numbers
Long dash	0000 0000 1111 1111	&H00FF
Short dash	0000 1111 0000 1111	&H0F0F
"Short-short" dash	1100 1100 1100 1100	&HCCCC
Solid line	1111 1111 1111 1111	&HFFFF
OFF/ON	0101 0101 0101 0101	&H5555
"Wide" dots	0000 1000 0000 1000	&H0808
Medium dots	1000 1000 1000 1000	&H8888
Dot-dash	1000 1111 1111 1000	&H8FF8

Table 5. Sample Line Styles

Examples

```
LINE -(100,40)
```

This example draws a line in white (ON) starting at the last endpoint used and ending at (100,40).

```
LINE (0,0)-(319,199)
```

This statement draws a white line starting at (0,0) and ending at (319,199).

```
LINE(100,100)-(200,200),1,,45
```

This example draws a line from (100,100) to (200,200) using line style 45 (&H002D).

```
LINE (100,100)-(300,200),1,,&H00FF
```

This LINE statement draws a line with "long dashes." Each dash is eight pixels long and there are eight blank pixels between each dash.

```
LINE (100,100)-(300,200),1,,-1000
```

This statement draws a line from (100,100) to (300,200) using line style -1000.

```
LINE (200,200)-(-100,100)
```

A line is drawn from the startpoint of (200,200) to (-100,100).

```
10 LINE (30,30)-(180,120)
20 LINE -(120,180)
30 LINE -(30,30)
```

This program draws a triangle.

```
10 LINE -(50,50)
20 LINE -(120,80)
30 LINE -(-100,-100)
40 LINE -(3000,1000)
```

This program draws four line segments using each endpoint as the startpoint for the next segment.

PAINT
Paints Screen

PAINT (x,y), tiling, border, background

(x,y) specifies the X-Y coordinates where painting is to begin. x is a numeric expression from 0 to 639 and y is a numeric expression from 0 to 239.

tiling specifies the paint style and can be a string or a numeric expression. tiling is optional; if omitted, 1 is used. tiling cannot be a null string ("") and no more than 64 bytes may be contained in the tiling string.

border specifies the OFF/ON color of the border where painting is to stop and is a numeric expression of either 0 (OFF) or 1 (ON). border is optional; if omitted, 1 is used.

background specifies the color of the background that is being painted and is a 1-byte string of either 0 (CHR$(&H00)) or 1 (CHR$(&HFF)). background is optional; if omitted, CHR$(&H00) is used.

PAINT shades the Graphics Screen with tiling starting at the specified X-Y coordinates, proceeding upward and downward.

x,y
Paint Startpoint

x,y is the coordinate where painting is to begin and must:

- Be inside the area to be painted.
- Be on the working area of the screen.

For example:

```
10 CIRCLE(320,120),80
20 PAINT(320,120),1,1
```

A circle with a centerpoint of (320,120) is drawn and painted in white.

tiling
Paint Style

tiling is the pattern in a graphics display. By specifying each pixel, you can produce a multitude of tiling styles thereby simulating different shades of paint on the screen.

tiling is convenient to use in bar graphs, pie charts, etc., or whenever you want to shade with a defined pattern.

There are two types of tiling:

- Numeric expressions
- Strings

Numeric Expressions. There are only two numeric expressions that can be used for the paint style -- 0 and 1. 1 paints all pixels ON (solid white) and 0 paints all pixels OFF (solid black).

To use numeric expressions, enter either a 0 or 1. For example:

```
          PAINT (320,120),1,1
```

Strings (Point-by-Point Painting). You can paint precise patterns using strings by defining a multi-pixel grid, pixel-by-pixel, on your screen as one contiguous pattern.

String painting is called "pixel" painting because you are literally painting the screen "pixel-by-pixel" in a predetermined order.

You can define the tile length as being one to 64 vertical tiles, depending on how long you want your pattern. Tile width, however, is always eight horizontal pixels (8 pixels representing one 8-bit byte). The dimensions of a tile pattern are length by width. Tile patterns are repeated as necessary to paint to the specified borders. Because of its symmetry, you'll probably find equilateral pixel grids most convenient.

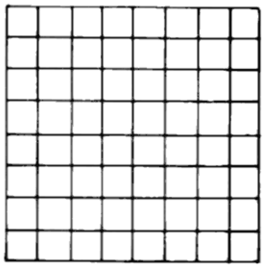


Figure 14. Example of an 8-by-8 Pixel Grid

Strings allow numerous graphic variations because of the many pixel combinations you can define.

Important Note: You cannot use more than two consecutive rows of tiles which match the background or an Illegal Function Call error will occur. For example:

```
PAINT (1,1),CHR$(&HFF)+CHR$(&HFF)+CHR$(&H00)+CHR$(&H00)
+CHR$(&H00)+CHR$(&H00),1,CHR$(&H00)
```

returns an Illegal Function Call error.

Using Tiling

You may want to use a sheet of graph paper to draw a style pattern. This way, you'll be able to visualize the pattern and calculate the binary and hexadecimal numbers needed.

Note: Tiling should only be done on either a totally black or white background; otherwise, results are unpredictable.

To draw an example of a tile on paper:

1. Take a sheet of paper and draw a grid according to the size you want (8 x 8, 24 x 8, etc.). Each boxed area on this grid, hypothetically, represents one pixel on your screen.
2. Decide what type of pattern you want (zigzag, diagonal lines, perpendicular lines, etc.).
3. Fill in each grid in each 8-pixel-wide row of the tile if you want that pixel to be ON, according to your pattern. If you want the pixel to be OFF, leave the

grid representing the pixel blank.

4. On your paper grid, count each ON pixel as 1 and each OFF pixel as 0. List the binary numbers for each row to the side of the grid. For example, you might have 0001 1000 on the first row, 0111 0011 on the second row, etc.

5. Using a hexadecimal conversion chart, convert the binary numbers to hexadecimal numbers. (Each row equates to a two-digit hexadecimal number.)

6. Insert the hexadecimal numbers in a tile string and enter the string in your program.

Note: For a listing of commonly used tiling styles, see Appendix F.

Example

For example, if you're working on an 8 x 8 grid and want to draw a plus ("+") sign:

8 x 8 grid								Binary	Hex
0	0	0	1	1	0	0	0	0001 1000	18
0	0	0	1	1	0	0	0	0001 1000	18
0	0	0	1	1	0	0	0	0001 1000	18
1	1	1	1	1	1	1	1	1111 1111	FF
1	1	1	1	1	1	1	1	1111 1111	FF
0	0	0	1	1	0	0	0	0001 1000	18
0	0	0	1	1	0	0	0	0001 1000	18
0	0	0	1	1	0	0	0	0001 1000	18

Figure 15

Tile string:

```
A$=CHR$(&H18)+CHR$(&H18)+CHR$(&H18)+CHR$(&HFF)+CHR$(&HFF)
   +CHR$(&H18)+CHR$(&H18)+CHR$(&H18)
```

<u>b</u>
Border

Border is the OFF/ON color of the border of a graphics design where painting is to stop and is a numeric expression of either 0 or 1. If omitted, 1 (ON) is used and all the pixels on the border are set (solid white).

<u>background</u>
Background Area

Background is a 1-byte character which describes the background of the area you are painting. CHR$(&H00) specifies a black background and CHR$(&HFF) is a totally white background. If background is not specified, BASICG uses CHR$(&H00).

Painting continues until a border is reached or until PAINT does not alter the state of any pixels in a row. However, if pixels in a given row are not altered and the tile that was to be painted in that row matches the background tile, painting will continue on to the next row.

Note: BASICG uses Free Memory for tiling.

Examples

```
10 CIRCLE (300,100),100
20 PAINT (300,100),1,1
```

Paints the circle in solid white.

```
10 CIRCLE (100,100),300
20 PAINT (100,100),1,1
```

Paints the circle. Only the visible portion of the circle is painted on the screen.

```
5 A=1
6 SCREEN 0
10 CIRCLE (320,120),100
20 CIRCLE (100,100),50
30 CIRCLE (400,200),60
40 CIRCLE (500,70),50
50 PAINT (320,120),A,1
60 PAINT (100,100),A,1
70 PAINT (400,200),A,1
80 PAINT (500,70),A,1
```

The tiling style is assigned the value 1 in line 5 (A=1) for all PAINT statements. Four circles are drawn and painted in solid white.

```
10 LINE (140,80)-(500,200),1,B
20 PAINT (260,120),CHR$(&HEE)+CHR$(&H77)+CHR$(00),1
```

Paints box in specified tiling style using strings.

```
10 CIRCLE (300,100),100
20 PAINT (300,100),"D",1
```

This example uses a character constant to paint the circle in vertical black and white stripes. The character "D" (0100

0100) sets this vertical pattern: one vertical row of pixels ON, three rows OFF.

```
10 CIRCLE (320,120),200
20 PAINT (320,120),"332211",1
30 PAINT (100,70),"EFEF",1
```

This example draws and paints a circle, then paints the area surrounding the circle with a different paint style (line 30). This PAINT statement's (line 30) startpoint must be outside the border of the circle.

```
10 PAINT (320,120),CHR$(&HFF),1
20 CIRCLE (320,120),100,0
30 PAINT (320,120),CHR$(0)+CHR$(&HFF),0,CHR$(&HFF)
```

Paints the screen white, draws a circle and paints the circle with a pattern.

```
10 PAINT (320,120),CHR$(&HFF),1
20 CIRCLE (320,120),100,0
30 PAINT (320,120),CHR$(0)+CHR$(&HAA),0,CHR$(&HFF)
```

Paints the screen white, draws a circle and paints the circle with a pattern.

```
10 CIRCLE(300,100),100
20 A$=CHR$(&H00)+CHR$(&H7E)+CHR$(&H18)+CHR$(&H18)+CHR$(&H18)
   +CHR$(&H18)+CHR$(&H18)+CHR$(&H00)
30 PAINT(300,100),A$,1
```

This draws the circle and paints with the letter T within the parameters of the circle.

```
10 A$=CHR$(&H41)+CHR$(&H22)+CHR$(&H14)+CHR$(&H08)+CHR$(&H14)
  +CHR$(&H22)+CHR$(&H41)+CHR$(&H00)
20 PAINT (300,100),A$, 1
```

This paints Xs over the entire screen.

```
1 CLEAR 100
5 SCREEN 0
10 TILE$(0)=CHR$(&H22)+CHR$(&H00)
20 TILE$(1)=CHR$(&HFF)+CHR$(&H00)
30 TILE$(2)=CHR$(&H99)+CHR$(&H66)
40 TILE$(3)=CHR$(&H99)
50 TILE$(4)=CHR$(&HFF)
60 TILE$(5)=CHR$(&HF0)+CHR$(&HF0)+CHR$(&H0F)+CHR$(&H0F)
70 TILE$(6)=CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)
80 TILE$(7)=CHR$(&H03)+CHR$(&H0C)+CHR$(&H30)+CHR$(&HC0)
90 A$=TILE$(0)+TILE$(1)+TILE$(2)+TILE$(3)+TILE$(4)
   +TILE$(5)+TILE$(6)+TILE$(7)
100 PAINT(300,100),A$,1
```

This example paints the screen with a tiling pattern made up of eight individually defined tile strings (0-7).

&POINT (function)
Returns Pixel Value

&POINT(x,y)

x specifies an X-coordinate and is an integer expression.
y specifies an Y-coordinate and is an integer expression.
values returned by &POINT are:
0 (pixel OFF)
1 (pixel ON)
-1 (pixel is off the screen)

The &POINT command lets you read the OFF/ON value of a pixel from the screen.

Values for &POINT that are off the screen (i.e., PRINT &POINT (800,500)) return a -1, signifying the pixel is off the screen.

Example

```
10 PSET(300,100),1
20 PRINT &POINT(300,100)
```

Reads and prints the value of the pixel at the point's coordinates (300,100) and displays its value: 1.

```
PRINT &POINT(3000,1000)
```

Since the pixel is off the screen, a -1 is returned.

```
PRINT &POINT(-3000,-1000)
```

Since the pixel is off the screen, a -1 is returned.

```
PSET(200,100),0
PRINT &POINT(200,100)
```

Reads and prints the value of the pixel at the point's coordinates (200,100) and displays its value: 0.

```
10 PSET(300,100),1
20 IF &POINT(300,100)=1 THEN PRINT "GRAPHICS BASIC!"
```

Sets the point ON. Since the point's value is 1, line 20 is executed and Graphics BASIC is displayed:

```
GRAPHICS BASIC!
```

```
5 SCREEN 0
10 PSET(RND(640),RND(240)),1
20 IF &POINT(320,120)=1 THEN STOP
30 GOTO 10
```

Sets points randomly until (320,120) is set.

```
5 CLR
10 LINE(50,80)-(120,100),1,BF
20 PRINT &POINT(100,80)
30 PRINT &POINT(110,80)
40 PRINT &POINT(115,90)
50 PRINT &POINT(50,40)
60 PRINT &POINT(130,120)
```

The first three pixels are in the filled box, so the value 1 (one) is displayed for each of the statements in lines 20, 30, and 40. The pixels specified in lines 50 and 60 are not in the shaded box and 0s are returned.

PRESET

Sets Pixel OFF (or ON)

PRESET(x,y),switch

x specifies an X-coordinate and is an integer expression.
y specifies an Y-coordinate and is an integer expression.
switch specifies a pixel's OFF/ON code and is an integer of either 0 (OFF) or 1 (ON).
switch is optional; if omitted, 0 (OFF) is used.

PRESET sets a pixel either OFF (0) or ON (1), depending on switch. If switch is not specified, 0 (OFF) is used.

Values for (x,y) that are larger than the parameters of the screen (i.e., greater than 639 for x and 239 for y) are accepted, but these points are off the screen and therefore are not PRESET.

Note: The only choice for switch is 0 or 1. If you enter any other number, an Illegal Function Call error will result.

Examples

```
10 PRESET (50,50),1
20 PRESET (50,50),0
```

Turns ON the pixel located at the specified coordinates (in line 10) and turns the pixel OFF (in line 20).

```
10 PRESET (320,120),1
20 PRESET (300,100),1
30 PRESET (340,140),1
40 FOR I=1 TO 1000: NEXT I
50 PRESET (320,120)
60 PRESET (300,100)
70 PRESET (340,140)
80 FOR I=1 TO 1000: NEXT I
```

Sets the three specified pixels ON (through the three PRESET statements), pauses, and then turns the three pixels OFF.

```
PRESET(3000,1000),1
```

The values for (x,y) are accepted, but since the coordinates are beyond the parameters of the screen, the point is not PRESET.

PRINT #-3,
Write Text Characters to the Graphics Screen

PRINT #-3, item list

item list may be either string constants (messages enclosed in quotes), string variables, numeric constants (numbers), variables, or expressions involving all of the preceding items. The items to be printed may be separated by commas or semicolons. If commas are used, the Cursor automatically advances to the next print zone before printing the next item. If semicolons are used, a space is not inserted between the items printed on the screen. In cases where no ambiguity would result, all punctuation can be omitted.

PRINT #-3, is used to write text characters to the Graphics Screen. This is the easiest way to display textual data on the Graphics Screen. Characters are displayed starting at the current Graphics Cursor and going in the direction specified by the most recently executed GLOCATE command. If a GLOCATE command was not executed prior to the PRINT #-3, command, a direction of 0 is assumed.

PRINT #-3, will only print text characters (see Appendix C of the **Model III Operation and BASIC Language Reference Manual**). Each character displayed in the 0 or 2 direction uses an 8 X 8 pixel grid; each character displayed in the 1 or 3 direction uses a 16 X 8 grid. Executing this command will position the Graphics Cursor to the end of the last character that was displayed.

Displaying text in direction 0 engages a wraparound feature. If the end of a line is reached, BASICG will continue the display on the next line. If the end of the screen is reached, BASICG will continue the display at the beginning of the screen without scrolling. If there is not enough room to display at least one character at the current Graphics Cursor, an Illegal Function Call error will result. When displaying text in other directions, an attempt to display text outside of the currently defined screen will cause an Illegal Function Call error to be given.

PSET
Sets Pixel ON (or OFF)

PSET(x,y),switch

x specifies an X-coordinate and is an integer expression.
y specifies an Y-coordinate and is an integer expression.
switch specifies a pixel's OFF/ON color code and is a numeric expression of 0 (OFF) or 1 (ON). switch is optional; if omitted, 1 (ON) is used.

PSET sets a pixel either OFF (0) or ON (1), depending on switch. If switch is not specified, 1 (ON) is used.

The only choice for switch with PSET is 0 and 1. If you enter any other number, an Illegal Function Call will occur.

Values for (x,y) that are larger than the parameters of the screen (i.e., greater than 639 for x and 239 for y) are accepted, but these points are off the screen and therefore are not PSET.

Note: The only distinction between PRESET and PSET in BASICG is the default value for switch. The default value for PRESET is 0, while the value for PSET is 1.

Examples

```
10 A=1
20 PSET (50,50),A
```

Turns the pixel located at the specified coordinates ON.

```
10 PSET (RND(640),RND(240)),1
20 GOTO 10
```

Pixels are randomly set to 1 (ON) over the defined area (the entire screen).

```
PSET (-300,-200),1
```

The values for (x,y) are accepted, but since it is beyond the parameters of the screen, the pixel is not set.

```
10 PSET (320,120),1
20 A$=INKEY$: IF A$= "" THEN 20
30 PSET(320,120),0
```

Line 10 sets ("turns ON") a pixel; line 30 resets ("turns OFF") the same dot.

PUT
Puts Rectangular Pixel Area from Array onto Screen

PUT(x1,y1),array name,action

(x1,y1) are coordinates of the upper-left corner of the rectangular pixel area which is to contain a graphic display. x1 is a numeric expression from 0 to 639 and y1 is a numeric expression from 0 to 239.

array name is the name of an array (previously specified by GET) that contains the data to be written into the rectangular pixel area.

action determines how the data is written into the rectangular pixel area and is one of the following:

PSET	Sets or resets each point in the specified pixel area to the value in the specified array.
PRESET	Sets or resets each point in the specified pixel area to the inverse of the value in the specified array.
XOR	Performs a logical exclusive-OR between the bits in the specified array and the pixels in the destination area and displays the result.
OR	Performs a logical OR between the bits in the specified array and the pixels in the destination area and displays the result.
AND	Performs a logical AND between the bits in the specified array and the pixels in the destination area and displays the result.

action is optional; if omitted, XOR is used.

Important Note: BASICG recognizes two syntaxes of the command PUT -- the syntax described in this manual and the syntax described in the **Model III Operation and BASIC Language Reference Manual.** BASIC recognizes only the PUT syntax described in the **Model III Operation and BASIC Language Reference Manual.**

The PUT function puts a rectangular pixel area stored in an array, and defined by GET, onto the screen. GET and PUT work jointly. Together, they allow you to "get" a rectangular pixel area which contains a graphic display, store it in an array, then "put" the array back on the screen later.

Remember that before you GET or PUT, you have to create an array to store the bit contents of the display rectangular pixel area. The size of the array must match that of the display rectangular pixel area.

PUT moves your GET rectangular pixel area to the startpoint in your PUT statement and the startpoint is the new upper-left corner of the rectangular pixel area.

To illustrate:

```
5 DIM V(3)
10 GET (2,3)-(7,7),V
100 PUT (50,50),V,PSET
```

After GETting, PUT this rectangular pixel area to (50,50). The new coordinates are:

(50,50)	(51,50)	(52,50)	(53,50)	(54,50)	(55,50)
(50,51)	(51,51)	(52,51)	(53,51)	(54,51)	(55,51)
(50,52)	(51,52)	(52,52)	(53,52)	(54,52)	(55,52)
(50,53)	(51,53)	(52,53)	(53,53)	(54,53)	(55,53)
(50,54)	(51,54)	(52,54)	(53,54)	(54,54)	(55,54)

The rectangular pixel area ((50,50)-(55,54)) is exactly the same pixel size as (2,3)-(7,7); only the location is different.

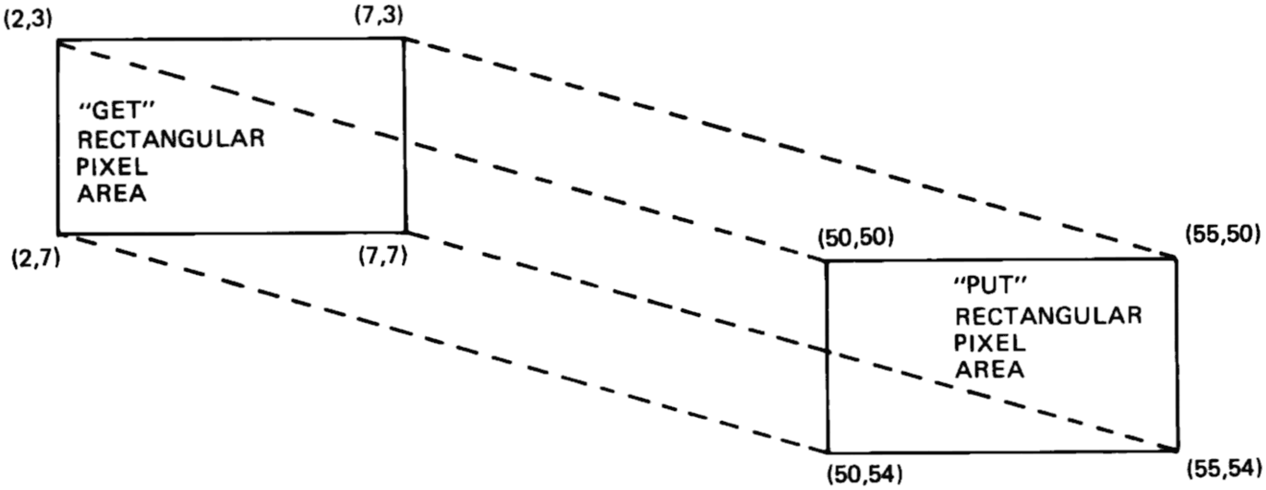


Figure 16

With PUT, action can be PSET, PRESET, OR, AND, or XOR.

These operators are used in BASICG to test the OFF/ON (or 0/1) conditions of a pixel in the original pixel area and the destination pixel area.

For example (using PSET), the pixel is set ON only if the bit in the PUT array is set ON. If the bit is OFF, the pixel is turned OFF (reset).

With PRESET, the pixel is set ON only if the bit in the PUT array is set OFF. If the bit is ON, the pixel is turned OFF (reset).

Using OR, the pixel is set ON if the bit in the PUT array is ON or the corresponding pixel in the destination area is ON. In all other cases, the pixel is turned OFF (reset). In other words:

OR	OFF	ON
OFF	OFF	ON
ON	ON	ON

With AND, the pixel is set ON if both the bit in the PUT array and the corresponding pixel in the destination area are ON. In all other cases, the pixel is turned OFF (reset). In other words:

AND	OFF	ON
OFF	OFF	OFF
ON	OFF	ON

Using XOR, the pixel is set ON if either the bit in the PUT array or the corresponding pixel in the destination area (but not both) is ON. In all other cases, the pixel is turned OFF (reset). In other words:

XOR	OFF	ON
OFF	OFF	ON
ON	ON	OFF

The following BASICG program will graphically illustrate the differences between the various action options. Since the program will give you a "hard-copy" printout of the action options, you'll need to connect your TRS-80 to a graphic printer. See "Graphics Utilities" later in this manual for more details on using the Computer Graphics package with a printer.

```
10 DATA "OR", "AND", "PRESET", "PSET", "XOR"
20 CLR : SCREEN 0
30 FOR Y= 10 TO 210 STEP  50
40 FOR X=  0 TO 400 STEP 200
50 LINE (X+40,Y-5)-(X+100,Y+25),1,B
60 NEXT X
70 LINE (50,Y)-(90,Y+10),1,BF
80 FOR X= 200 TO 400 STEP 200
90 LINE (X+50,Y)-(X+70,Y+20),1,BF
100 NEXT X
110 NEXT Y
120 DIM V(100)
130 GET (50,10)-(90,30),V
140 FOR N= 1 TO 5
150 R= (N-1)*5+1
160 READ A$
165 GLOCATE (136,R*10),0
170 PRINT #-3, A$;
175 GLOCATE (360,R*10),0
180 PRINT #-3, "= ";
190 ON N GOTO 200, 210, 220, 230, 240
200 PUT (450,10),  V,OR:     GOTO 250
210 PUT (450,60),  V,AND:    GOTO 250
220 PUT (450,110), V,PRESET: GOTO 250
230 PUT (450,160), V,PSET:   GOTO 250
240 PUT (450,210), V,XOR
250 NEXT N
260 CMD "I", "GPRINT"
270 SCREEN1
```

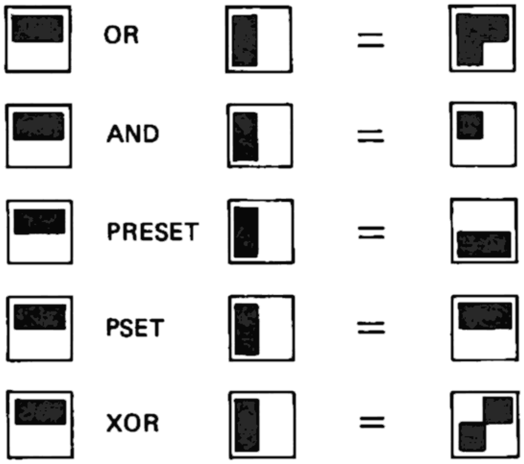


Figure 17

Hints and Tips about PUT:

- An Illegal Function Call error will result if you attempt to PUT a rectangular pixel area to a section of the screen which is totally or partially beyond the parameters of the screen. For example:

  ```
  GET(50,50)-(150,150),V
  PUT(200,200),V,PSET
  ```

  returns an error because the rectangular pixel area cannot be physically moved to the specified rectangular pixel area (i.e., (200,200)-(300,300)).

- If you use PUT with a viewport (see VIEW), all coordinates must be within the parameters of the viewport or you'll get an Illegal Function Call error.

Examples

PUT with PSET

```
10 DIM V%(63)
15 SCREEN 0
20 CIRCLE (30,30),10
30 GET (10,10)-(40,40),V%
40 FOR I=1 TO 500: NEXT I
50 CLR
60 PUT (110,110),V%,PSET
70 FOR I=1 TO 500: NEXT I
```

In this example, the circle is drawn, stored, moved and re-created. First the white-bordered circle appears in the upper left corner of the screen (position (30,30) -- program line 20). After a couple of seconds (because of the delay loop), it disappears and then reappears on the screen -- (110,110) -- program line 60.

What specifically happened is:

1. An array was created (line 10).

2. A circle was drawn (line 20).

3. GET -- The circle which was within the source rectangular pixel area, as specified in the GET

statement's parameters is stored in the array (line 30).

4. The screen is cleared (line 50).

5. PUT -- The circle from the array was PUT into the destination rectangular pixel area as specified in the PUT statement (line 60) with the PSET option.

```
10 DIM V%(700)
20 LINE (20,20)-(20,80)
30 LINE (80,0)-(80,80)
40 LINE (30,30)-(30,80)
50 LINE (10,5)-(10,80)
60 GET (0,0)-(100,100),V%
70 FOR I=1 TO 1000: NEXT I
80 PUT (180,120),V%,PSET
90 FOR I=1 TO 1000: NEXT I
```

Draws four lines. GET stores the lines in the rectangular pixel area. PUT moves the lines to another rectangular pixel area.

SCREEN
Selects Screen

SCREEN __type__

__type__ specifies which "Screen" to use and is a numeric expression of either 0 or 1.
0 = Graphics Screen
1 = Text Screen

SCREEN lets you set the proper screen. SCREEN 0 selects the Graphics Screen; SCREEN 1 selects the Text Screen. Any value other than 0 or 1 with SCREEN gives an error.

SCREEN is convenient to use when you want to display either a Graphics Screen or a Text Screen. For example, you may have run a program and then added to it. With SCREEN, you can remove the graphics display, add to the program, and then return to the Graphics Screen.

Whenever BASICG tries to display a character on the Text Screen (like in an INPUT or PRINT statement), the screen is automatically set to the Text Screen. If the program is still running after executing the statement, BASICG will revert to the screen that was in effect prior to executing the statement.

Examples

```
10 SCREEN 1
20 LINE (150,150)-(200,200)
```

The computer executes the short program but the Graphics Screen cannot display the graphics because of the SCREEN 1 command. To display the line, type: SCREEN 0 <ENTER>.

```
10 CLR
20 SCREEN 1
30 LINE(10,10)-(255,191)
40 LINE(0,191)-(255,0)
50 A$=INKEY$: IF A$=""THEN 50
60 SCREEN 0
70 A$=INKEY$: IF A$=""THEN 70
80 GOTO 10
```

The computer executes the program (draws two intersecting lines) but the screen cannot display the graphics because of SCREEN 1. By pressing any key, the graphics are displayed because of SCREEN 0.

```
10 CIRCLE (200,100),100
20 PAINT (200,100),"44",1
```

Now run the program and type:

```
SCREEN 0 <ENTER>
```

This command turns the Graphics Screen ON.
By entering the SCREEN 1 and SCREEN 0 commands, you can alternately turn the Graphics Screen OFF and ON without losing the executed program display.

VIEW (command)
Redefines the Screen (Creates a Viewport)

VIEW (x1,y1)-(x2,y2), c, b

(x1,y1) are coordinates of the upper-left corner of a rectangular viewport area. x1 is an integer expression between 0 and 639. y1 is an integer expression between 0 and 239.
(x2,y2) are coordinates of the lower-right corner of a rectangular viewport area. x2 is an integer expression >= to x1 and <= 639. y2 is an integer expression >= y1 and <=239.
c specifies the color of the interior of the viewport and is an integer expression of either 0 or 1. c is optional; if omitted, the viewport is not shaded.
b specifies the border color of the viewport and is an numeric expression of either 0 or 1. b is optional; if omitted, a border is not drawn.

VIEW creates a "viewport" which redefines the screen parameters (0-639 for X and 0-239 for Y). This defined area then becomes the only place you can draw graphics displays.

If you enter more than one viewport, you can only draw displays in the last defined viewport.

Since VIEW redefines the SCREEN:

- CLR clears the interior of the viewport only.
- If you PSET or PRESET points, draw circles, etc., beyond the parameters of the currently defined viewport, only the portions that are in the viewport will be displayed.
- If you try to read a point beyond the viewport (with POINT), it will return a -1.
- You can only GET and PUT arrays within the viewport.
- You can't PAINT outside the viewport.

The upper-left corner of the viewport is read as (0,0) (the "relative origin") when creating items inside the viewport. All the other coordinates are read relative to this origin. However, the "absolute coordinates" of the viewport, as they are actually defined on the Graphics Cartesian system, are retained in memory and can be read using VIEW as a function.

Every viewport has absolute and relative coordinates and graphic displays are drawn inside using the relative coordinates. For example:

```
10 VIEW (100,100)-(200,200),0,1
20 LINE (30,15)-(80,60),1
```

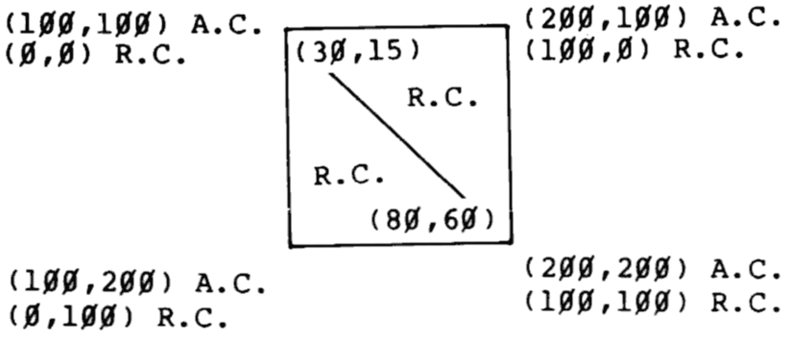


Figure 18

Note: After each of the following examples, you'll have to redefine the entire screen to VIEW(0,0)-(639,239) before performing any other Graphics functions.

Examples

```
VIEW (100,100)-(200,200),0,1
```

Draws a black viewport (pixels OFF) that is outlined in white (border pixels ON).

```
VIEW (100,100)-(200,200),1,1
```

Draws a white viewport (pixels ON) that is outlined in white (border pixels ON).

```
VIEW (50,50)-(100,100),1,0
```

Draws a white viewport (pixels ON) that is outlined in black (border pixels OFF).

```
10 VIEW (10,10)-(600,200),0,1
20 VIEW (50,50)-(100,100),0,1
30 LINE(RND(500),RND(190))-(RND(500),RND(190))
40 GOTO 30
```

First you defined a large viewport that almost covered the entire screen. Next you defined a smaller viewport. The Random command draws lines within the specified parameters but only the segments of the lines that are within the parameters of the smaller viewport are visible since it was specified last.

```
10 VIEW(80,80)-(400,200),0,1
20 VIEW(100,90)-(300,170),0,1
30 VIEW(120,100)-(200,200),0,1
40 VIEW(50,50)-(100,100),0,1
```

Draws four viewports. All further drawing takes place in the last viewport specified.

```
10 VIEW(210,80)-(420,160),0,1
20 CIRCLE(300,120),180,1
30 LINE(15,15)-(60,60),1
40 CIRCLE(90,40),50,1
50 LINE(40,30)-(500,30),1
```

Draws a viewport. Draws a circle but only a portion is within the parameters of the viewport. This circle's centerpoint is relative to the upper left corner of the viewport and not to the absolute coordinates of the graphics Cartesian system. A line is drawn which is totally within the parameters of the viewport. Another circle is drawn which is totally within the parameters of the viewport. Another line is drawn which is only partially within the parameters of the viewport.

```
10 VIEW (190,70)-(440,180),0,1
20 CIRCLE (300,140),170,1
30 CIRCLE (100,230),400,1
40 LINE (10,10)-(500,230),1
```

Draws a viewport. A circle is drawn but only a portion is within the parameters of the viewport. Another circle is drawn and a larger portion is within the parameters of the viewport. A line is drawn but only a segment is within the parameters of the viewport.

&VIEW (function)

Returns Viewport Coordinates

&VIEW(p)

(p) specifies a coordinate on the X- or Y-axis and is a integer expression between 0-3. 0 returns the left X-coordinate of your viewport. 1 returns the upper Y-coordinate. 2 returns the right X-coordinate. 3 returns the lower Y-coordinate.

&VIEW returns a corner coordinate of a viewport. It is important to note the parentheses are not optional. If you enter the &VIEW function without the parentheses, a Syntax Error will result.

To display one of the four viewport coordinates, you must enter one of the following values for p:

- 0 returns the upper left X-coordinate
- 1 returns the upper left Y-coordinate
- 2 returns the lower right X-coordinate
- 3 returns the lower right Y-coordinate

Important Note: When you have defined several viewports, &VIEW only returns the coordinates of the last-defined viewport.

Examples

Set up the following viewport:

```
VIEW(100,80)-(220,150),0,1
```

Now type: `PRINT &VIEW(0) <ENTER>`

Displays: `100`

Type: `PRINT &VIEW(1) <ENTER>`

Displays: `80`

Enter: PRINT &VIEW(2) <ENTER>

Displays: 220

Type: PRINT &VIEW(3) <ENTER>

Displays: 150

Set up the following viewports:

```
VIEW(100,80)-(220,150),0,1 <ENTER>
VIEW(250,170)-(350,220),0,1 <ENTER>
```

Now enter: PRINT &VIEW(0) <ENTER>

Displays: 250

Type: PRINT &VIEW(1) <ENTER>

Displays: 170

Now type: PRINT &VIEW(2) <ENTER>

Displays: 350

Type: PRINT &VIEW(3) <ENTER>

Displays: 220

3/ Graphics Utilities

There are six utilities included with the TRS-80 Computer Graphics package which are intended to be used as stand-alone programs. However, if you are an experienced programmer, you can use these with BASICG and FORTRAN. The source-code for each utility, that illustrate Graphics programming techniques, is listed later in this section.

The Graphics Utilities let you:

- Save graphic displays to diskette.
- Load graphic displays from diskette.
- Print graphic displays on a graphics printer.
- Turn graphics display OFF or ON.
- Clear graphics memory.

To use these utilities from BASICG, use the CMD"I" command followed by a comma and the name of the utility in quotation marks (e.g., CMD"I","GCLS" <ENTER>) and control returns to TRSDOS Ready. From TRSDOS, enter the utility directly, without quotation marks (e.g., GCLS <ENTER>).

To call these routines from FORTRAN, see the Subprogram Linkage section of your **TRS-80 Model III FORTRAN Manual** (26-2200).

Note: These utilities load into high memory starting at F000 (hex); therefore, they cannot be used with DEBUG, DO, or any communication drivers that use high memory.

Utilities

Command	Action
GCLS	Clears graphics screen.
GLOAD	**Loads graphics memory from diskette.**
GPRINT	Lists graphics on the printer.
GPRT2	**Prints graphic display on the printer without 90 degree rotation.**
GPRT3	Prints graphic display on the printer without 90 degree rotation.
GROFF	**Turns Graphic Screen OFF.**
GRON	Turns Graphic Screen ON.
GSAVE	**Saves graphics memory to diskette.**

Table 6

GCLS
Clears Graphics Screen

```
GCLS
```

GCLS clears the Graphics Screen by erasing the contents of graphics memory corresponding to the visible Graphics Screen. GCLS erases graphics memory by writing zeroes (OFF) to every bit in memory. GCLS does not clear the Text Screen (video memory).

Examples

When TRSDOS Ready is displayed, type:

```
GCLS <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GCLS" <ENTER>
```

or

```
100 CMD"I","GCLS"
```

GLOAD

Loads Graphics Memory from Diskette

GLOAD filename /ext .password :d

filename consists of a name of up to eight characters; the first character must be a letter.
/ext is an optional name-extension; ext is a sequence of up to three numbers or letters.
.password is an optional password; password is a name of up to eight characters; the first character must be a letter.
:d is an optional drive specification; d is one of the digits 0 through 3.

Note: There cannot be spaces within a file specification. TRSDOS terminates the file specification at the first space.

With GLOAD, you can load TRSDOS files that have graphic contents into graphics memory. These files must have been previously saved to diskette using GSAVE.

Examples

When TRSDOS Ready is displayed, type:

```
GLOAD PROGRAM/DAT.PASSWORD:0 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GLOAD PROGRAM" <ENTER>
```

or

```
100 CMD"I", "GLOAD PROGRAM"
```

GPRINT
Lists Graphic Display to Printer

GPRINT

GPRINT lets you print graphics memory on a graphics (dot-addressable) printer, such as Radio Shack's DMP-100 (26-1253) or DMP-200 (26-1254). Both of these printers have a 9 1/2" carriage. However, distortion will occur when Graphic routines are printed with GPRINT. This is because GPRINT is not a true pixel-by-pixel "Screen Dump" since the pixel size and spacing on the screen is different from the pixel size and spacing on the Printer. GPRINT is a point of departure for you to obtain hard-copy representations of graphics.

To print graphic displays, GPRINT turns the contents of the Graphic Screen clockwise 90 degrees and then prints.

However, FORMS must be used to set printing parameters.

Most uses will require that you set FORMS when TRSDOS Ready is displayed:

```
FORMS (LINES=60,WIDTH=0) <ENTER>
```

See your **Model III Operation and BASIC Language Reference** and printer owner's manual for more details on setting printing parameters.

Important Note! Do not press <BREAK> while GPRINT is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRINT <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRINT" <ENTER>
```

or

```
100 CMD"I","GPRINT"
```

For a complete GPRINT sample session, see Appendix D.

GPRT2
Print Graphics

```
GPRT2
```

GPRT2 is similar to GPRINT but is designed for use with wide-carriage (15") printers such as the DMP-400 and DMP-500.

GPRT2 is different from GPRINT in that the image is not rotated 90 degrees and a different aspect ratio is used.

If GPRT2 does not produce the quality of print out you desire, try GPRT3 or GPRINT.

Important Note! Do not press <BREAK> while GPRT2 is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRT2 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRT2" <ENTER>
```

or

```
100 CMD"I","GPRT2"
```

GPRT3
Print Graphics (Double on the Y-Axis)

```
GPRT3
```

GPRT3 is similar to GPRINT but is designed for use with wide-carriage (15") printers such as the DMP-400 and DMP-500.

GPRT3 is different from GPRINT in that the image is not rotated 90 degrees and a different aspect ratio is used.

If GPRT3 does not produce the quality of print-out you desire, try GPRT2 or GPRINT.

Important Note! Do not press <BREAK> while GPRT3 is executing.

Examples

When TRSDOS Ready is displayed, type:

```
GPRT3 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GPRT3" <ENTER>
```

or

```
100 CMD"I","GPRT3"
```

GROFF
Turns Graphics Display OFF

```
GROFF
```

GROFF turns the Graphics Screen OFF. GROFF is different from GCLS since GROFF simply removes the Graphics display without erasing the contents of graphic memory. GCLS completely clears graphics memory by writing zeroes (OFF) to every bit in memory.

Examples

When TRSDOS Ready is displayed, type:

```
GROFF <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
          CMD"I","GROFF" <ENTER>
or
          100 CMD"I","GROFF"
```

GRON
Turns Graphics Display ON

```
          GRON
```

GRON turns the Graphics Screen ON.

Examples

When TRSDOS Ready is displayed, type:

```
          GRON <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
          CMD"I","GRON" <ENTER>
or
          100 CMD"I","GRON"
```

GSAVE

Saves Graphics Memory to Diskette

GSAVE filename /ext .password :d

filename consists of a name of up to eight characters; the first character must be a letter.
/ext is an optional name-extension; ext is a sequence of up to three numbers or letters.
.password is an optional password; password is a name of up to eight characters; the first character must be a letter.
:d is an optional drive specification; d is one of the digits 0 through 3.

Note: There cannot be spaces within a file specification. TRSDOS terminates the file specification at the first space.

With GSAVE, the contents in graphics memory is saved under a specified filename which follows the standard TRSDOS format. To load the file back into memory, use GLOAD.

Examples

When TRSDOS Ready is displayed, type:

```
GSAVE PROGRAM/DAT.PASSWORD:0 <ENTER>
```

or when the BASICG READY prompt (>) is displayed, type:

```
CMD"I","GSAVE PROGRAM" <ENTER>
```

or

```
100 CMD"I","GSAVE PROGRAM"
```

Graphic Utilities Source Code Listings

```
; GCLS -- Clear graphics screen
;
        PSECT  0F000H
GCLS    PUSH   HL             ;Save registers
        PUSH   DE
        PUSH   BC
        CALL   INITG
        LD     A,INCY         ;Set graphics status:
        OUT    (STATUS),A     ; Graphics off, waits off, inc Y
        XOR    A
        OUT    (X),A          ;Set X & Y address to 0
        OUT    (Y),A
        LD     B,80           ;80 X addresses
OUTER   LD     C,B
        LD     B,239          ;239 Y addresses. 240th done after loop.
INNER   OUT    (WRITE),A      ;Zero graphics memory
        DJNZ   INNER          ;Go clear next Y
        LD     A,INCXY        ;Set status to inc X & Y after write
        OUT    (STATUS),A
        XOR    A
        OUT    (WRITE),A      ;and clear last (240th) Y address
        OUT    (Y),A          ;Set Y back to zero
        LD     A,INCY         ;Reset status to inc Y only
        OUT    (STATUS),A
        XOR    A
        LD     B,C
        DJNZ   OUTER          ;Go clear next X
        LD     A,0FFH         ;Set status to graphics, waits, no incs.
        OUT    (STATUS),A
        POP    BC             ;Restore registers
        POP    DE
        POP    HL
        XOR    A
        RET                   ;All done.  Go back to caller.
INCY    EQU    70H
INCXY   EQU    30H
X       EQU    80H
Y       EQU    81H
WRITE   EQU    82H
STATUS  EQU    83H
;
;   INITG --  Initialize Model III Graphics Board
;
INITG   LD     A,10H
        OUT    (236),A        ;Turn on port
        LD     BC,15
```

```
047          LD     HL,DATA
048 LOOP     LD     A,B            ;Program CRTC chip for 80 by 24
049          OUT    (136),A
050          LD     A,(HL)
051          OUT    (137),A
052          INC    HL
053          INC    B
054          LD     A,B
055          CP     C
056          JR     NZ,LOOP
057          RET
058 DATA     DEFB   99
059          DEFB   80
060          DEFB   85
061          DEFB   8
062          DEFB   25
063          DEFB   4
064          DEFB   24
065          DEFB   24
066          DEFB   0
067          DEFB   9
068          DEFB   0
069          DEFB   0
070          DEFB   0
071          DEFB   0
072          DEFB   0
073          DEFB   0
074 ;
075          END    GCLS
```

```
; GRON --  Turn on graphics display with waits on
;
       PSECT 0F000H
GRON   CALL  INITG
       LD    A,0FFH
       OUT   (STATUS),A
       XOR   A
       RET
STATUS EQU   83H
;
;  INITG --  Initialize Model III Graphics Board
;
INITG  PUSH  HL
       PUSH  BC
       PUSH  AF
       LD    A,10H
       OUT   (236),A      ;Turn on port
       LD    BC,15
       LD    HL,DATA
LOOP   LD    A,B          ;Program CRTC chip for 80 by 24
       OUT   (136),A
       LD    A,(HL)
       OUT   (137),A
       INC   HL
       INC   B
       LD    A,B
       CP    C
       JR    NZ,LOOP
       POP   AF
       POP   BC
       POP   HL
       RET
DATA   DEFB  99
       DEFB  80
       DEFB  85
       DEFB  8
       DEFB  25
       DEFB  4
       DEFB  24
       DEFB  24
       DEFB  0
       DEFB  9
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
;
```

```
050          END    GRON
```

```
; GROFF --  Turn graphics display off with waits off
;
       PSECT 0F000H
GROFF  CALL  INITG
       LD    A,0FCH
       OUT   (STATUS),A
       XOR   A
       RET
STATUS EQU   83H
;
;  INITG --  Initialize Model III Graphics Board
;
INITG  PUSH  HL
       PUSH  BC
       PUSH  AF
       LD    A,10H
       OUT   (236),A     ;Turn on port
       LD    BC,15
       LD    HL,DATA
LOOP   LD    A,B         ;Program CRTC chip for 80 by 24
       OUT   (136),A
       LD    A,(HL)
       OUT   (137),A
       INC   HL
       INC   B
       LD    A,B
       CP    C
       JR    NZ,LOOP
       POP   AF
       POP   BC
       POP   HL
       RET
DATA   DEFB  99
       DEFB  80
       DEFB  85
       DEFB  8
       DEFB  25
       DEFB  4
       DEFB  24
       DEFB  24
       DEFB  0
       DEFB  9
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
       DEFB  0
;
```

```
Ø5Ø          END    GROFF
```

```
; GSAVE --  Save graphics display to disk
;
        PSECT 0F000H
GSAVE   PUSH  HL            ;Save registers
        PUSH  DE
        PUSH  BC
        PUSH  IY
        PUSH  HL
        CALL  INITG
        LD    HL,DCBEE ;Zero DCB buffer
        LD    DE,DCBEE+1
        LD    BC,49
        LD    (HL),00H
        LDIR
        POP   HL
        LD    A,0DH
        CP    (HL)
        JP    Z,ERROR       ;Error if filespec not given
        LD    DE,DCBEE
        CALL  441CH         ;Move filespec to DCB
        JP    NZ,BOMB
        LD    HL,BUFFER
        LD    DE,DCBEE
        LD    B,0
        CALL  4420H         ;Open file
        JP    NZ,BOMB
        XOR   A
        LD    (OPNFLG),A    ;Set flag: file is open
;
        LD    A,0E3H        ;status = inc X after read
        OUT   (STATUS),A
        XOR   A
        OUT   (X),A         ;init X & Y to zero
        OUT   (Y),A
        LD    E,A           ;counter for X values
        LD    D,80          ;80 X values
        LD    B,75          ;75 disk records for entire screen
NXTREC  LD    HL,BUFFER
        LD    C,B
        LD    B,0           ;256 bytes per record
NGRPH   IN    A,(GRAPH)     ;Get next graphics byte
        LD    (HL),A        ; and put in buffer
        INC   HL
        INC   E
        LD    A,E
        CP    D
        JR    NZ,EGRPH ;Same row?
        XOR   A
        LD    E,A
```

```
0050        OUT    (X),A         ;Next row. Set X to zero
0051        LD     A,(YPOS)
0052        INC    A
0053        LD     (YPOS),A
0054        OUT    (Y),A
0055 EGRPH  DJNZ   NGRPH         ;Go get next graphics byte
0056        PUSH   DE
0057        LD     DE,DCBEE
0058        CALL   4439H         ;Write disk record
0059        POP    DE
0060        JR     NZ,BOMB
0061        LD     B,C
0062        DJNZ   NXTREC        ;Go fill buffer for next record
0063 ;
0064 EXIT   CALL   CLOSE
0065        LD     A,0FFH        ;Status = graphics, waits, no incs
0066        OUT    (STATUS),A
0067        POP    IY
0068        POP    BC
0069        POP    DE
0070        POP    HL
0071        LD     A,(EFLAG)
0072        CP     0
0073        RET                  ;All done.  Return to caller.
0074 ;
0075 ; Subroutines
0076 ;
0077 CLOSE  LD     A,(OPNFLG)
0078        OR     A
0079        RET    NZ            ;Return if file not open
0080        LD     DE,DCBEE
0081        JP     4428H         ;Go close file
0082 ;
0083 ; Error exits
0084 ;
0085 ;
0086 ERROR  LD     A,47          ;Required Command Parameter Not Found
0087 ;
0088 BOMB   LD     (EFLAG),A
0089        LD     B,A
0090        LD     A,39
0091        RST    8             ;Print "ERROR nn" message
0092        JP     EXIT
0093 ;
0094 X      EQU    80H
0095 Y      EQU    81H
0096 GRAPH  EQU    82H
0097 STATUS EQU    83H
0098 EFLAG  DEFB   0
```

```
099 YPOS    DEFB   0
100 OPNFLG  DEFB   1
101 DCBEE   DEFS   50
102 BUFFER  DEFS   256
103 ;   INITG --   Initialize Model III Graphics Board
104 ;
105 INITG   LD     A,10H
106         OUT    (236),A        ;Turn on port
107         LD     BC,15
108         LD     HL,DATA
109 LOOP    LD     A,B            ;Program CRTC chip for 80 by 24
110         OUT    (136),A
111         LD     A,(HL)
112         OUT    (137),A
113         INC    HL
114         INC    B
115         LD     A,B
116         CP     C
117         JR     NZ,LOOP
118         RET
119 DATA    DEFB   99
120         DEFB   80
121         DEFB   85
122         DEFB   8
123         DEFB   25
124         DEFB   4
125         DEFB   24
126         DEFB   24
127         DEFB   0
128         DEFB   9
129         DEFB   0
130         DEFB   0
131         DEFB   0
132         DEFB   0
133         DEFB   0
134         DEFB   0
135 ;
136         END    GSAVE
```

```
; GLOAD --  Save graphics display to disk
;
        PSECT  0F000H
GLOAD   PUSH   HL             ;Save registers
        PUSH   DE
        PUSH   BC
        PUSH   IY
        PUSH   HL
        CALL   INITG
        LD     HL,DCBEE ;Zero DCB buffer
        LD     DE,DCBEE+1
        LD     BC,49
        LD     (HL),H
        LDIR
        POP    HL
        LD     A,0DH
        CP     (HL)
        JR     Z,ERROR
        LD     DE,DCBEE
        CALL   441CH          ;Move filespec to DCB
        JR     NZ,BOMB
        LD     HL,BUFFER
        LD     DE,DCBEE
        LD     B,0
        CALL   4424H          ;Open file
        JP     NZ,BOMB
        XOR    A
        LD     (OPNFLG),A     ;Set flag: file is open
;
        LD     A,0B3H         ;status = inc X after write
        OUT    (STATUS),A
        XOR    A
        OUT    (X),A          ;init X & Y to zero
        OUT    (Y),A
        LD     E,A            ;counter for X values
        LD     D,80           ;80 X values
        LD     B,75           ;75 disk records for entire screen
NXTREC  PUSH   DE
        LD     DE,DCBEE
        CALL   4436H          ;Read record from disk
        POP    DE
        JR     NZ,BOMB
        LD     HL,BUFFER
        LD     C,B
        LD     B,0            ;256 bytes per record
NGRPH   LD     A,(HL)
        OUT    (GRAPH),A
        INC    HL
        INC    E
```

```
0050        LD     A,E
0051        CP     D
0052        JR     NZ,EGRPH        ;Same row?
0053        XOR    A
0054        LD     E,A
0055        OUT    (X),A           ;Next row. Set X to zero
0056        LD     A,(YPOS)
0057        INC    A
0058        LD     (YPOS),A
0059        OUT    (Y),A
0060 EGRPH  DJNZ   NGRPH           ;Go get next graphics byte
0061        LD     B,C
0062        DJNZ   NXTREC          ;Go read next disk record
0063 ;
0064 EXIT   CALL   CLOSE
0065        LD     A,0FFH          ;Status = graphics, waits, no incs.
0066        OUT    (STATUS),A
0067        POP    IY
0068        POP    BC
0069        POP    DE
0070        POP    HL
0071        LD     A,(EFLAG)
0072        CP     0
0073        RET
0074 ;
0075 ; Subroutines
0076 ;
0077 CLOSE  LD     A,(OPNFLG)
0078        OR     A
0079        RET    NZ              ;Return if file not open
0080        LD     DE,DCBEE
0081        JP     4428H           ;Go close file
0082 ;
0083 ; Error exits
0084 ;
0085 ERROR  LD     A,47            ;Required Command Parameter Not Found
0086 ;
0087 BOMB   LD     (EFLAG),A
0088        LD     B,A
0089        LD     A,39
0090        RST    8               ;Print "ERROR nn" message
0091        JP     EXIT
0092 ;
0093 X      EQU    80H
0094 Y      EQU    81H
0095 GRAPH  EQU    82H
0096 STATUS EQU    83H
0097 EFLAG  DEFB   0
0098 YPOS   DEFB   0
```

```
099 OPNFLG DEFB  1
100 DCBEE  DEFS  50
101 BUFFER DEFS  256
102 ;
103 ;   INITG --  Initialize Model III Graphics Board
104 ;
105 INITG  LD    A,10H
106        OUT   (236),A      ;Turn on port
107        LD    BC,15
108        LD    HL,DATA
109 LOOP   LD    A,B          ;Program CRTC chip for 80 by 24
110        OUT   (136),A
111        LD    A,(HL)
112        OUT   (137),A
113        INC   HL
114        INC   B
115        LD    A,B
116        CP    C
117        JR    NZ,LOOP
118        RET
119 DATA   DEFB  99
120        DEFB  80
121        DEFB  85
122        DEFB  8
123        DEFB  25
124        DEFB  4
125        DEFB  24
126        DEFB  24
127        DEFB  0
128        DEFB  9
129        DEFB  0
130        DEFB  0
131        DEFB  0
132        DEFB  0
133        DEFB  0
134        DEFB  0
135 ;
136        END   GLOAD
```

```
; GPRINT --  Print graphics screen to graphics printer
;
        PSECT 0F000H
GPRINT  PUSH  HL              ;Save registers
        PUSH  DE
        PUSH  BC
        PUSH  IX
        CALL  INITG
        OR    0DBH            ;Output a Control byte to cause
        OUT   (STATUS),A      ; Y to automatically dec. on a read
        CALL  INITBF
;
        XOR   A               ;Set A to 0
        OUT   (X),A           ;Initialize the X position
        LD    (BPOS),A ;       "       "  bit position
        LD    (XLOC),A ;       "       "  "  location counter
        LD    HL,BGMODE
        LD    B,1
        LD    C,0DH
        CALL  PRLINE          ;Begin graphics print mode
;
LOOP1   LD    IX,BUFFER       ;point IX at the printer buffer
        LD    B,240           ;go through a whole column of bytes
        LD    A,B             ;Put value in A and decrement
        DEC   A               ;  so it can be put out as
        OUT   (Y),A           ;    the Y position
COLUMN  LD    HL,MASK         ;point HL at the mask byte
        IN    A,(GRAPH)       ;input a graphics byte
        AND   (HL)            ;chop off all but proper bit
        CALL  PO,SET0         ;if result is odd parity set bit 0
                              ; otherwise bit A is 0
        LD    HL,BPOS         ;point HL at the bit position
        PUSH  BC              ;save register B (for DJNZ loop)
        LD    B,(HL)          ;get count
        INC   B               ;increment (in case it is 0)
DECJ    DEC   B               ;move bit left BPOS number of times
        JR    Z,PAST          ;if done, move on...
        RLC   A               ;move bit left one position
        JR    DECJ            ;repeat loop
PAST    POP   BC              ;get loop counter back
        OR    (IX)            ;merge A with byte of printer buffer
        LD    (IX),A          ;put merged result in buffer
        INC   IX              ;increment buffer pointer
        DJNZ  COLUMN          ;continue loop
;-----------------------------------------------------------------
        LD    A,7             ;See if BPOS has gotten to 8.
        INC   (HL)            ;  If it has (printer uses 7 bits)
        CP    (HL)            ;    print the buffer and reset
        CALL  Z,PRNDRS ;         BPOS to 0
```

```
0050 ;
0051        LD     HL,MASK       ;After getting a vertical row of bits
0052        RRC    (HL)          ; rotate the mask right one position
0053        LD     A,80H         ;Check to see if its back to
0054        CP     (HL)          ; it's original value, if not
0055        JR     NZ,LOOP1 ; go get another row of bits
0056        LD     A,(XLOC) ;If so, get X pos (to increment it)
0057        CP     79            ;Check to see if we are at the end...
0058        JP     Z,BYE
0059        INC    A             ;otherwise increment the X counter
0060        LD     (XLOC),A ;and store it back
0061        OUT    (X),A         ;also update the port value
0062        JR     LOOP1         ;now go get another row of bits
0063 ;-----------------------------------------------------------------
0064 SET0   LD     A,1           ;set A to binary 0000 0001
0065        RET                  ; and return
0066 ;
0067 PRNDRS LD     HL,BUFFER     ;Set up the
0068        LD     B,240         ;  PRLINE call and
0069        LD     C,0DH         ;    send the buffer
0070        CALL   PRLINE
0071        XOR    A             ;clear A
0072        LD     (BPOS),A ;reset bit position counter
0073 ;
0074 INITBF LD     HL,BUFFER     ;Initialize the printer buffer
0075        LD     DE,BUFFER+1   ;  with all 80H
0076        LD     BC,239
0077        LD     A,80H
0078        LD     (HL),A
0079        LDIR
0080        RET
0081 ;---------------------------------------------------------------
0082 PRLINE EQU    $             ;Print a line. HL==>line to print
0083        LD     A,(HL)        ;B = # characters to print
0084        INC    HL            ;C = EOL char (sent after line)
0085        CALL   3BH           ;HL, BC, AF, and DE used
0086        DJNZ   PRLINE
0087        LD     A,C
0088        CALL   3BH
0089        RET
0090 ;---------------------------------------------------------------
0091 BYE    CALL   PRNDRS
0092        LD     HL,EGMODE
0093        LD     B,1
0094        LD     C,0DH
0095        CALL   PRLINE        ;End graphics print mode
0096        POP    IX            ;Restore registers
0097        POP    BC
0098        POP    DE
```

```
        POP    HL
        XOR    A
        RET
X       EQU    80H
Y       EQU    81H
GRAPH   EQU    82H
STATUS  EQU    83H
MASK    DEFB   80H             ;Mask to use in extracting bits
BGMODE  DEFB   12H             ;Control byte: start graphics mode
BUFFER  DEFS   240             ;Printer data buffer
EGMODE  DEFB   1EH             ;Control byte: end graphics mode
BPOS    DEFB   0               ;Bit position in printer buffer
XLOC    DEFB   0               ;Current X location value
;
;
;   INITG --  Initialize Model III Graphics Board
;
INITG   LD     A,10H
        OUT    (236),A         ;Turn on port
        LD     BC,15
        LD     HL,DATA
LOOP    LD     A,B             ;Program CRTC chip for 80 by 24
        OUT    (136),A
        LD     A,(HL)
        OUT    (137),A
        INC    HL
        INC    B
        LD     A,B
        CP     C
        JR     NZ,LOOP
        RET
DATA    DEFB   99
        DEFB   80
        DEFB   85
        DEFB   8
        DEFB   25
        DEFB   4
        DEFB   24
        DEFB   24
        DEFB   0
        DEFB   9
        DEFB   0
        DEFB   0
        DEFB   0
        DEFB   0
        DEFB   0
        DEFB   0
;
        END    GPRINT
```

NOTES

```
;  GPRT2 -- Print graphics X horizontal
;
        PSECT  0F000H
GPRT2   PUSH   HL
        PUSH   DE
        PUSH   BC
        PUSH   IX
        CALL   INITG
        LD     HL,BGMODE     ;Turn on graphics print mode
        LD     B,1
        LD     C,0DH
        CALL   PRLINE
        LD     C,0           ;Graphics Y address
        LD     A,0E3H
        OUT    (STATUS),A
;
NEWLN   PUSH   BC
        LD     HL,BUF        ;Clear buffer
        LD     DE,BUF+1
        LD     BC,639
        LD     A,80H
        LD     (HL),A
        LDIR
;
        POP    BC
        LD     D,1           ;Bit in buf to set
;
NEWRW   LD     A,C
        OUT    (Y),A         ;Update Y address
        INC    C
        LD     HL,BUF
        XOR    A
        OUT    (X),A         ;Restart X address
        LD     B,80          ;Get 80 graphics bytes
;
BYTE    PUSH   BC            ;Save Y & loop counter
        IN     A,(GRAPH)
        LD     C,A           ;Save graphics byte in C
        LD     E,80H         ;Get bits left to right
BIT     LD     A,C
        AND    E
        JR     Z,OFF1
        LD     A,D
        OR     (HL)
        LD     (HL),A        ;Set bit in buffer
OFF1    INC    HL            ;Next buffer byte
        SRL    E             ;Next bit
```

```
00048        JR     NZ,BIT
00049        POP    BC
00050        DJNZ   BYTE
00051 ;
00052        LD     A,240
00053        CP     C               ;Last Y address?
00054        JR     Z,DONE
00055        SLA    D               ;Next bit in buffer
00056        JP     P,NEWRW
00057 ;
00058        CALL   PRINT           ;Print buffer
00059        JR     NEWLN
00060 ;
00061 DONE   CALL   PRINT
00062        LD     A,0FCH
00063        OUT    (STATUS),A
00064        LD     HL,EGMODE       ;Turn off graphics print
00065        LD     B,1
00066        LD     C,0DH
00067        CALL   PRLINE
00068        POP    IX
00069        POP    BC
00070        POP    DE
00071        POP    HL
00072        XOR    A
00073        RET
00074 ;
00075 PRINT  PUSH   BC
00076        LD     DE,0            ;Offset for print buffer
00077 PART   LD     HL,BUF
00078        ADD    HL,DE
00079        XOR    A
00080        CP     (HL)            ;End of buffer?
00081        JR     Z,EPRT
00082        LD     BC,(CTL)
00083        CALL   PRLINE
00084        LD     HL,214
00085        ADD    HL,DE           ;Next part of buffer
00086        EX     DE,HL
00087        JR     PART
00088 EPRT   POP    BC
00089        RET
00090 ;
00091 PRLINE EQU    $               ;Print a line. HL==>line to print
00092        PUSH   DE
00093 PRL2   LD     A,(HL)          ;B = # characters to print
00094        INC    HL              ;C = EOL char (sent after line)
00095        CALL   3BH             ;HL, BC, AF, and DE used
00096        DJNZ   PRL2
```

```
00097        LD     A,C
00098        OR     A
00099        CALL   NZ,3BH
00100        POP    DE
00101        RET
00102 ;
00103 ;  INITG --  Initialize Model III Graphics Board
00104 ;
00105 INITG  LD     A,10H
00106        OUT    (236),A        ;Turn on port
00107        LD     BC,15
00108        LD     HL,DATA
00109 LOOP   LD     A,B            ;Program CRTC chip for 80 by 24
00110        OUT    (136),A
00111        LD     A,(HL)
00112        OUT    (137),A
00113        INC    HL
00114        INC    B
00115        LD     A,B
00116        CP     C
00117        JR     NZ,LOOP
00118        RET
00119 DATA   DEFB   99
00120        DEFB   80
00121        DEFB   85
00122        DEFB   8
00123        DEFB   25
00124        DEFB   4
00125        DEFB   24
00126        DEFB   24
00127        DEFB   0
00128        DEFB   9
00129        DEFB   0
00130        DEFB   0
00131        DEFB   0
00132        DEFB   0
00133        DEFB   0
00134        DEFB   0
00135 ;
00136 BGMODE DEFB   12H
00137 EGMODE DEFB   1EH
00138 CTL    DEFB   0              ;Print 214 char, followed by null
00139        DEFB   214
00140 BUF    DEFS   640
00141        DEFB   0              ;Filler
00142        DEFB   0DH            ;Carriage return
00143        DEFB   0              ;End of buffer signal
00144 X      EQU    80H
00145 Y      EQU    81H
```

```
00146 GRAPH  EQU   82H
00147 STATUS EQU   83H
00148 ;
00149        END   GPRT2
```

```
;  GPRT3 -- Print graphics X horizontal double Y axis
;
       PSECT 0F000H
GPRT3  PUSH  HL
       PUSH  DE
       PUSH  BC
       PUSH  IX
       CALL  INITG
       LD    HL,BGMODE     ;Turn on graphics print
       LD    B,1
       LD    C,0DH
       CALL  PRLINE
       LD    C,0           ;Graphics Y address
       LD    A,0E3H
       OUT   (STATUS),A
       LD    D,3           ;Bit(s) in buf to set
;
NEWLN  PUSH  BC
       PUSH  DE
       LD    HL,BUF        ;Clear buffer
       LD    DE,BUF+1
       LD    BC,639
       LD    A,80H
       LD    (HL),A
       LDIR
;
       POP   DE
       POP   BC
;
NEWRW  LD    A,C
       OUT   (Y),A         ;Update Y address
       LD    A,40H
       CP    D
       JR    Z,NEWR1       ;If printing row second time
       INC   C             ;Move to next row
NEWR1  LD    HL,BUF
       XOR   A
       OUT   (X),A         ;Restart X address
       LD    B,80          ;Get 80 graphics bytes
       LD    A,4
       CP    D
       JR    NZ,BYTE
       LD    D,6
;
BYTE   PUSH  BC            ;Save Y & loop counter
       IN    A,(GRAPH)
       LD    C,A           ;Save graphics byte in C
       LD    E,80H         ;Get bits left to right
```

```
00049 BIT    LD     A,C
00050        AND    E
00051        JR     Z,OFF1
00052        LD     A,D
00053        OR     (HL)
00054        LD     (HL),A          ;Set bit in buffer
00055 OFF1   INC    HL              ;Next buffer byte
00056        SRL    E               ;Next bit
00057        JR     NZ,BIT
00058        POP    BC
00059        DJNZ   BYTE
00060 ;
00061        LD     A,240
00062        CP     C               ;Last Y address?
00063        JR     Z,DONE
00064        SLA    D               ;Next bit in buffer
00065        SLA    D
00066        JR     Z,ENDRW
00067        JP     P,NEWRW
00068        LD     A,7FH
00069        AND    D
00070        LD     D,A
00071        JR     NZ,NEWRW
00072        LD     D,3
00073        JR     ENDR2
00074 ;
00075 ENDRW  LD     D,1
00076 ENDR2  PUSH   DE
00077        CALL   PRINT           ;Print buffer
00078        POP    DE
00079        JR     NEWLN
00080 ;
00081 DONE   CALL   PRINT
00082        LD     A,0FCH
00083        OUT    (STATUS),A
00084        LD     HL,EGMODE       ;Turn off graphics print
00085        LD     B,1
00086        LD     C,0DH
00087        CALL   PRLINE
00088        POP    IX
00089        POP    BC
00090        POP    DE
00091        POP    HL
00092        XOR    A
00093        RET
00094 ;
00095 PRINT  PUSH   BC
00096        LD     DE,0            ;Offset for print buffer
00097 PART   LD     HL,BUF
```

```
00098          ADD    HL,DE
00099          XOR    A
00100          CP     (HL)           ;End of buffer?
00101          JR     Z,EPRT
00102          LD     C,(CTL)
00103          CALL   PRLINE
00104          LD     HL,214
00105          ADD    HL,DE          ;Next part of buffer
00106          EX     DE,HL
00107          JR     PART
00108 EPRT     POP    BC
00109          RET
00110 ;
00111 PRLINE   EQU    $              ;Print a line. HL==>line to print
00112          PUSH   DE
00113 PRL2     LD     A,(HL)         ;B = # characters to print
00114          INC    HL             ;C = EOL char (sent after line)
00115          CALL   3BH            ;HL, BC, AF, and DE used
00116          DJNZ   PRL2
00117          LD     A,C
00118          OR     A
00119          CALL   NZ,3BH
00120          POP    DE
00121          RET
00122 ;
00123 ;   INITG --  Initialize Model III Graphics Board
00124 ;
00125 INITG    LD     A,10H
00126          OUT    (236),A        ;Turn on port
00127          LD     BC,15
00128          LD     HL,DATA
00129 LOOP     LD     A,B            ;Program CRTC chip for 80 by 24
00130          OUT    (136),A
00131          LD     A,(HL)
00132          OUT    (137),A
00133          INC    HL
00134          INC    B
00135          LD     A,B
00136          CP     C
00137          JR     NZ,LOOP
00138          RET
00139 DATA     DEFB   99
```

```
00140        DEFB  80
00141        DEFB  85
00142        DEFB  8
00143        DEFB  25
00144        DEFB  4
00145        DEFB  24
00146        DEFB  24
00147        DEFB  0
00148        DEFB  9
00149        DEFB  0
00150        DEFB  0
00151        DEFB  0
00152        DEFB  0
00153        DEFB  0
00154        DEFB  0
00155 ;
00156 BGMODE DEFB  12H
00157 EGMODE DEFB  1EH
00158 CTL    DEFB  0
00159        DEFB  214          ;Print 214 char, followed by null
00160 BUF    DEFS  640
00161        DEFB  0            ;Filler
00162        DEFB  0DH          ;Carriage return
00163        DEFB  0            ;End of buffer signal
00164 X      EQU   80H
00165 Y      EQU   81H
00166 GRAPH  EQU   82H
00167 STATUS EQU   83H
00168 ;
00169        END   GPRT3
```

4/ Graphics Subroutine Library (FORTRAN)

The Graphics Subroutine Library included on the Computer Graphics diskette lets you use the functions of TRS-80 Computer Graphics while programming in Model III FORTRAN (26-2200). This library (GRPLIB/REL) must be linked to any FORTRAN program that accesses the Graphics Subroutines.

BASICG vs. the Graphics Subroutine Library

The Graphics Subroutine Library contains subroutines which provide the same capabilities as the Graphics commands and functions in BASICG. The Graphics subroutines have basically the same names and parameters as the BASICG commands. The major differences between the Library subroutines and the BASICG commands are:

- The BASICG command LINE has three corresponding library subroutines: LINE, LINEB, and LINEBF. LINEB and LINEBF provide the functions of the BASICG command LINE with the parameters B and BF respectively.
- The BASICG command PAINT has two corresponding library subroutines: PAINT and PAINTT. PAINT is for painting solid black or white, and PAINTT is for painting with tiling.
- The Library subroutines that correspond to BASICG commands that use (x,y) coordinates (except for VIEW) use (x,y) coordinates that have been previously set. The subroutines used to set the coordinates are SETXY and SETXYR.

Setting Points Using SETXY and SETXYR

The coordinates specified by SETXY or SETXYR will be called the "current" and "previous" coordinates. Subroutines that use one (x,y) coordinate pair use the "current" coordinates and subroutines that use two (x,y) pairs use both the "current" and the "previous" coordinates. Each call to SETXY or SETXYR sets the coordinates as follows:

1. Assign the values of the "current" (x,y) coordinates to the "previous" (x,y) coordinates, (discarding the old "previous" coordinates).

2. Assign new values for the "current" (x,y) coordinates as specified by the arguments supplied. SETXY simply sets the "current" coordinates to the values of its arguments. SETXYR adds the values of its arguments to the "current" coordinates to obtain the new coordinates.

Initialization

Before any calls are made to Graphics, the Graphics library and board must be initialized. A special initialization routine (GRPINI) is included in the library. A call to GRPINI must be made as the first access to the Graphics library.

Example

```
00100  C     SAMPLE INITIALIZATION
00150        DIMENSION V(30,30)
00200        CALL GRPINI(0)
```

Linking

The Library (GRPLIB/REL) must be linked to any programs that access the Graphics Subroutines. You must use the linker (L80) to generate the load module.

Example

```
L80 <ENTER>
*SAMPLE:1-N
*GRPHSAM,GRPLIB-S,FORLIB-S,-U
*-E
```

This example links both the Graphics Library and the FORTRAN Subroutine Library to the relocatable file GRPHSAM/REL. In this example, SAMPLE:1-N is the file name, drive specification, and switch, respectively; GRPHSAM, GRPLIB-S, FORLIB-S, and -U are the names of the relocatable modules to be linked and their respective switches. -E ends the routine and creates the executable program SAMPLE. The *'s in the example are prompts for the user -- not data to be entered.

Note: If there are unresolved external references, the FORTRAN Library may need to be scanned a second time.

Errors

If you enter incorrect parameters for any of the Graphics Subroutines, your screen will display:

GRAPHICS ERROR

and return program control to TRSDOS Ready. This is the only error message you'll get when executing the Subroutines.

Important Note: Free memory is utilized by the Graphic Routine for temporary storage. Extreme care should be exercised if your program accesses this memory.

Routines/Functions

Most of the FORTRAN Subroutines and functions described in this section have a corresponding command in the Graphics BASIC Language Reference section of this manual.

FORTRAN Routines

Routine	Action
CIRCLE	Draws a circle, arc, semicircle, or ellipse.
CLS	Clears the Graphics Screen.
GET	Reads the contents of a rectangular pixel area into an array.
GPRINT	Displays textual data on the Graphics Screen.
GRPINI	Graphics initialization routine.
LINE	Draws a line.
LINEB	Draws a box.
LINEBF	Draws a filled box.
LOCATE	Sets the direction for displaying textual data on the Graphics Screen.
PAINT	Paints the screen in specified OFF/ON color.
PAINTT	Paints the screen in a specified pattern.
PRESET	Sets pixel OFF/ON.
PSET	Sets pixel OFF/ON.
PUT	Puts the stored array on the screen.
SCREEN	Selects the screen.
SETXY	Sets (x,y) coordinates (absolute).
SETXYR	Sets (x,y) coordinates (relative).
VIEW	Sets up a viewport where graphics is displayed.

Table 7

FORTRAN Functions

Function	Action
POINT	Reads a pixel's value at a specified coordinate.
FVIEW	Reads a viewport's parameters.

Table 8

CIRCLE

Draws a Circle, Arc, Semicircle, Point or Ellipse

CIRCLE (radius,color,start,end,ar)

radius is of INTEGER type and specifies the radius of the circle.
color is of LOGICAL type, specifies the OFF/ON color of the border of the circle and is a integer expression of either 0 or 1.
start is of REAL type and specifies the startpoint of the circle.
end is of REAL type and specifies the endpoint of the circle.
ar is the aspect ratio, is of REAL type and determines the major axis of the circle. If ar is 0, 0.5 is used.

CIRCLE draws a circle. By varying start, end, and aspect ratio, you can draw arcs, semicircles, or ellipses using current X- and Y-coordinates as the centerpoint (set by SETXY or SETXYR).

If start and end are 0.0, a circle is drawn starting from the center right side of the circle. Note: In the CIRCLE statement, end is read as 2 x PI even though you have entered 0.0. If you enter 0.0 for aspect ratio, a symmetric circle is drawn.

Example

```
CALL CIRCLE(100,1,0.0,0.0,0.0)
```

Sample Program

This example draws and paints a circle.

```
00010  C    SAMPLE PROGRAM FOR CIRCLE
00020       LOGICAL COLOR,OPTION
00030       COLOR=1
00040       OPTION=0
00050       CALL GRPINI(OPTION)
00060       CALL CLS
00070       CALL SETXY(300,100)
00080       CALL CIRCLE(100,COLOR,0.0,0.0,0.0)
00090       CALL PAINT(COLOR,COLOR)
00100       END
```

CLS
Clears Graphics Screen

CLS

Example

```
CALL CLS
```

Sample Program (see CIRCLE)

GET
Reads Contents of a Rectangular Pixel Area into an Array

GET (array,size)

array is any type and is the name of the array you specify.
size is of INTEGER type and specifies the size of the array in terms of bytes.

GET reads the contents of a rectangular pixel area into an array for future use by PUT. The pixel area is a group of pixels which are defined by the current x and y, and the previous X- and Y-coordinates specified by the SETXY call.

The first two bytes of array are set to the horizontal (X-axis) number of pixels in the pixel area; the second two bytes are set to the vertical (Y-axis) number of pixels in the pixel area. The remainder of array represents the status of each pixel (either ON or OFF) in the pixel area. The data is stored in a row-by-row format. The data is stored eight pixels per byte and each row starts on a byte boundary.

Array Limits

When the array is defined, space is reserved in memory for each element of the array. The size of the array is limited by the amount of memory available for use by your program -- each real number in your storage array uses four memory locations (bytes).

The array must be large enough to hold your graphic display and the rectangular area defined must include all the points you want to store.

To determine the minimum array size:

1. Divide the number of X-axis pixels by 8 and round up to the next higher integer.
2. Multiply the result by the number of Y-axis pixels.

   When counting the X-Y axis pixels, be sure to include the first and last pixel.
3. Add four to the total.
4. Divide by four (for real numbers) and two (for integers) rounding up to the next higher integer. (Note: If you're using a LOGICAL array, the result of Step #3 above will produce the desired array size.)

When using arrays, the position and size of the rectangular pixel area is determined by the current and previous (x,y) coordinates.

Position: upper left corner = startpoint = (x1,y1)
lower left corner = endpoint = (x2,y2)

Size (in pixels): width = x2-x1+1
length = y2-y1+1

Example

```
CALL GET(A,4000)
```

Sample Program

This example draws a circle, saves the circle into an array, then restores the array to the graphics video.

```
00050  C     SAMPLE FOR GET AND PUT
00100        LOGICAL V(128),ACTION
00150                 ACTION=1
00200        CALL GRPINI(0)
00300        CALL CLS
00350  C     DRAW A CIRCLE
00400        CALL SETXY(30,30)
00500        CALL CIRCLE(10,1,0.0,0.0,0.0)
00550  C     SET COORDINATES FOR GET ARRAY
00600        CALL SETXY(10,10)
00700        CALL SETXY(40,40)
00750  C     STORE GRAPHICS INTO ARRAY WITH GET
00800        CALL GET(V,128)
00900        DO 10 I=1,5000
01000  10    CONTINUE
01050  C     CLEAR SCREEN AND RESTORE GRPH FROM ARRAY
01100        CALL CLS
01200        CALL SETXY(110,110)
01300        CALL PUT(V,ACTION)
01400        DO 20 I=1,5000
01500  20    CONTINUE
01600        END
```

GPRINT
Write Text Characters to the Graphics Screen

GPRINT (cnt,array)

cnt is of INTEGER type and specifies the number of characters to display.
array is a one dimensional LOGICAL array containing the characters to be displayed.

GPRINT is used to write text characters to the Graphics Screen. This is the easiest way to display textual data on the Graphics Screen. Characters are displayed starting at the current (x,y) coordinates and going in the direction specified by the most recently executed LOCATE call. If no LOCATE call was executed prior to the GPRINT call, a direction of 0 is assumed.

GPRINT will only print text characters (see Appendix C of the **Model III Operation and BASIC Language Reference Manual**). Each character displayed in the 0 or 2 direction uses an 8 X 8 pixel grid; each character displayed in the 1 or 3 direction uses a 16 X 8 grid. Executing this command will set the current (x,y) coordinates to the end of the last character that was displayed.

Displaying text in direction 0 engages a wraparound feature. If the end of a line is reached, the display will be continued on the next line. If the end of the screen is reached, the display will be continued at the beginning of the screen without scrolling. If there is not enough room to display at least one character at the current (x,y) coordinates, a GRAPHICS ERROR will result. When displaying text in other directions, an attempt to display text outside the currently defined screen will cause a GRAPHICS ERROR to be given.

GRPINI
Graphics Initialization Routine

GRPINI(option)

option is of LOGICAL type; 0 clears the Graphics Screen, non-zero does not clear the Graphics Screen.

GRPINI is the graphics initialization routine. This function must be called before any other graphics calls are made in FORTRAN.

Example

```
CALL GRPINI(1)
```

Sample Program (see CIRCLE)

LINE
Draws Line

LINE (color, style)

color is of LOGICAL type, specifies the OFF/ON color of a line and is an integer expression of either 0 (OFF, black) or 1 (ON, white).
style is of INTEGER type, specifies the pattern of the line and is a number in the integer range. -1 indicates a solid line.

LINE draws a line between the previous and current coordinates. These coordinates are set by the SETXY or SETXYR subroutines.

Example

```
CALL LINE (1,-1)
```

Sample Program

This example draws a diagonal line connected to a box, which is connected to a filled box.

```
00010  C    SAMPLE FOR LINE LINEB LINEBF
00020       LOGICAL COLOR
00030       COLOR=1
00040       CALL GRPINI(0)
00050       CALL CLS
00060       CALL SETXY(1,1)
00070       CALL SETXY(210,80)
00080       CALL LINE(COLOR,-1)
00090       CALL SETXY(420,160)
00100  C    COORDINATES ARE NOW (210,80) (420,160)
00110       CALL LINEB(COLOR,-1)
00120       CALL SETXY(639,239)
00130  C    COORDINATES ARE NOW (420,160) (639,239)
00140       CALL LINEBF(COLOR)
00150       END
```

LINEB
Draws Box

LINEB (color, style)

color is of LOGICAL type, specifies the OFF/ON color of a line and is a integer expression of either 0 (OFF, black) or 1 (ON, white).
style is of INTEGER type and specifies the pattern of the line. -1 indicates a solid line.

LINEB is the same as LINE except LINEB draws a box between the two sets of coordinates set by the SETXY or SETXYR subroutines.

Example

```
CALL LINEB (1,-1)
```

Sample Program (see LINE)

LINEBF
Draws Painted Box

LINEBF (color)

color is of LOGICAL type, specifies the OFF/ON color of a line and is an integer expression of either 0 (OFF, black) or 1 (ON, white).

LINEBF is the same as LINEB except LINEBF fills the box (colors in the box) and the argument style is not used.

Example

```
CALL LINEBF (1)
```

Sample Program (see LINE)

LOCATE

Sets the Direction for Displaying Text on the Graphics Screen

LOCATE (direction)

direction is of LOGICAL type, specifies the direction that GLOCATE will use to display textual data and is an integer expression of 0-3.

LOCATE sets the direction that GPRINT will use to display textual data. The allowable values for direction are:

0 - zero degree angle
1 - 90 degree angle
2 - 180 degree angle
3 - 270 degree angle

Examples

```
CALL LOCATE (0)
```

This program line will cause characters to be displayed at the current (x,y) coordinates in normal left to right orientation.

```
CALL LOCATE (1)
```

This program line will cause characters to be displayed at the current (x,y) coordinates in a vertical orientation going from the top of the screen to the bottom of the screen.

```
CALL LOCATE (2)
```

This program line will cause characters to be displayed upside down starting at the right of the screen and going towards the left.

```
CALL LOCATE (3)
```

This program line will cause the characters to be displayed vertically starting at the lower portion of the screen going towards the top of the screen.

PAINT
Paints Screen in Specified Color

PAINT (<u>color</u>, <u>border</u>)

<u>color</u> is of LOGICAL type, specifies the OFF/ON color of painting and is an integer expression of either 0 (OFF, black) or 1 (ON, white).
<u>border</u> is of LOGICAL type, specifies the OFF/ON color of the border and is an integer expression of either 0 (OFF, black) or 1 (ON, white).

PAINT paints the screen in the specified OFF/ON <u>color</u> (black or white). It uses the current X- and Y-coordinates (see SETXY) as its startpoint.

Example

```
CALL PAINT(1,1)
```

Sample Program (see CIRCLE)

PAINTT
Paints Screen in Specified Pattern

PAINTT (arrayT, border, arrayS)

arrayT is a byte array which defines a multi-pixel pattern to be used when painting (tiling). The first byte of arrayT indicates the length of the "tile" (number of bytes).
border is of LOGICAL type and specifies the color of the border. border is an integer expression of either 0 (black) or 1 (white).
arrayS is a byte array that is used to define the background. The first byte is always set to 1; the second byte describes the background you are painting on (X'FF' = white, X'00' = black).

PAINTT lets you paint a precisely defined pattern using a graphics technique called "tiling." You can paint with tiling by defining a multi-pixel grid in an array and then using that array as the paint pattern.

Example

```
CALL PAINTT (A,1,V)
```

Sample Program

```
00100  C     EXAMPLE FOR PAINT WITH TILE
00150        LOGICAL A,B,BORDER
00200        DIMENSION A(9)
00300        DIMENSION B(2)
00350  C     DEFINE TILE ARRAY HERE
00400        DATA A(1), A(2), A(3) / 8, X'81', X'42'/
00500        DATA A(4),A(5),A(6)/X'24',X'18',X'18'/
00600        DATA A(7),A(8),A(9)/X'24',X'42',X'81'/
00650  C     DEFINE BACKGROUND ARRAY HERE
00700        DATA  B(1),B(2)/1,0/
00800        CALL GRPINI(0)
00900        CALL CLS
01000        CALL SETXY(300,100)
01100        CALL CIRCLE(150,1,0.0,0.0,0.0)
01200        BORDER=1
01300        CALL PAINTT(A,BORDER,B)
01400        END
```

PRESET
Sets Pixel ON/OFF

PRESET (<u>color</u>)

<u>color</u> is of LOGICAL type, specifies whether a pixel is to be set ON or OFF and is an integer expression of either 0 (OFF) or 1 (ON).

PRESET sets the pixel defined by the current (x,y) coordinates either ON or OFF.

Example

```
CALL PRESET(0)
```

Sample Program

```
00100   C       PRESET EXAMPLE
00200           LOGICAL COLOR
00300           COLOR=1
00400           CALL GRPINI(0)
00500           CALL CLS
00600   C       SET PIXEL TO ON
00600           CALL SETXY(300,120)
00800           CALL PRESET(COLOR)
00900   C       TEST PIXEL WHETHER ON OR OFF
01000           K=POINT(M)
01100   30      WRITE (3,35)K
01200   35      FORMAT ('2','PIXEL VALUE IS',I4)
01300           END
```

PSET
Sets Pixel ON/OFF

> **PSET (<u>color</u>)**
>
> **<u>color</u> is of LOGICAL type, specifies whether a pixel is to be set ON or OFF and is an integer expression of either 0 (OFF) or 1 (ON).**

PSET sets the pixel defined by the current (x,y) coordinates either ON or OFF.

Example

```
CALL PSET(0)
```

Sample Program

```
00100  C     PSET EXAMPLE
00200        LOGICAL COLOR
00300        LOGICAL POINT
00400        COLOR=1
00500        CALL GRPINI(0)
00600        CALL CLS
00700  C     SET PIXEL TO ON
00800        CALL SETXY(300,120)
00900        CALL PSET(COLOR)
01000  C     TEST PIXEL WHETHER ON OR OFF
01100        K=POINT(M)
01200        WRITE (3,35)K
01300  35    FORMAT ('2','PIXEL VALUE IS',I4)
01400        END
```

PUT
Puts Stored Array onto Screen

PUT (array, action)

array is usually LOGICAL type, although any type is permissible. Specifies the array (stored with GET) to be restored.
action is of LOGICAL type and specifies how the data is to be written to the video. Action may be one of the following:

1 = OR
2 = AND
3 = PRESET
4 = PSET
5 = XOR

PUT takes a rectangular pixel area that has been stored by GET and puts it on the screen at current x and y coordinates set by calling SETXY.

Example

```
CALL PUT (V,1)
```

Sample Program (see GET)

SCREEN
Selects Screen

SCREEN (switch)

switch is of LOGICAL type and specifies the type of screen display and may be one of the following:
0 = Graphics Screen
1 = Text Screen

SCREEN lets you select the proper screen.

Example

```
CALL SCREEN(0)
```

Sample Program

This example turns off the graphics display, draws a circle, then turns on the graphics display. The circle is then visible.

```
00010  C     EXAMPLE FOR SCREEN
00020        LOGICAL CMD
00040        CMD=1
00050        CALL GRPINI(0)
00060        CALL CLS
00070        CALL SCREEN(CMD)
00080        CALL SETXY(300,120)
00090        CALL CIRCLE(100,1,0.0,0.0,0.0)
00100        CALL PAINT(1,1)
00110        DO 20 I=1,10000
00120  20    CONTINUE
00130        CMD=0
00140        CALL SCREEN(CMD)
00150        END
```

SETXY
Sets Coordinates

SETXY(x,y)

(x,y) are INTEGER type and represent coordinates on the Graphics Screen.

SETXY sets and holds both current and previous X- and Y-coordinates. When a new coordinate is given, it is designated as the "current coordinate" and the last coordinate is designated as the "previous coordinate." If a new coordinate is specified, the "previous coordinate" is lost and the "current coordinate" becomes the "previous coordinate."

Example

```
CALL SETXY(100,100)
```

Sample Program (see LINE)

SETXYR
Sets Relative Coordinates

SETXYR(p1,p2)

(p1,p2) are INTEGER type and represent Relative Coordinates on the Graphics Screen.

SETXYR sets the current (x,y) coordinates relative to the previously set (x,y) coordinates. For example, if the "current" coordinates are (100,100), CALL SETXYR(10,10) will set the "current" coordinates to (110,110); the "previous" coordinates will then be (100,100).

Example

```
CALL SETXYR(30,30)
```

Sample Program

```
00010  C    DRAW TWO INTERSECTING CIRCLES
00020       CALL GRPINI(1)
00030       CALL CLS
00040       CALL SETXY(100,100)
00050       CALL CIRCLE(50,1,0.0,0.0,0.0)
00060  C    DRAW SECOND CIRCLE WITH CENTER 20
00070  C    PIXELS TO THE RIGHT OF FIRST CIRCLE
00080       CALL SETXYR(20,0)
00090       CALL CIRCLE(50,1,0.0,0.0,0.0)
00100       END
```

VIEW
Sets Viewport

VIEW(leftX,leftY,rightX,rightY,color,border)

leftX, leftY, rightX, rightY are INTEGER type and specify the viewport's parameters.
leftX and rightX are numeric expressions from 0 to 639 and specify viewport's corner X-coordinates. leftY and rightY are numeric expressions from 0 to 239 and specify the viewport's corner Y-coordinates.
color is of LOGICAL type, specifies the OFF/ON color code and is a numeric expression of either 0 (OFF, black), 1 (ON, white), or -1 (viewport is not shaded).
border is of LOGICAL type, specifies the border color for the viewport and is an integer expression of either 0 (OFF, black), 1 (ON, white), or -1 (border is not drawn).

VIEW draws viewports on your screen. Graphics is displayed only in the last defined viewport.

The upper-left corner of viewport is read as (0,0) (the "relative origin") when creating items inside the viewport. All the other coordinates are read relative to this origin. However, the "absolute coordinates" of the viewport, as they are actually defined on the Graphics Cartesian system, are retained in memory and can be read using VIEW as a function.

Example

```
CALL VIEW(100,100,200,200,0,1)
```

Sample Program

```
00100 C     SAMPLE VIEW PROGRAM
00200       LOGICAL COLOR,BORDER,K
00300       INTEGER FVIEW
00400       CALL GRPINI(1)
00500       CALL CLS
00500 C     SET UP VIEW PORT
00700       COLOR=0
00800       BORDER=1
00900       CALL VIEW(210,80,420,160,COLOR,BORDER)
01000 C     DRAW MULTIPLE CIRCLES
01100       CALL SETXY(105,40)
01200       DO 20 I=10,150,10
01300       CALL CIRCLE(I,1,0.0,0.0,0)
01400 20    CONTINUE
01500 C     DISPLAY VIEWPORT COORDINATES
01600       DO 40 I=1,4
01700       K=I-1
01800       J=FVIEW(K)
01900       WRITE (3,35)I,J
02000 35    FORMAT ('2','VIEW PORT COORDINATE ',I4,' IS AT',I4)
02100 40    CONTINUE
02200 C     PRINT EMPTY LINES
02300       DO 60 I=1,6
02400       WRITE (3,50)
02500 50    FORMAT (1H1)
02600 60    CONTINUE
02700       END
```

The following two descriptions are functions in the Graphics Subroutine Library and must be declared as LOGICAL and INTEGER, respectively, in any routine that uses them.

Functions

POINT
Reads Pixel Value at Current Coordinates

V=POINT(X)

X is a dummy variable needed to set up the proper FORTRAN linkage to the POINT routine.

POINT returns the OFF/ON pixel value at current x and y coordinate as specified by SETXY or SETXYR. If the point is not in the current viewport, POINT returns -1.

Example

```
K=POINT(M)
```

Sample Program (see PSET)

FVIEW
Reads Viewport's Parameters

FVIEW (n)

n is of LOGICAL type and is an integer expression from 0 to 3.

FVIEW returns the specified viewport parameter:
- 0 = returns the left X-coordinate
- 1 = returns the left Y-coordinate
- 2 = returns the right X-coordinate
- 3 = returns the right Y-coordinate

Example

```
I=FVIEW(0)
```

Sample Program (see VIEW)

5/ Programming the Graphics Board

The Graphics Board provides 640 X 240 byte addressable pixels on a TRS-80 Model III. The Graphics Board contains 32K of screen RAM to store video data consisting of four 64K RAMs which are double accessed for 8 bytes of data. Regular alphanumeric data is stored in the static RAM on the Video Board. The Graphics Board uses separate hardware to generate a 640 X 240 display, so only one screen may be displayed at a time. If the video is switched from Text to Graphics Screen very rapidly, the Video display may lose horizontal/vertical synchronization.

I/O port mapping is used to read and write data to the board. The Board is addressable at 80-83 Hex.

There are four internal registers which can be written to or read on the board. They are as follows:

1. **X-Position** - X-address (0 to 127) for data write only. (0 to 79 for display.)
2. **Y-Position** - Y-address (0 to 255) for data write only. (0 to 238 for display.)
3. **Data** - Graphics data in "byte" form. Each byte turns on or off 8 consecutive horizontal dots.
4. **Options** - 8 flags which turn on or off the user programmable options (Write only).

The I/O port mapping of the board is:

- x0 - X-Register Write. (80)
- x1 - Y-Register Write. (81)
- x2 - Video data read or write. (82)
- x3 - Options write. (83)

where x denotes the upper nibble of the I/O boundary as set by the DIP Switches. They are set by the factory at 80H.

The Graphics Board uses X-Y addressing to locate the start of a Graphics data byte. The upper-left of the screen is (0,0) while the lower-right is (079,239). If the bit is a 1, the dot will be ON. For example, if you wanted to turn on the 5th dot on the top row, the registers would contain: X POSITION=0, Y POSITION=0, DATA=(00001000)=08H. Note that in calculating points to plot, the Y-position is correct for a single dot. Only the X-position must be corrected to compensate for the byte addressing. This can be accomplished in a simple subroutine.

Line Drawing Options

There are two 8-bit counters which act as latches for the X- and Y-address. You may select, through the options register, if they are to automatically count after a read or write to graphic memory. Also, the counters may increment or decrement independently. These counters do not count to their respective endpoints and reset. Instead, they will overflow past displayable video addresses. Therefore, the software should not allow the counters to go past 79 and 239. However, these extra memory locations may be used for data storage.

Examples

The following are brief examples on how to use the Graphics Board.

Read the video byte at X=0, Y=0

```
        XOR     A                   ;CLEAR A
        OUT     (80H),A             ;OUTPUT X ADDRESS
        OUT     (81H),A             ;OUTPUT Y ADDRESS
        IN      A,(82H)             ;READ VIDEO BYTE
```

Draw a line from X=0,Y=0 to X=639, Y=0 using the hardware line drawing

```
        LD      B,79                ;B HAS CHARACTER COUNT
        LD      A,0B1H              ;OPTIONS:INCREMENT X AFTER WRITE
                                    ;10110001 Binary
        OUT     (83H),A
        XOR     A
        OUT     (80H),A             ;OUT X ADDRESS STARTING
        OUT     (81H),A             ;OUTPUT Y ADDRESS
        LD      A,0FFH              ;LOAD A WITH ALL DOTS ON
LOOP    OUT     (82H),A             ;OUTPUT DOTS
        DJNZ    LOOP                ;OUTPUT NUMBER IN B REGISTER
```

Options Programming

No.	Option	Description
0	GRAPHICS/ALPHA*	Turns graphics ON and OFF. "1" turns graphics ON.
1	NOT USED	
2	XREG DEC/INC*	Selects whether X decrements or increments. "1" selects decrement.
3	YREG DEC/INC*	Selects whether Y decrements or increments. "1" selects decrement.
4	X CLK RD*	If address clocking is desired, a "0" clocks the X address up or down AFTER a Read depending on the status of BIT 2.
5	Y CLK RD*	If address clocking is desired, a "0" clocks the Y address up or down AFTER a Read depending on the status of BIT 3.
6	X CLK WR*	A "0" clocks AFTER a Write.
7	Y CLK WR*	A "0" clocks AFTER a Write.

Table 9. Options Programming

Appendix A/ BASICG/Utilities Reference Summary

Argument ranges are indicated below by special letters and words:

ar is a single-precision floating point number > 0.0 (to 1* 10^{38}).

b	is an integer expression of either 0 or 1.
B	specifies a box.
BF	specifies a shaded box.
c	is an integer expression of 0 or 1.
n	is an integer expression from 0 to 2.
p	is an integer expression from 0 to 3.
r	is an integer expression from 0 to 639.
x	is an integer expression from 0 to 639.
y	is an integer expression from 0 to 239.
action	is either AND, PSET, PRESET, OR, or XOR.
background	is a string of either 0 or 1.
border	is an integer expression of either 0 or 1.
end	is an expression from -6.283185 to 6.283185.
start	is an expression from -6.283185 to 6.283185.
switch	is an integer expression of 0 or 1.
tiling	is a string or an integer expression of 0 or 1.
type	is an integer expression of 0 or 1.

CIRCLE(x,y)r,c,start,end,ar Draws a circle, ellipse, semicircle, arc, or point.

```
CIRCLE(100,100),25,1                CIRCLE(150,150),40,1,,,.6
CIRCLE(100,100),100,PI,2*PI,5       CIRCLE(-50,-50),200
```

CLS Clears the Text Screen and video memory.

```
CLS          SYSTEM"CLS"
```

CLR Clears the Graphics Screen.

```
CLR
```

GCLS Clears the Graphics Screen and memory.

```
GCLS      CMD "I","GCLS"     100 CMD "I","GCLS"
```

GET(x1,y1)-(x2,y2),array name Reads the contents of a rectangular pixel area into an array.

```
GET(10,10)-(50,50),V
```

GLOAD filename/ext.password:d Loads graphics memory.

```
GLOAD PROG    CMD "I","GLOAD PROG"
```

GLOCATE (x,y),direction Sets the Graphics Cursor

```
GLOCATE (320,120),0
```

GPRINT Dumps graphic display on the printer.

```
 GPRINT            CMD "I","GPRINT"    100 CMD "I","GPRINT"
```

GPRT2 Dumps graphic display on the printer without rotating 90 degrees.

```
GPRT2              CMD"I","GPRT2"      100 CMD"I","GPRT2"
```

GPRT3 Dumps graphic display on the printer without rotating 90 degrees.

```
GPRT3              CMD"I","GPRT3"      100 CMD"I","GPRT3"
```

GROFF Turns Graphic Display OFF.

```
GROFF   CMD "I","GROFF"
```

GRON Turns Graphic Display ON.

```
GRON  CMD "I","GRON"
```

GSAVE filename/ext.password:d Saves graphics memory.

```
GSAVE PROG       CMD "I","GSAVE PROG"
```

LINE(x1,y1)-(x2,y2),c,B or BF, style Draws a line/box.

```
LINE -(100,100)             LINE(100,100)-(200,200),1,B,45
LINE(0,0)-(100,100),1,BF       LINE(-200,-200)-(100,100)
```

PAINT(x,y),tiling,border,background Paints the screen.

```
PAINT(320,120),1,1            PAINT(320,120),"DDDDD",1
PAINT(320,120),A$,1
PAINT(320,120),CHR$(0)+CHR$(&HFF),0,CHR$(&H00)
PAINT(320,120),CHR$(E)+CHR$(77)+CHR$(3)
```

&POINT(x,y) A function. Tests graphics point.

```
PRINT &POINT(320,120)     IF &POINT(320,120)=1 THEN . . .
PRINT &POINT(320,120),-1
```

PRESET(x,y),switch Sets pixel OFF or ON.

```
PRESET(100,100),0
```

PRINT #-3, item list Write text characters to the Graphics Screen.

```
PRINT #-3,"MONTHLY"
```

PSET(x,y),switch Sets pixel ON or OFF.

```
PSET(100,100),1
```

PUT(x1,y1),array name,action Puts graphics from an array onto the screen.

```
PUT(100,100),A,PSET          PUT(100,100),A,AND
PUT(A,B),B
```

SCREEN type Selects the screen.

```
SCREEN 0
```

VIEW(x1,y1)-(x2,y2),c,b Redefines the screen and creates a viewport.

```
VIEW(100,100)-(150,150)      VIEW(100,100)-(150,150),0,1
```

&VIEW(p) A function. Returns viewport's coordinates.

```
PRINT &VIEW(1)
```

Appendix B/ BASICG Error Messages

Code	Abbreviation	Explanation
1	NF	NEXT without FOR. NEXT is used without a matching FOR statement. This error may also occur if NEXT variables are reversed in a nested loop.
2	SN	Syntax. This is usually the result of incorrect punctuation, an illegal character or a misspelled command.
3	RG	RETURN without GOSUB. A RETURN statement was executed with insufficient data available. The DATA statement may have been left out or all data may have been read.
4	OD	Out of data. A READ statement was executed with insufficient data available. The DATA statement may have been left out or all data may have been read.
5	FC	Illegal function call. An attempt was made to execute an operation using an illegal parameter. Graphic examples: PUTting a display that is partially off the Screen, GETting an array that is not properly dimensioned, or using more than two OFF tiles or two ON tiles in a string when tiling (with PAINT).
6	OV	Overflow. The magnitude of the number derived or input is too large for the data storage type assigned to it. The integer range is (-32768 to 32767) for BASICG.
7	OM	Out of memory. All available memory has been used or reserved. This may occur with large array dimensions and nested branches such as GOSUB and

		FOR/NEXT loops.
8	UL	Undefined line. An attempt was made to reference a non-existent line.
9	BS	Bad subscript. An attempt was made to assign an array element with a subscript beyond the dimensioned range.
10	DD	Double-dimensioned array. An attempt was made to dimension an array which had previously been created with DIM or by default statements. ERASE must be used first.
11	/0	Division by zero. An attempt was made to use a value of zero in the denominator. Note: If you can't find an obvious division by zero, check for division by numbers smaller than allowable ranges (see OV).
12	ID	Illegal direct. An attempt was made to use a program only statement like INPUT in an immediate (non-program) line.
13	TM	Type mismatch. An attempt was made to assign a number to a string variable or a string to a numeric variable.
14	OS	Out of string space. The amount of string space allocated was exceeded. Use CLEAR to allocate more string space. 100 bytes is the default string space allocation.
15	LS	Long string. A string variable was assigned a string which exceeded 255 characters in length.
16	ST	String too complex. A string operation was too complex to handle. The operation must be broken into shorter steps.
17	CN	Can't continue. A CONT command was given at a point where the command can't be carried out, e.g., directly after the program has been edited.

18	UF	Undefined user function. An attempt has been made to call a USR function without first defining its entry point via a DEFUSR statement.
19	NR	No RESUME. During an error-trapping routine, BASICG has reached the end of the program without encountering a RESUME.
20	RW	RESUME without error. A RESUME was encountered when no error was present. You need to insert END or GOTO in front of the error-handling routine.
21	UE	Undefined error. Reserved for future use.
22	MO	Missing operand. An operation was attempted without providing one of the required operands.
23	BO	Buffer overflow. An attempt was made to input a data line which has too many characters to be held in the line buffer.
24	NB	Files not compatible. An attempt was made to load a BASIC file (in compressed format) into BASICG.
25-49	UE	Undefined error. Reserved for future use.
50	FO	Field overflow. An attempt was made to have more characters than the direct-access file record length allows. The record length is assigned when the file is first opened. The default length is 256.
51	IE	Internal error. Also indicates an attempt to use EOF on a file which is not open.

52	BN	Bad file number. An attempt was made to use a file number which specifies a file that is not open or that is greater than the number of files specified when BASICG was started up.
53	FF	File not found. Reference was made in a LOAD, KILL or OPEN statement to a file which did not exist on the diskette specified.
54	BM	Bad file mode. Program attempted to perform direct access on a file opened for sequential access or vice-versa.
55	AO	File already open. An attempt was made to open a file that was already open. This error is also output if KILL, LOAD, SAVE, etc., is given for an open file.
56	IO	Disk I/O error. An error has been detected during a disk access.
57	UE	Undefined in Model III BASIC.
58	UE	Undefined error. Reserved for future use.
59	DF	Diskette full. All storage space on the diskette has been used. KILL unneeded files or use a formatted diskette which has available space.
60	EF	End of file. An attempt was made to read past the end of file.
61	RN	Bad record number. In a PUT or GET statement, the record number is either greater than the allowable maximum, equal to zero, or negative.
62	NM	Bad file name.
63	MM	Mode mismatch. A sequential OPEN was executed for a file that already existed on the diskette as a direct access file, or vice versa.

64	UE	Undefined error. Reserved for future use.
65	DS	Direct statement. A direct statement was encountered during a load of a program in ASCII format. The load is terminated.
66	FL	Too many files.

==

Appendix C/ Subroutine Language Reference Summary

CIRCLE (radius,color,start,end,ar) Draws a circle, ellipse, semicircle, arc, or point. (x,y) coordinates set by SETXY.

```
CALL CIRCLE(100,1,0,0,0)
```

CLS Clears the Graphics Screen.

```
CALL CLS
```

FVIEW (n) Returns viewport parameter.

```
I=FVIEW(0)
```

GET (array,size) Reads the contents of a rectangular pixel area into an array for future use by PUT.

```
CALL GET(A,4000)
```

GPRINT (size,array) Displays textual data.

```
CALL GPRINT (28,ARRAY1)
```

GRPINI(option) Graphics initialization routine.

```
CALL GRPINI(0)
```

LINE (color,style) Draws a line. Coordinates set by SETXY or SETXYR.

```
CALL LINE (1,-1)
```

LINEB (color,style) Draws a box. Coordinates set by SETXY or SETXYR.

```
CALL LINEB (1,-1)
```

LINEBF (color) Draws a filled box. Coordinates set by SETXY or SETXYR.

```
CALL LINEBF (1)
```

LOCATE (n) Sets the direction for displaying textual data.

```
CALL LOCATE (0)
```

PAINT (color,border) Paints the screen.

```
CALL PAINT(1,1)
```

PAINTT (arrayT,border,arrayS) Paints the screen with defined paint style.

CALL PAINTT (A,1,V)

POINT Returns the pixel value at current coordinates.
K=POINT(M)

PRESET (color) Sets the pixel ON or OFF.
CALL PRESET(0)

PSET (color) Sets the pixel ON or OFF.
CALL PSET(0)

SCREEN (type) Sets the screen.
CALL SCREEN(1)

SETXY(X,Y) Sets the coordinates (absolute).
CALL SETXY(100,100)

SETXYR(X,Y) Sets the coordinates (relative).
CALL SETXYR(50,50)

VIEW(leftX,leftY,rightX,rightY,color,border)
Sets the viewport.
CALL VIEW(100,100,200,200,0,1)

Appendix D/ Sample Programs

BASICG

```
10 '
20 ' Pie Graph Program ("PECANPIE/GRA")
30 '
40 ' Object
50 '   The object of this program is to draw a pie graph of the
60 '   expenses for a given month of eight departments of a company,
70 '   along with the numerical value of each pie section
80 '   representation.
90 '
100 '
110 ' Running the program
120 '  The month and the amounts spent by each department are input,
130 '  and the program takes over from there.
140 '
150 ' Special features
160 '    The amounts spent by each account as well as the total
170 '    amount spent are stored in strings.  The program will
180 '    standardize each string so that it is 9 characters long
190 '    and includes two characters to the right of the decimal
200 '    point.  This allows for input of variable length and an
210 '    optional decimal point.
220 '
230 '    The various coordinates used in the program are found
240 '    based on the following equations:
250 '
260 '     x = r * cos(theta)
270 '     y = r * sin(theta)
280 '
290 '    where x and y are the coordinates, r is the radius, and theta
300 '    is the angle.  (Note:  The y-coordinates are always multiplied
310 '    by 0.5.  This is because the y pixels are twice the size of the
320 '    x pixels.)
330 '
340 '    If an angle theta is generated by a percent less than 1%, the
350 '    section is not graphed, and the next theta is calculated.
360 '    However, the number will still be listed under the key.
370 '
380 ' Variables
390 '    ACCT$(i)   Description of the account
400 '    BUD$(i)    Amount spent by the account
```

```
410 '     DS$              Dollar sign (used in output)
420 '     HXCOL            Column number for the pie section number
430 '     HYRW             Row number for the pie section number
440 '     I                Counter
450 '     MN$              Month
460 '     PER(i)           Percent value of BUD$(i)
470 '     R                Radius of circle
480 '     T0               Angle value line to be drawn
490 '     T1               Angle value of the next line
500 '     TBUD$            Total of all the BUD$(i)'s
510 '     THALF            Angle halfway between T1 and T0 (used for
520 '                      location position for section number)
530 '     TILE$(i)         Paint style for each section
540 '     TWOPI            Two times the value of pi
550 '     X0               X-coordinate for drawing the line represented
560 '                      by T0
570 '     XP               X-coordinate for painting a section
580 '     Y0               Y-coordinate for drawing the line represented
590 '                      by T0
600 '     YP               Y-coordinate for painting a section
610 '
620 ' Set initial values
630 '
640 CLEAR 1000
650 DIM THALF(15),BUD$(15),ACCT$(15),PER(16)
660 TWOPI=2*3.14159
670 R=180
680 DS$="$"
690 ACCT$(1) = "Sales"
700 ACCT$(2) = "Purchasing"
710 ACCT$(3) = "R&D       "
720 ACCT$(4) = "Accounting"
740 ACCT$(5) = "Advertising "
750 ACCT$(6) = "Utilities  "
760 ACCT$(7) = "Security "
770 ACCT$(8) = "Expansion"
780 TILE$(0)=CHR$(&H22)+CHR$(&H00)
790 TILE$(1)=CHR$(&HFF)+CHR$(&H00)
800 TILE$(2)=CHR$(&H99)+CHR$(&H66)
810 TILE$(3)=CHR$(&H99)
820 TILE$(4)=CHR$(&HFF)
830 TILE$(5)=CHR$(&HF0)+CHR$(&HF0)+CHR$(&H0F)+CHR$(&H0F)
840 TILE$(6)=CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)
850 TILE$(7)=CHR$(&H03)+CHR$(&H0C)+CHR$(&H30)+CHR$(&HC0)
860 '
870 ' Enter values to be graphed, standardize them, and calculate
880 ' the percent they represent
890 '
```

```
900 CLR
910 CLS
920 SCREEN1
930 PRINT @64,"Enter month "
940 PRINT @192,"Enter amount spent by"
950 PRINT @256,"$"
960 PRINT @0,""
970 LINE INPUT "Enter month ";MN$
980 FOR I=1 TO 8
990 PRINT @214,ACCT$(I);"
1000 PRINT @256,"$"
1010 PRINT @192,""
1020 LINE INPUT "$";BUD$(I)
1030 IF INSTR(BUD$(I),".") = 0 THEN BUD$(I)=BUD$(I)+".00"
1040 IF LEN(BUD$(I))<9 THEN BUD$(I)=" "+BUD$(I):GOTO 1040
1050 TBUD$=STR$(VAL(TBUD$)+VAL(BUD$(I)))
1060 NEXT I
1070 IF INSTR(TBUD$,".")=0 THEN TBUD$=TBUD$+".00"
1080 IF LEN(TBUD$)<9 THEN TBUD$=" "+TBUD$:GOTO 1080
1090 FOR I=1 TO 8
1100 PER(I)=VAL(BUD$(I))/VAL(TBUD$)*100
1110 NEXT I
1120 SCREEN0
1130 '
1140 ' Draw the circle and calculate the location of the lines and
1150 ' the line numbers
1160 '
1170 CIRCLE(410,120),R
1180 FOR I=0 TO 8
1190 T0=TWOPI/100*PER(I)+T0
1200 X0=410+R*COS(T0)
1210 Y0=120-R*SIN(T0)*0.5
1220 T1=TWOPI/100*PER(I+1)+T0
1230 THALF(I)=(T0+T1)/2
1240 HXCOL=(410+R*1.15*COS(THALF(I)))
1250 HYRW=(120-R*1.15*SIN(THALF(I))*0.5)
1260 IF PER(I)>1 THEN LINE (410,120)-(X0,Y0)
1270 GLOCATE (HXCOL,HYRW),0
1280 IF I<8 AND PER(I+1)>1 THEN PRINT #-3,I+1
1290 NEXT I
1300 '
1310 ' Paint the appropriate sections of the pie
1320 '
1330 FOR I=0 TO 7
1340 XP=410+R*0.5*COS(THALF(I))
1350 YP=120-R*0.5*SIN(THALF(I))*0.5
1360 IF PER(I+1)<=1 THEN 1380
1370 PAINT (XP,YP),TILE$(I),1
1380 NEXT I
```

```
1390 '
1400 ' Print the key for the graph
1410 '
1420 GLOCATE(0,10),0
1430 PRINT #-3,"Expenditures for"
1440 GLOCATE(0,25),0
1450 PRINT #-3,MN$
1460 GLOCATE(0,40),0
1470 PRINT #-3,"#    Description   Amount"
1480 FOR I=1 TO 8
1490 GLOCATE(0,(4+I)*15),0
1500 PRINT #-3,I
1510 GLOCATE(40,(4+I)*15),0
1520 PRINT #-3,ACCT$(I)
1530 GLOCATE(130,(I+4)*15),0
1540 PRINT #-3,DS$;BUD$(I)
1550 DS$=" "
1560 NEXT I
1570 GLOCATE(0,195),0
1580 PRINT #-3,STRING$(26,"-")
1590 GLOCATE(40,210),0
1600 PRINT #-3,"Total      ";TBUD$
1610 FOR I=1 TO 10000
1620 NEXT I
1630 SCREEN1
1640 END
```

```
10 ' "THREEDEE/GRA"
20 '
30 ' Object
40 '    The object of this program is to produce a three
50 ' dimensional bar graph representation of the gross
60 ' income for a company over a one year period.
70 '
80 ' Variables
90 '   A  Vertical alphanumeric character
100 '  BMSG$ Bottom message
110 '  CHAR$ Disk file input field
120 '  GI$  Gross income
130 '  I  Counter
140 '  J  Counter
150 '  MN$  Month
160 '  REC  Record number of vertical character
170 '  S1$  Single character of vertical message
180 '  TILE$ Tile pattern for painting
190 '  TTINC Total income for the year
200 '  X  X-coordinate of bar
210 '  Y(i) Y-coordinate of bar
220 '
230 'Input/output
240 '  The program prompts you to enter the gross income, in millions.
250 'for each month.  The program requires these values to be between one
260 'and nine.
270 '
280 'Set initial values
290 '
300 CLS
310 DIM Y(12),A(8),MN$(12)
320 DEFINT A
330 VMSG$=" Millions of dollars "
340 TMSG$="G r o s s   I n c o m e   F o r   1 9 8 0 "
350 BMSG$="M o n t h"
360 MN$(1)="January"
370 MN$(2)="February"
380 MN$(3)="March"
390 MN$(4)="April"
400 MN$(5)="May"
410 MN$(6)="June"
420 MN$(7)="July"
430 MN$(8)="August"
440 MN$(9)="September"
450 MN$(10)="October"
460 MN$(11)="November"
470 MN$(12)="December"
480 TILE$=CHR$(&H99)+CHR$(&H66)
490 X=-10
```

```
500 '
510 'Input gross income, and calculate the Y-coordinate
520 '
530 FOR I=1 TO 12
540 CLS
550 PRINT "Enter gross income in millions (1-9) for ";MN$(I)
560 LINE INPUT "$";GI$
570 Y(I)=205-20*VAL(GI$)
580 TTINC=TTINC+VAL(GI$)
590 NEXT I
600 CLR
610 SCREEN0
620 '
630 'Draw the graph and bars
640 '
650 FOR I=1 TO 12
660 CLS
670 X=X+50
680 LINE (X,Y(I))-(X+20,205),1,BF
690 LINE -(X+40,195)
700 LINE -(X+40,Y(I)-10)
710 LINE -(X+20,Y(I)-10)
720 LINE -(X,Y(I))
730 LINE (X+20,Y(I))-(X+40,Y(I)-10)
740 PAINT(X+21,Y(I)+2),TILE$,1
750 NEXT I
760 GLOCATE(40,215),0
770 PRINT #-3,"Jan   Feb   Mar    Apr   May   June  July   Aug   Sept  Oct
Nov   Dec"
780 GLOCATE(290,230),0
790 PRINT #-3,BMSG$
800 FOR I=1 TO 10
810 IF I>9 THEN C=1 ELSE C=2
820 GLOCATE((C*10)-5,(20-I*2)*10),0
830 PRINT #-3,STR$(I);"-"
840 NEXT I
850 LINE (35,0)-(35,205)
860 LINE -(639,205)
870 GLOCATE(0,180),3
880 PRINT #-3,VMSG$
890 GLOCATE(220,0),0
900 PRINT #-3,TMSG$
910 GLOCATE(260,10),0
920 PRINT #-3,"(Total income is";TTINC;" million)"
930 FOR I=1 TO 10000
940 NEXT I
950 SCREEN1
960 END
```

Printing Graphics Displays

There are many ways to use the stand-alone utilities (described in Graphic Utilities). The following discussion demonstrates one way to use the utilities with graphic displays generated under BASICG.

To print graphics, follow these steps:

1. When TRSDOS Ready appears, set FORMS to FORMS (WIDTH=255, LINES=60). (See your **Model III Disk System Owner's Manual.**)
2. Set the printer into Graphic Mode, if possible, and set the printer's other parameters (elongation, non-elongated, etc.), if applicable, according to instructions in your printer owner's manual.
3. Write, run and save your program as a BASICG program file.
4. Save the graphics memory to diskette using GSAVE.
5. Load the file into memory using GLOAD.
6. Enter the print command GPRINT.

Example

1. Set FORMS with your printer's printing parameters.

2. Load BASICG and type in this program:

```
  5 SCREEN 0
 10 DEFDBL Y
 20 CLR
 30 LINE (0,120)-(640,120)
 40 LINE (320,0)-(320,240)
 50 FOR X=0 TO 640
 60 PI=3.141259
 70 X1=X/640*2*PI-PI
 80 Y=SIN(X1)*100
 90 IF Y>100 THEN X=X+7
100 PSET (X,-Y+120)
110 NEXT X
120 GLOCATE(0,0),0
130 PRINT #-3,"THIS IS A SINE WAVE."
```

3. RUN the program.

   The program draws a sine wave on the Graphics Screen (graphics memory) and prints the statement in line 130 ("THIS IS A SINE WAVE.") on the Graphics Screen.

4. SINE (for sine wave) is the name we are giving this TRSDOS file. To save the contents of the graphics memory (which now includes the converted video memory) to diskette, type: `CMD"I","GSAVE SINE" <ENTER>`.

5. The graphics memory is saved as a TRSDOS file on your diskette and you will return to TRSDOS Ready.

6. Type: `GCLS <ENTER>`

   The graphics memory is now cleared.

7. To load the file back into memory, type:

   ```
   GLOAD SINE <ENTER>
   ```

   The display is now on the Graphics Screen.

8. To print, type: `GPRINT <ENTER>`.

FORTRAN Sample Programs

```
00100   C       HIGH RESOLUTION GRAPHICS TEST - MAIN PROGRAM
00200   C
00300           CALL GRPINI(0)
00400           CALL SCREEN(0)
00500   C
00600   C       CIRCLE TEST
00700   C
00800           CALL CTEST
00900   C
01000   C       LINE TEST
01100   C
01200           CALL LTEST
01300   C
01400   C       LINEB TEST
01500   C
01600           CALL LBTST
01700   C
01800   C       LINEBF TEST
01900   C
02000           CALL LBFTST
02100   C
02200   C       PAINTT TEST
02300   C
02400           CALL PTTTST
02500   C
02600   C       GET AND PUT TEST
02700   C
02800           CALL GPTST
02900   C
03000   C       PSET/POINT TEST
03100   C
03200           CALL PPTST
03300   C
03400   C       PRESET/POINT TEST
03500   C
03600           CALL PRETST
03700   C
03800   C       SCREEN TEST
03900   C
04000           CALL SCRTST
04100   C
04200   C       VIEW/FVIEW TEST
04300   C
04400           CALL VTEST
```

```
04500          CALL CLS(2)
04600          END
```

```
                SUBROUTINE CTEST
      C
      C         THIS SUBROUTINE TESTS CIRCLE, SETXY, AND PAINT
      C
                LOGICAL MSG(29)
                CALL CLS
                ENCODE(MSG,100)
      100       FORMAT('TEST CIRCLE, SETXY, AND PAINT')
                CALL SETXY(0,0)
                CALL LOCATE(0)
                CALL GPRINT(29,MSG)
                CALL WAIT
                CALL VIEW(0,30,639,239,0,0)
                DO 10 I=1,100
                IX=MOD(I*17,640)
                IY=MOD(I*13,210)
                IR=I*1.5
                START=MOD(I,13)-6.0
                END=MOD(I*3,13)-6.0
                IF (START.LT.END) GOTO 1
                T=START
                START=END
                END=T
      1         CONTINUE
                RATIO=MOD(I*3,100)
                IF (RATIO.GT.0) RATIO=RATIO/40.
                CALL SETXY(IX,IY)
                CALL CIRCLE(IR,1,START,END,RATIO)
      10        CONTINUE
      C
      C         RANDOMLY FILL IN THE AREAS
      C
                DO 11 I=1,50
                IX=MOD(I*23,640)
                IY=MOD(I*11,210)
                CALL SETXY(IX,IY)
                CALL PAINT(1,1)
      11        CONTINUE
                CALL WAIT
                CALL VIEW(0,0,639,239,-1,-1)
                RETURN
                END
```

```
00100           SUBROUTINE LTEST
00200   C
00300   C       THIS ROUTINE EXERCISES LINE
00400   C
00500           LOGICAL MSG(19)
00600           CALL CLS(0)
00700           ENCODE(MSG,100)
00800   100     FORMAT('LINE AND PAINT TEST')
00900           CALL SETXY(0,0)
01000           CALL LOCATE(0)
01100           CALL GPRINT(19,MSG)
01200           CALL WAIT
01300           J=100
01400           DO 10 I=1,639,2
01500           CALL SETXY(I,15)
01600           CALL SETXY(I,239)
01700           CALL LINE(1,J)
01800           J=J-1
01900   10      CONTINUE
02000           CALL WAIT
02100           CALL VIEW(0,15,639,239,0,0)
02200           CALL CLS
02300   C
02400   C       DRAW WHITE LINES AND FILL IN RANDOMLY
02500   C
02600           IX=MOD(I*19,639)
02700           IY=MOD(I*17,223)
02800           CALL SETXY(IX,IY)
02900           DO 11 I=1,100
03000           IX=MOD(I*23,639)
03100           IY=MOD(I*29,223)
03200           CALL SETXY(IX,IY)
03300           CALL LINE(1,-1)
03400   11      CONTINUE
03500           DO 12 I=1,50
03600           IX=MOD(I*31,639)
03700           IY=MOD(I*37,223)
03800           CALL SETXY(IX,IY)
03900           CALL PAINT(1,1)
04000   12      CONTINUE
04100           CALL WAIT
04200   C
04300   C   WHITE OUT SCREEN, DRAW BLACK LINES, PAINT BLACK RANDOMLY
04400   C
04500           CALL VIEW(0,15,639,239,1,1)
04600           DO 15 I=1,100
04700           IX=MOD(I*11,639)
04800           IY=MOD(I*13,223)
04900           CALL SETXY(IX,IY)
```

```
05000          CALL LINE(0,-1)
05100   15     CONTINUE
05200          DO 16 I=1,50
05300          IX=MOD(I*17,639)
05400          IY=MOD(I*19,223)
05500          CALL SETXY(IX,IY)
05600          CALL PAINT(0,0)
05700   16     CONTINUE
05800          CALL WAIT
05900          CALL VIEW(0,0,639,239,0,0)
06000          RETURN
06100          END
```

```
        SUBROUTINE LBFTST
C
C       LINEBF TEST
C
        LOGICAL MSG(11)
        CALL CLS
        ENCODE(MSG,100)
100     FORMAT('LINEBF TEST')
        CALL SETXY(0,0)
        CALL LOCATE(0)
        CALL GPRINT(11,MSG)
        CALL WAIT
        IXP=639
        ICLR=1
        DO 10 IX=0,120
        CALL SETXY(IX*2,IX+30)
        CALL SETXY(IXP,IXP-400)
        CALL LINEBF(ICLR)
        IXP=IXP-3
        ICLR=ICLR-1
        IF (ICLR.LT.0) ICLR=1
10      CONTINUE
        CALL WAIT
        RETURN
        END
```

```
00100          SUBROUTINE PTTTST
00200    C
00300    C     PAINT WITH TILES TEST
00400    C
00500          LOGICAL A(65),B(4),IS(16),MSG(23)
00600          DATA A(1)/8/
00700    C     X
00800          DATA A(2),A(3),A(4),A(5)/X'41',X'22',X'14',X'08'/
00900          DATA A(6),A(7),A(8),A(9)/X'14',X'22',X'41',X'00'/
01000    C     FINE HORIZONTAL LINES
01100          DATA A(10),A(11),A(12)/2,X'FF',X'00'/
01200    C     MEDIUM HORIZONTAL LINES
01300          DATA A(13)/4/
01400          DATA A(14),A(15),A(16),A(17)/X'FF',X'FF',X'00',X'00'/
01500    C     DIAGONAL LINES
01600          DATA A(18)/4/
01700          DATA A(19),A(20),A(21),A(22)/X'03',X'0C',X'30',X'C0'/
01800    C     LEFT TO RIGHT DIAGONALS
01900          DATA A(23)/4/
02000          DATA A(24),A(25),A(26),A(27)/X'C0',X'30',X'0C',X'03'/
02100    C     FINE VERTICAL LINES
02200          DATA A(28),A(29)/1,X'AA'/
02300    C     MEDIUM VERTICAL LINES
02400          DATA A(30),A(31)/1,X'CC'/
02500    C     COARSE VERTICAL LINES
02600          DATA A(32),A(33)/1,X'F0'/
02700    C     ONE PIXEL DOTS
02800          DATA A(34),A(35),A(36)/2,X'22',X'00'/
02900    C     TWO PIXEL DOTS
03000          DATA A(37),A(38),A(39)/2,X'99',X'66'/
03100    C     PLUSES
03200          DATA A(40),A(41),A(42),A(43)/3,X'3C',X'3C',X'FF'/
03300    C     SOLID
03400          DATA A(44),A(45)/1,X'FF'/
03500    C     BROAD CROSS HATCH
03600          DATA A(46),A(47),A(48),A(49)/3,X'92',X'92',X'FF'/
03700    C     THICK CROSS HATCH
03800          DATA A(50)/4/
03900          DATA A(51),A(52),A(53),A(54)/X'FF',X'FF',X'DB',X'DB'/
04000    C     FINE CROSS HATCH
04100          DATA A(54),A(55),A(56)/2,X'92',X'FF'/
04200    C     ALTERNATING PIXELS
04300          DATA A(57),A(58),A(59)/2,X'55',X'AA'/
04400          DATA B(1),B(2),B(3),B(4)/1,0,1,X'FF'/
04500          DATA IS(1),IS(2),IS(3),IS(4),IS(5),IS(6)/1,10,13,18,23,28/
04600          DATA IS(7),IS(8),IS(9),IS(10),IS(11)/30,32,34,37,40/
04700          DATA IS(12),IS(13),IS(14),IS(15),IS(16)/44,46,50,54,57/
04800          CALL CLS
04900          ENCODE(MSG,100)
```

```
05000   100     FORMAT('PAINTT AND SETXYR TESTS')
05100           CALL SETXY(0,0)
05200           CALL LOCATE(0)
05300           CALL GPRINT(23,MSG)
05400           CALL WAIT
05500   C
05600   C       PAINT ON A BLACK BACKGROUND
05700   C
05800           DO 10 I=1,16
05900           CALL SETXY(0,40)
06000           CALL SETXYR(639,199)
06100           CALL LINEB(1,-1)
06200           CALL SETXYR(-300,-100)
06300           ITMP=IS(I)
06400           CALL PAINTT(A(ITMP),1,B)
06500           CALL WAIT
06600           CALL VIEW(0,40,639,239,0,0)
06700           CALL VIEW(0,0,639,239,-1,-1)
06800   10      CONTINUE
06900   C
07000   C       PAINT ON A WHITE BACKGROUND
07100   C
07200           DO 11 I=1,16
07300           IF(I.EQ.12) GOTO 11
07400           CALL VIEW(0,40,639,239,0,0)
07500           CALL VIEW(0,0,639,239,-1,-1)
07600           CALL SETXY(0,40)
07700           CALL SETXYR(639,199)
07800           CALL LINEBF(1)
07900           CALL SETXYR(-300,-100)
08000           ITMP=IS(I)
08100           CALL PAINTT(A(ITMP),0,B(3))
08200           CALL WAIT
08300   11      CONTINUE
08400           RETURN
08500           END
```

```
00100              SUBROUTINE GPTST
00200    C
00300    C         GET AND PUT TEST
00400    C
00500              LOGICAL A(1000),MSG(16)
00600              CALL CLS
00700              ENCODE(MSG,100)
00800    100       FORMAT('GET AND PUT TEST')
00900              CALL SETXY(0,0)
01000              CALL LOCATE(0)
01100              CALL GPRINT(16,MSG)
01200              CALL VIEW(0,30,639,239,0,0)
01300              CALL SETXY(100,100)
01400              CALL SETXYR(30,30)
01500              CALL LINEBF(1)
01600              CALL GET(A,1000)
01700              CALL CLS
01800              CALL WAIT
01900              CALL SETXY(100,100)
02000              CALL PUT(A,1)
02100              CALL WAIT
02200              CALL VIEW(0,0,639,239,0,-1)
02300              RETURN
02400              END
```

```
00100              SUBROUTINE PPTST
00200     C
00300     C        PSET AND POINT TEST
00400     C
00500              LOGICAL POINT,MSG(21)
00600              CALL CLS
00700              ENCODE(MSG,100)
00800     100      FORMAT('PSET AND POINT TEST')
00900              CALL SETXY(0,0)
01000              CALL LOCATE(0)
01100              CALL GPRINT(19,MSG)
01200              CALL WAIT
01300              CALL CLS
01400     C
01500     C        SET AND CHECK ALL PIXELS
01600     C
01700              DO 10 I=0,639
01800              DO 11 J=0,239
01900              CALL SETXY(I,J)
02000              CALL PSET(1)
02100              K=POINT(L)
02200              IF(K.EQ.0) GOTO 999
02300     11       CONTINUE
02400     10       CONTINUE
02500     C
02600     C        RESET AND CHECK ALL PIXELS
02700     C
02800              DO 12 I=0,639
02900              DO 13 J=0,239
03000              CALL SETXY(I,J)
03100              CALL PSET(0)
03200              K=POINT(L)
03300              IF (K.EQ.1) GOTO 999
03400     13       CONTINUE
03500     12       CONTINUE
03600              CALL CLS
03700              ENCODE(MSG,101)
03800     101      FORMAT('PSET AND POINT PASSED')
03900              CALL SETXY(0,0)
04000              CALL LOCATE(0)
04100              CALL GPRINT(21,MSG)
04200              GOTO 1000
04300     999      CALL CLS
04400              ENCODE(MSG,102)
04500     102      FORMAT('PSET AND POINT FAILED')
04600              CALL SETXY(0,0)
04700              CALL LOCATE(0)
04800              CALL GPRINT(21,MSG)
04900     1000     CALL WAIT
```

```
05000            RETURN
05100            END
```

```
00100           SUBROUTINE PRETST
00200     C
00300     C     PRESET AND POINT TEST
00400     C
00500           LOGICAL POINT,MSG(23)
00600           CALL CLS
00700           ENCODE(MSG,100)
00800     100   FORMAT('PRESET AND POINT TEST')
00900           CALL SETXY(0,0)
01000           CALL LOCATE(0)
01100           CALL GPRINT(23,MSG)
01200           CALL WAIT
01300           CALL CLS
01400     C
01500     C     SET AND CHECK ALL PIXELS
01600     C
01700           DO 10 I=0,639
01800           DO 11 J=0,239
01900           CALL SETXY(I,J)
02000           CALL PRESET(1)
02100           K=POINT(L)
02200           IF(K.EQ.0) GOTO 999
02300     11    CONTINUE
02400     10    CONTINUE
02500     C
02600     C     RESET AND CHECK ALL PIXELS
02700     C
02800           DO 12 I=0,639
02900           DO 13 J=0,239
03000           CALL SETXY(I,J)
03100           CALL PRESET(0)
03200           K=POINT(L)
03300           IF (K.EQ.1) GOTO 999
03400     13    CONTINUE
03500     12    CONTINUE
03600           CALL CLS
03700           ENCODE(MSG,101)
03800     101   FORMAT('PRESET AND POINT PASSED')
03900           CALL SETXY(0,0)
04000           CALL LOCATE(0)
04100           CALL GPRINT(23,MSG)
04200           GOTO 1000
04300     999   CALL CLS
04400           ENCODE(MSG,102)
04500     102   FORMAT('PRESET AND POINT FAILED')
04600           CALL SETXY(0,0)
04700           CALL LOCATE(0)
04800           CALL GPRINT(23,MSG)
04900     1000  CALL WAIT
```

```
05000          RETURN
05100          END
```

```
00100               SUBROUTINE SCRTST
00200     C
00300     C         SCREEN TEST
00400     C
00500               LOGICAL MSG(11)
00600               CALL CLS
00700               ENCODE(MSG,100)
00800     100       FORMAT('SCREEN TEST')
00900               CALL SETXY(0,0)
01000               CALL LOCATE(0)
01100               CALL GPRINT(11,MSG)
01200               CALL WAIT
01300               CALL SETXY(300,120)
01400               CALL CIRCLE(100,1,0.0,6.28,0.5)
01500               CALL CIRCLE(100,1,0.0,6.28,0.25)
01600               CALL CIRCLE(50,1,0.0,6.28,0.5)
01700               CALL PAINT(1,1)
01800     C
01900     C         GRAPHICS SCREEN
02000     C
02100               CALL SCREEN(0)
02200               CALL WAIT
02300               CALL WAIT
02400               CALL WAIT
02500     C
02600     C         TEXT SCREEN
02700     C
02800               CALL SCREEN(1)
02900               CALL WAIT
03000               CALL WAIT
03100               CALL WAIT
03200     C
03300     C         GRAPHICS SCREEN
03400     C
03500               CALL SCREEN(0)
03600               CALL WAIT
03700               CALL WAIT
03800               CALL WAIT
03900               RETURN
04000               END
```

```
00100           SUBROUTINE VTEST
00200   C
00300   C       VIEW AND FVIEW TEST
00400   C
00500           INTEGER FVIEW
00600           LOGICAL MSG(19)
00700           CALL CLS
00800           ENCODE(MSG,100)
00900   100     FORMAT('VIEW AND FVIEW TEST')
01000           CALL SETXY(0,0)
01100           CALL LOCATE(0)
01200           CALL GPRINT(19,MSG)
01300           CALL WAIT
01400   C
01500   C       DRAW VIEWPORT AND CIRCLES
01600   C
01700           CALL VIEW(0,40,639,239,0,1)
01800           CALL DCIRCL(1)
01900   C
02000   C       DRAW VIEWPORT AND LINES
02100   C
02200           CALL VIEW(20,50,619,229,1,0)
02300           CALL DLINE(0)
02400   C
02500   C       DRAW VIEWPORT AND CIRCLES
02600   C
02700           CALL VIEW(40,60,599,209,0,0)
02800           CALL DCIRCL(1)
02900   C
03000   C       DRAW VIEWPORT AND LINES
03100   C
03200           CALL VIEW(60,70,579,199,1,1)
03300           CALL DLINE(0)
03400   C
03500   C       CLEAR SCREEN
03600   C
03700           IX1=FVIEW(0)
03800           IY1=FVIEW(1)
03900           IX2=FVIEW(2)
04000           IY2=FVIEW(3)
04100           CALL VIEW(60-IX1,70-IY1,60+IX2,40+IY2,0,1)
04200           CALL CLS
04300           RETURN
04400           END
```

```
04500           SUBROUTINE DCIRCL(ICLR)
04600           CALL SETXY(100,100)
04700           DO 10 I=5,300,5
04800           CALL CIRCLE(I,ICLR,0.0,6.28,0.5)
04900    10     CONTINUE
05000           CALL WAIT
05100           RETURN
05200           END

05300           SUBROUTINE DLINE(ICLR)
05400           DO 11 I=2,200,4
05500           CALL SETXY(-10,-10)
05600           CALL SETXY(I+200,I)
05700           CALL LINE(ICLR,-1)
05800    11     CONTINUE
05900           CALL WAIT
06000           RETURN
06100           END
```

```
00100          SUBROUTINE WAIT
00200   C
00300   C      THIS SUBROUTINE INTRODUCES A TIME DELAY
00400   C
00500          DO 11 J=1,20
00600          DO 10 I=1,10000
00700   10     CONTINUE
00800   11     CONTINUE
00900          RETURN
01000          END
```

Appendix E/ Base Conversion Chart

DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
0	00	00000000	40	28	00101000
1	01	00000001	41	29	00101001
2	02	00000010	42	2A	00101010
3	03	00000011	43	2B	00101011
4	04	00000100	44	2C	00101100
5	05	00000101	45	2D	00101101
6	06	00000110	46	2E	00101110
7	07	00000111	47	2F	00101111
8	08	00001000	48	30	00110000
9	09	00001001	49	31	00110001
10	0A	00001010	50	32	00110010
11	0B	00001011	51	33	00110011
12	0C	00001100	52	34	00110100
13	0D	00001101	53	35	00110101
14	0E	00001110	54	36	00110110
15	0F	00001111	55	37	00110111
16	10	00010000	56	38	00111000
17	11	00010001	57	39	00111001
18	12	00010010	58	3A	00111010
19	13	00010011	59	3B	00111011
20	14	00010100	60	3C	00111100
21	15	00010101	61	3D	00111101
22	16	00010110	62	3E	00111110
23	17	00010111	63	3F	00111111
24	18	00011000	64	40	01000000
25	19	00011001	65	41	01000001
26	1A	00011010	66	42	01000010
27	1B	00011011	67	43	01000011
28	1C	00011100	68	44	01000100
29	1D	00011101	69	45	01000101
30	1E	00011110	70	46	01000110
31	1F	00011111	71	47	01000111
32	20	00100000	72	48	01001000
33	21	00100001	73	49	01001001
34	22	00100010	74	4A	01001010
35	23	00100011	75	4B	01001011
36	24	00100100	76	4C	01001100
37	25	00100101	77	4D	01001101
38	26	00100110	78	4E	01001110
39	27	00100111	79	4F	01001111

DEC.	HEX.	BINARY
80	50	01010000
81	51	01010001
82	52	01010010
83	53	01010011
84	54	01010100
85	55	01010101
86	56	01010110
87	57	01010111
88	58	01011000
89	59	01011001
90	5A	01011010
91	5B	01011011
92	5C	01011100
93	5D	01011101
94	5E	01011110
95	5F	01011111
96	60	01100000
97	61	01100001
98	62	01100010
99	63	01100011
100	64	01100100
101	65	01100101
102	66	01100110
103	67	01100111
104	68	01101000
105	69	01101001
106	6A	01101010
107	6B	01101011
108	6C	01101100
109	6D	01101101
110	6E	01101110
111	6F	01101111
112	70	01110000
113	71	01110001
114	72	01110010
115	73	01110011
116	74	01110100
117	75	01110101
118	76	01110110
119	77	01110111
120	78	01111000
121	79	01111001
122	7A	01111010
123	7B	01111011
124	7C	01111100
125	7D	01111101
126	7E	01111110
127	7F	01111111
128	80	10000000
129	81	10000001
130	82	10000010
131	83	10000011
132	84	10000100
133	85	10000101
134	86	10000110
135	87	10000111
136	88	10001000
137	89	10001001
138	8A	10001010
139	8B	10001011
140	8C	10001100
141	8D	10001101
142	8E	10001110
143	8F	10001111
144	90	10010000
145	91	10010001
146	92	10010010
147	93	10010011
148	94	10010100
149	95	10010101
150	96	10010110
151	97	10010111
152	98	10011000
153	99	10011001
154	9A	10011010
155	9B	10011011
156	9C	10011100
157	9D	10011101
158	9E	10011110
159	9F	10011111

DEC.	HEX.	BINARY	DEC.	HEX.	BINARY
160	A0	10100000	200	C8	11001000
161	A1	10100001	201	C9	11001001
162	A2	10100010	202	CA	11001010
163	A3	10100011	203	CB	11001011
164	A4	10100100	204	CC	11001100
165	A5	10100101	205	CD	11001101
166	A6	10100110	206	CE	11001110
167	A7	10100111	207	CF	11001111
168	A8	10101000	208	D0	11010000
169	A9	10101001	209	D1	11010001
170	AA	10101010	210	D2	11010010
171	AB	10101011	211	D3	11010011
172	AC	10101100	212	D4	11010100
173	AD	10101101	213	D5	11010101
174	AE	10101110	214	D6	11010110
175	AF	10101111	215	D7	11010111
176	B0	10110000	216	D8	11011000
177	B1	10110001	217	D9	11011001
178	B2	10110010	218	DA	11011010
179	B3	10110011	219	DB	11011011
180	B4	10110100	220	DC	11011100
181	B5	10110101	221	DD	11011101
182	B6	10110110	222	DE	11011110
183	B7	10110111	223	DF	11011111
184	B8	10111000	224	E0	11100000
185	B9	10111001	225	E1	11100001
186	BA	10111010	226	E2	11100010
187	BB	10111011	227	E3	11100011
188	BC	10111100	228	E4	11100100
189	BD	10111101	229	E5	11100101
190	BE	10111110	230	E6	11100110
191	BF	10111111	231	E7	11100111
192	C0	11000000	232	E8	11101000
193	C1	11000001	233	E9	11101001
194	C2	11000010	234	EA	11101010
195	C3	11000011	235	EB	11101011
196	C4	11000100	236	EC	11101100
197	C5	11000101	237	ED	11101101
198	C6	11000110	238	EE	11101110
199	C7	11000111	239	EF	11101111

DEC.	HEX.	BINARY
240	F0	11110000
241	F1	11110001
242	F2	11110010
243	F3	11110011
244	F4	11110100
245	F5	11110101
246	F6	11110110
247	F7	11110111
248	F8	11111000
249	F9	11111001
250	FA	11111010
251	FB	11111011
252	FC	11111100
253	FD	11111101
254	FE	11111110
255	FF	11111111

Appendix F/ Pixel Grid Reference

The following hexadecimal numbers include commonly used tiling designs.

Important Note: You cannot use more than two empty rows of tiles when tiling or you'll get an Illegal Function Call error.

Example (four rows of empty tiles):

```
CHR$(&HFF)+CHR$(&HFF)+CHR$(&H00)+CHR$(&H00)+CHR$(&H00)+CHR$(&H00)
```

gives you an Illegal Function Call error.

1. "X"

```
CHR$(&H41)+CHR$(&H22)+CHR$(&H14)+CHR$(&H08)+CHR$(&H14)
+CHR$(&H22)+CHR$(&H41)+CHR$(&H00)
```

								Hex	Decimal
0	1	0	0	0	0	0	1	41	65
0	0	1	0	0	0	1	0	22	34
0	0	0	1	0	1	0	0	14	20
0	0	0	0	1	0	0	0	08	8
0	0	0	1	0	1	0	0	14	20
0	0	1	0	0	0	1	0	22	34
0	1	0	0	0	0	0	1	41	65
0	0	0	0	0	0	0	0	00	0

2. "Fine" horizontal lines

CHR$(&HFF)+CHR$(&H00)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
0	0	0	0	0	0	0	0	00	0

3. "Medium" horizontal lines

CHR$(&HFF)+CHR$(&HFF)+CHR$(&H00)+CHR$(&H00)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
1	1	1	1	1	1	1	1	FF	255
0	0	0	0	0	0	0	0	00	0
0	0	0	0	0	0	0	0	00	0

4. Diagonal lines

(Right to left):
CHR$(&H03)+CHR$(&H0C)+CHR$(&H30)+CHR$(&HC0)

								Hex	Decimal
0	0	0	0	0	0	1	1	03	3
0	0	0	0	1	1	0	0	0C	12
0	0	1	1	0	0	0	0	30	48
1	1	0	0	0	0	0	0	C0	192

(Left to right)
CHR$(&HC0)+CHR$(&H30)+CHR$(&H0C)+CHR$(&H03)

								Hex	Decimal
1	1	0	0	0	0	0	0	C0	192
0	0	1	1	0	0	0	0	30	48
0	0	0	0	1	1	0	0	0C	12
0	0	0	0	0	0	1	1	03	3

5. "Fine" vertical lines

CHR$(&HAA)

								Hex	Decimal
1	0	1	0	1	0	1	0	AA	170

6. "Medium" vertical lines

CHR$(&HCC)

								Hex	Decimal
1	1	0	0	1	1	0	0	CC	204

7. "Coarse" vertical lines

CHR$(&HF0)

								Hex	Decimal
1	1	1	1	0	0	0	0	F0	240

8. One-pixel dots

CHR$(&H22)+CHR$(&H00)

								Hex	Decimal
0	0	1	0	0	0	1	0	22	34
0	0	0	0	0	0	0	0	00	0

9. Two-pixel dots

CHR$(&H99)+CHR$(&H66)

								Hex	Decimal
1	0	0	1	1	0	0	1	99	153
0	1	1	0	0	1	1	0	66	102

10. Pluses ("+")

CHR$(&H3C)+CHR$(&H3C)+CHR$(&HFF)

								Hex	Decimal
0	0	1	1	1	1	0	0	3C	60
0	0	1	1	1	1	0	0	3C	60
1	1	1	1	1	1	1	1	FF	255

11. Solid (all pixels ON)

CHR$(&HFF)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255

12. "Broad" cross-hatch

CHR$(&H92)+CHR$(&H92)+CHR$(&HFF)

								Hex	Decimal
1	0	0	1	0	0	1	0	92	146
1	0	0	1	0	0	1	0	92	146
1	1	1	1	1	1	1	1	FF	255

13. "Thick" cross-hatch

CHR$(&HFF)+CHR$(&HFF)+CHR$(&HDB)+CHR$(&HDB)

								Hex	Decimal
1	1	1	1	1	1	1	1	FF	255
1	1	1	1	1	1	1	1	FF	255
1	1	0	1	1	0	1	1	DB	219
1	1	0	1	1	0	1	1	DB	219

14. "Fine" cross-hatch

CHR$(&H92)+CHR$(&HFF)

								Hex	Decimal
1	0	0	1	0	0	1	0	92	146
1	1	1	1	1	1	1	1	FF	255

15. Alternating pixels

CHR$(&H55)+CHR$(&HAA)

								Hex	Decimal
0	1	0	1	0	1	0	1	55	85
1	0	1	0	1	0	1	0	AA	170

Appendix G/ Line Style Reference

Type	Binary Numbers	Hex	Decimal
Long dash	0000 0000 1111 1111	&H00FF	255
Short dash	1111 0000 1111 0000	&HF0F0	-3856
"Short-short" dash	1100 1100 1100 1100	&HCCCC	-13108
Solid line	1111 1111 1111 1111	&HFFFF	-1
OFF/ON	0101 0101 0101 0101	&H5555	21845
"Wide" dots	0000 1000 0000 1000	&H0808	2056
"Medium" dots	1000 1000 1000 1000	&H8888	-30584
"Dot-dash"	1000 1111 1111 1000	&H8FF8	-28680

Index

SERVICE POLICY

Radio Shack's nationwide network of service facilities provides quick, convenient, and reliable repair services for all of its computer products, in most instances. Warranty service will be performed in accordance with Radio Shack's Limited Warranty. Non-warranty service will be provided at reasonable parts and labor costs.

Because of the sensitivity of computer equipment, and the problems which can result from improper servicing, the following limitations also apply to the services offered by Radio Shack:

1. If any of the warranty seals on any Radio Shack computer products are broken, Radio Shack reserves the right to refuse to service the equipment or to void any remaining warranty on the equipment.
2. If any Radio Shack computer equipment has been modified so that it is not within manufacturer's specifications, including, but not limited to, the installation of any non-Radio Shack parts, components, or replacement boards, then Radio Shack reserves the right to refuse to service the equipment, void any remaining warranty, remove and replace any non-Radio Shack part found in the equipment, and perform whatever modifications are necessary to return the equipment to original factory manufacturer's specifications.
3. The cost for the labor and parts required to return the Radio Shack computer equipment to original manufacturer's specifications will be charged to the customer in addition to the normal repair charge.

RADIO SHACK, A DIVISION OF TANDY CORPORATION

U.S.A.: FORT WORTH, TEXAS 76102
CANADA: BARRIE, ONTARIO L4M 4W5

TANDY CORPORATION

AUSTRALIA	BELGIUM	U. K.
280-316 VICTORIA ROAD RYDALMERE, N.S.W. 2116	PARC INDUSTRIEL DE NANINNE 5140 NANINNE	BILSTON ROAD WEDNESBURY WEST MIDLANDS WS10 7JN

8749382-1282

Printed in USA