IMPORTANT SOFTWARE DISKETTE INFORMATION

For your own protection, do not use this product until you have made a backup copy of your software diskette(s). The backup procedure is described in the user's guide for your computer.

Please read the DISKID file on your new software diskette. DISKID contains important information including:

- The product name and revision number.
- The part number of the product.
- The date of the DISKID file.
- A list of the files on the diskette, with a description and revision number for each one.
- Configuration information (when applicable).
- Release notes giving special instructions for using the product.
- Information not contained in the current manual, including updates, additions, and deletions.

To read the DISKID file onscreen, follow these steps:

1. Load the operating system.
2. Remove your system diskette and insert your new software diskette.
3. Enter —

   TYPE DISKID

   and press Return.
4. The contents of the DISKID file is displayed on the screen. If the file is large (more than 24 lines), the screen display will scroll. Type ALT-S to freeze the screen display; type ALT-S again to continue scrolling.

Graphics Tool Kit, II

VICTOR Publications
380 El Pueblo Road
Scotts Valley, California 95066
(408) 438-6680

Second VICTOR printing April, 1984.

ISBN 0-88182-116-0 Printed in U.S.A.

OVERVIEW

The *Graphics Tool Kit, II* includes:

- **GRAFIX** — A powerful programming library that helps you program screen graphics. The GRAFIX programing commands provide such functions as drawing of multiple line types and widths, drawing of circles and arcs, filling of regions and bars, definition of cursor and fill patterns, logical combination of screens and windows, and columnar printing.
- **BUSIGRAF** — A business graphics package that allows you to make and edit pie charts, bar graphs, line plots, and organization charts.
- **CHARGRAF** — A character graphics system that lets you create and print graphics with characters instead of high-resolution graphics.
- **EFONT** — A font editor used to define or modify the character set of the shapes of symbols displayed on the screen.
- **KEYGEN** — A keyboard generator used to define the characteristics of individual keys on the keyboard.
- **GW-BASIC** — An interactive interpreter that supports the BASIC language, with extensions for music and high-resolution graphics.

In Applications Programmer's Toolkit II Vol I

GRAFIX

VICTOR Publications
380 El Pueblo Road
Scotts Valley, California 95066
(408) 438-6680

Second VICTOR printing January, 1984.

ISBN 0-88182-102-0 Printed in U.S.A.

CONTENTS

APPENDIXES

FIGURES

TABLES

CHAPTERS

1

INTRODUCTION

GRAFIX gives you a powerful set of commands to fully use the high resolution screen. GRAFIX provides such functions as:

- Drawing of multiple line types and widths
- Drawing of circles and arcs
- Filling of regions and bars
- Definition of cursor and fill patterns
- Logical combination of screens and windows
- Columnar printing

Screen file routines let you create screens or portions of screens, and then save them on disk. You can call up screen files whenever you need them. Furthermore, character sets created with the EFONT™ program can be read from disk and used in various modes, including double-size, reverse video, and proportional printing.

Since high resolution graphics requires a substantial amount of memory, the standard memory configuration of 128K bytes might not be sufficient for your needs. Refer to Chapter 2, "Memory Requirements," to see if you need additional memory for your application.

2

MEMORY REQUIREMENTS

With GRAFIX you can use up to ten different character sets of 128 characters each, and up to eight full screens. A character set uses slightly over 4K bytes; a screen uses exactly 40,000 bytes.

The specific number of screens that you can use is determined by the total amount of available memory in the system. Table 2-1 gives some examples which assume that you are using four character sets.

Table 2-1: Approximate User Space with GRAFIX

SYSTEM RAM CAPACITY	128K	256K	384K	512K
No. of Screens	SPACE AVAILABLE TO USER			
1	32K	180K	288K	416K
2	—	141K	249K	377K
3	—	102K	210K	338K
4	—	63K	171K	299K
5	—	23K	132K	260K
6	—	—	93K	221K
7	—	—	43K	182K
8	—	—	14K	143K

Note: The above values represent memory available for the user after allocating memory to the operating system (30K), GRAFIX (20K), and four character sets (4K each).

In a system with 512K bytes of RAM, you can use all ten character sets and all eight screens; 128K of RAM remain for your own use. For smaller RAM configurations, you must decrease the number of screens and character sets by an appropriate amount. In an application, you must ensure that enough memory is left in RAM for both your program and your run-time package.

You can use GRAFIX with any language that is capable of executing a print statement.

3

INSTALLING GRAFIX

To install GRAFIX, type **GRAFIX**, followed by a space, a dollar sign, and the letter **S**. Then type the number of screens you want to use, the letter **C**, and the number of extra character sets you want to activate.

To install GRAFIX with two character sets and two screens, for example, type the following command:


```
grafix $S2C2
```

To specify a printer, enter a third parameter, **P**, and one of the following:

F or **M**	Epson
T	Tally
C or **S**	C. Itoh
O	Okidata
N	No printer

For example, to install GRAFIX with one character set, three screens, and the printer driver for an Epson printer, type the following command:

```
grafix $S3C1PF
```

If you type the GRAFIX command without any parameters, you install GRAFIX with only one screen and the standard system character sets. Once GRAFIX is installed, you must reboot the system to return to any other mode.

3.1 CHARACTER SET PARAMETER

The minimum number of character sets available is two. The first 128 ASCII characters and the upper 128 graphics characters are the two system character sets; neither of these sets is ever overwritten. The maximum number of character sets that you can specify is eight (a total of ten, including the two system character sets).

If you specify one character set (for a total of three), you can load only one additional character set at a time. Each time you load a new character set, the previously loaded character set is overwritten.

The only reason to specify an additional character set is to minimize disk access. If an application requires many different character sets, you can load all the sets once. Then the sets are resident in RAM, and you can access them at any time without repeatedly loading from disk.

3.2 SCREEN PARAMETER

The minimum number of screens you can specify is 1, and the maximum is 8. Refer to Table 2-1 to select the number of screens.

4

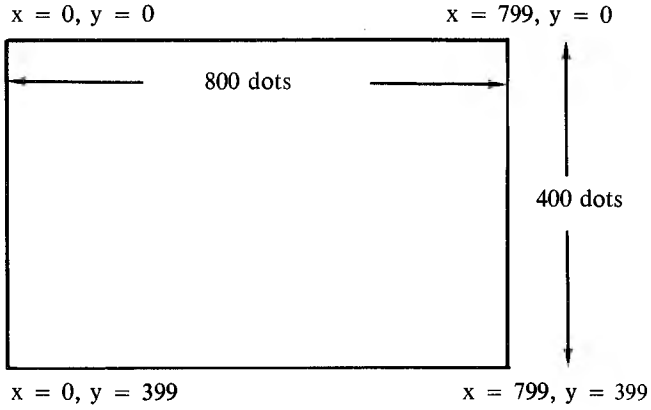
USING HIGH RESOLUTION GRAPHICS

THE HIGH RESOLUTION SCREEN 4.1

The high resolution screen consists of 320,000 individual dots. There are 800 dots in each row and 400 in each column.

You can describe any point on the screen by using its horizontal (x) and vertical (y) coordinates. The coordinates range from 0,0 in the upper left corner to 799,399 in the lower right corner. Figure 4-1 displays the coordinates of the corners of the screen.

Figure 4-1: Screen Coordinates



4.1.1 THE DISPLAY SCREEN AND THE WORK SCREEN

There are two types of screens in GRAFIX:

- DISPLAY SCREEN—currently displayed on the video screen
- WORK SCREEN—where you send GRAFIX commands

If you make the Display Screen and the Work Screen the same screen, then the effects of the GRAFIX commands appear on the video screen. If they are not the same, the effects of the commands do not appear on the Display Screen until the Display Screen number is switched to match the Work Screen number, or vice versa.

Regardless of how many screens you install when booting GRAFIX, you can have only one Display Screen and one Work Screen. But you can easily reassign Display Screen or Work Screen status to any other screen. The escape sequence used to change the Display Screen is **Esc 5 B**. The sequence used to change the Work Screen is **Esc 5 A**.

4.1.2 THE TEXT WINDOW

With this feature, you can define a text window with any number of lines anywhere on the screen. The text window simulates normal VT52 terminal operation under Hi-Res and implements most of the normal functions and their escape sequences (see Appendix A). The only difference from normal character mode operation is that all cursor positioning, insert, and delete functions refer to the text window rather than to the whole screen. When the bottom of the text window is reached, the text scrolls up. Any normal printout, such as an error message, uses the normal system character set and is directed to the text window. For example, if you print Esc H (which normally sets the cursor at Home Position), you will set the cursor to the upper left corner of whatever *text window* you have defined.

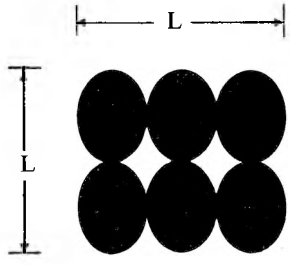
The text window can range from rows 3 to 24; the default is lines 21 to 24 at the bottom of the display. The text window is always on the currently viewed screen. The escape sequence used to specify the range of the text window is **Esc m2 p1p2** (described in Appendix A).

When you debug programs using Hi-Res graphics, do all Hi-Res printing and drawing on a Work Screen (for example, Screen 1), and leave Screen 0 totally in text mode. It is easy to select the Work Screen as the Display Screen in order to view the Hi-Res graphics. Select Screen 0 as the Display Screen to return to programming. (See Chapter 5 for descriptions of the Select Display Screen and Select Work Screen functions.) Program listings and error messages are printed in the text window in Screen 0; they do not disturb the graphics.

THE ASPECT RATIO 4.1.3

The aspect ratio is the ratio of x units to y units needed to provide equal length in both directions. The video screen has an aspect ratio of 3 to 2. Figure 4-2 shows the dots that make up a small square on the screen. The horizontal sides of the square have 1 1/2 times the number of vertical dots (3:2).

Figure 4-2: The Aspect Ratio



If you want to draw a ten-unit square on the screen, follow these steps: draw a 15-unit line in the x direction, a 10-unit line in the y direction, a 15-unit line in the negative x direction, and finally a 10-unit line in the negative y direction. To display a circle on the screen, you must draw an ellipse.

4.2 THE CHARACTER SET

A character set contains 128 characters. Each character is defined within an array which is 16 dots wide and 16 dots high.

The Character Height parameter specifies the height of the character; it is measured in dots, starting from the bottom row of the array. This number is constant for all the characters within a character set.

The Character Width parameter specifies the width of the character, measured in dots starting in the leftmost column of the array and moving to the right. In a given character set, this parameter can be fixed or can take on a different value for each character.

Each character set file contains the parameters used by GRAFIX to print characters correctly from that character set. The character set file headers contain flags specifying whether a character is:

- Normal or Special
- Horizontal or Vertical
- Fixed-width or Proportional

A normal character set is 10 dots wide and 16 dots long. It is booted at the same time as the system.

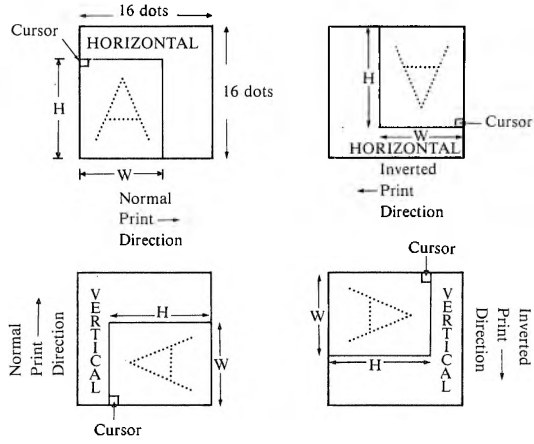
Once a character from any character set has been placed on the Hi-Res Screen, the character set doesn't need to remain in RAM for the character to be displayed on the screen. (In normal mode, the character set must reside in RAM to display any character from that set.)

Because of the screen's 3:2 aspect ratio, you must use different character sets for horizontal and vertical printing. A horizontal character set is normally printed left to right, and a vertical character set is normally printed bottom to top.

To print vertically, first specify a vertical character set. Then enter both the Hi-Res print escape sequence and the string of characters to be printed. Printing will be vertical until you return to horizontal printing.

The portion of the 16-by-16-dot character that is printed, together with the direction of print, is shown in Figure 4-3.

Figure 4-3: Character Orientation



5

GRAFIX COMMANDS

GRAFIX has the following four types of commands:

1. Parameter Input
2. Parameter Return
3. Action—Screen Draw, Print, or Move
4. File Maintenance

PARAMETER INPUT COMMANDS 5.1

- Select the screen to draw on
- Select the screen to be displayed
- Select the fill pattern
- Define a screen window
- Select a character set
- Select a cursor type
- Select a combination rule
- Set relative and absolute cursor positioning
- Set line width
- Set line type
- Set left margin
- Set a dot

- Set and reset superscript mode
- Set and reset subscript mode
- Set and reset invert character and print direction
- Set and reset double character size mode
- Define text window
- Define user cursor
- Define user fill pattern
- Enable and disable cursor
- Enable and disable shadow print
- Set and reset reverse video mode
- Set and reset underline mode
- Save GRAFIX cursor position

5.2 PARAMETER RETURN COMMANDS

- Get enabled screen number
- Get displayed screen number
- Get window parameters
- Get dot
- Get character width
- Get character height
- Get character type
- Get GRAFIX cursor

ACTION COMMANDS 5.3

- Fill a region
- Fill a bar
- Draw a circle
- Draw an arc
- Absolute and relative line draw
- Move a window or a screen
- Initialize
- Hi-Res print
- Clear screen
- Return GRAFIX cursor to previously saved position
- Toggle text window screen

FILE MAINTENANCE COMMANDS 5.4

- Save a window on disk
- Load a window from disk
- Select character set

USING THE GRAFIX ESCAPE SEQUENCES 5.5

All GRAFIX escape sequences begin with **Esc 5**. Many of the functions require parameters; others return values. The rest of the functions are simple commands and have no parameters. Any function call (escape sequence) must be terminated with a carriage return/line feed (ASCII 0D, 0A).

You must call the functions in sequence; that is, the first function must end before you call the next one. Note that only integer parameters are accepted.

The following program segments show how each of the four types of GRAFIX functions is used (Parameter Input, Parameter Return, Action, and File Maintenance).

IN BASIC

```
10 E$=CHR$(27) : G$=E$+"5"   'initialize GRAFIX call string
20 PRINT G$;"2"              'clear the Hi-Res screen
30 PRINT G$;"Q";400;200      'put cursor at 400,200
40 PRINT G$;"P";100          'draw circle with radius 100
50 PRINT G$;"u"              'get graphics cursor position
60 INPUT X                   'GRAFIX will return values into
70 INPUT Y                   'the input statements
80 PRINT G$;"iscript"        'load new font from disk (script)
90 PRINT G$;"pEND."          'print to Hi-Res screen
100 END
```

5

IN PASCAL

```
PROCEDURE grafix example;

CONST
     escape = chr (27);
     grafix = escape * '5';

VAR
     screen x : 0..799;
     screen y : 0..399;

BEGIN

     (* all statements perform the same functions
        as before*)
     writeln(grafix,'2');           (* Action Command *)
     writeln(grafix,'Q',400,200);(* Parameter Input *)
     writeln(grafix,'P',100);     (* Parameter Input *)
     writeln(grafix, 'u');       (* Parameter Return *)
     readln(screen x);             (* returned parameter *)
     readln(screen y);             (* returned parameter *)
     writeln(grafix,'iscript');  (* File Maintenance *)
     writeln(grafix,'pEND.');     (* Parameter Input *)

END  (* grafix example *);
```

ESC 5 A—SELECT WORK SCREEN

Action: Selects the screen specified as the Work Screen. All line-draws and Hi-Res prints are directed to the Work Screen.

Parameters passed: Single ASCII number between 0 and 7 corresponding to the screen desired.

Explanation: If only one screen is enabled when GRAFIX is loaded, the Work Screen and the Display Screen are the same screen. If more than one screen is enabled, you can program on Screen 0 (after specifying it as the Display Screen), but select Screen 1 as the Work Screen. Then, once drawing is completed, you can select Screen 1 as the Display Screen and view the completed picture, rather than its formation.

ESC 5 B—SELECT DISPLAY SCREEN

Action: The specified screen is displayed.

Parameters passed: An ASCII number specifying which screen is to be viewed on the CRT. Current hardware supports selection of Screen 0 or Screen 1.

Explanation: As before, if only one screen is specified when GRAFIX is installed, Screen 0 is both the Work and Display Screen at all times. If multiple screens are enabled on installation, Screen 0 or 1 may be selected as the Display Screen.

ESC 5 C—SET SUPERSCRIPT SHIFT MODE

Action: Shifts the cursor up by the number of dots specified in the currently enabled character set.

Parameters passed: None.

Explanation: The cursor shifts up a number of dots, usually equal to about one-half the character height stored in the currently enabled character set. This function is used to print superscripts.

ESC 5 D—RESET SUPERSCRIPT SHIFT MODE

Action: Moves cursor down the number of dots specified in the table for the currently enabled character set.

Parameters passed: None.

Explanation: This is the complement to Set Superscript Shift Mode. A reset is required to return to printing on the normal line.

ESC 5 E—SET SUBSCRIPT SHIFT MODE

Action: Moves cursor down the number of dots specified in the table for the currently enabled character set.

Parameters passed: None.

Explanation: Similar to Set Superscript Shift Mode, except that the shift is downward for the printing of subscripts.

ESC 5 F—RESET SUBSCRIPT MODE

Action: Moves cursor up the number of dots specified in the table for the currently enabled character set.

Parameters passed: None.

Explanation: Similar to Reset Superscript Shift Mode.

ESC 5 G—SET DOUBLE CHARACTER SIZE MODE

Action: Causes all further Hi-Res print to be double the normal size.

Parameters passed: None.

Explanation: Doubles the width and the height of each character, regardless of the character set enabled (a single character dot is blown up into four dots).

ESC 5 H—RESET DOUBLE CHARACTER SIZE MODE

Action: Returns all Hi-Res print to normal size.

Parameters passed: None.

ESC 5 I—DEFINE SCREEN WINDOW

Action: Stores current window dimensions in a table for subsequent move instructions.

Parameters passed: Two ASCII numbers: One specifies the width of the window (x extent), and the other specifies the height of the window (y extent). The cursor position specifies the upper left corner of the window.

Explanation: This function is used prior to the Move function or to the Save Window function. It is not a clipping window.

ESC 5 J—SET INVERT CHARACTER AND PRINT DIRECTION

Action: Changes print direction and inverts characters.

Parameters passed: None.

Explanation: If the currently enabled character set is horizontal, this function flips the characters upside down and prints from right to left. If the current character set is vertical, this function flips it over and prints from the top down.

ESC 5 K—RESET INVERT CHARACTER AND PRINT DIRECTION

Action: Returns to normal print direction.

Parameters passed: None.

Explanation: This function restores the default print direction.

ESC 5 L—SELECT FILL PATTERN

Action: Stores the selected fill pattern number in a table for use by any subsequent fill command.

Parameters passed: A single ASCII number specifying the number for the fill pattern. The number must be between 0 and 8.

Explanation: Fill patterns 0-7 are preprogrammed. The density of the fill pattern increases with the fill pattern number. Fill pattern number 8 is blank. Each of the nine fill patterns can be specified by selecting the Define User Fill Pattern function.

ESC 5 M—FILL REGION

Action: Fills the bounded region (specified by the cursor position within the region) with the currently enabled pattern.

Parameters passed: None.

Explanation: The region to be filled is specified by the position of the cursor. If the cursor is pointing to a dot that is ON when the Fill Region command is executed, no error is generated and no region is filled.

The region need not be bounded on all sides; the edges of the screen act as boundaries. Complex regions can be filled; however, if the region to be filled contains more than 64 discontinuities, a table overflow error will result.

ESC 5 N—FILL BAR

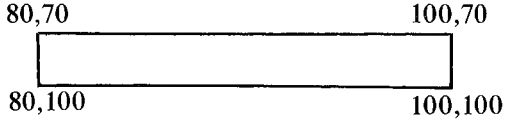
Action: The fill is accomplished using the currently enabled fill pattern.

Parameters passed: Two ASCII numbers specifying the relative x and y coordinates of the corner of the bar to be filled.

Explanation: The bar is a rectangular region. A reference corner is specified by the current position of the cursor. The diagonally opposite corner of the rectangular region is specified by the parameters passed to the Fill Bar function.

Note: The coordinates passed to the function are relative to the current cursor position. The x and y values may be positive, negative, or any combination.

Example: If the cursor is positioned at 100,100 and the Fill Bar function is called with the parameters −20 and −30, a bar with the following coordinates is drawn:



The bar will be filled using the currently selected fill pattern. None of the combination rules apply—the function erases the portion of the screen and fills it with the selected fill pattern.

ESC 5 O—SET LEFT MARGIN

Action: Stores left margin.

Parameters passed: An ASCII number ($0 \leq n \leq 799$).

Explanation: Sets the left margin to be used with Hi-Res print.

ESC 5 P—DRAW CIRCLE

Action: Draws a circle centered at the current cursor position.

Parameters passed: An ASCII number specifying radius in y units.

Explanation: To display a circle on the screen, this function draws an ellipse with an aspect ratio of 3:2.

Example: If the cursor position is 200,200 and the function is called passing a radius of 100, a circle is drawn with a radius of 100 dots in the y direction, and a radius of 150 dots in the x direction.

ESC 5 Q—ABSOLUTE POSITION GRAFIX CURSOR

Action: Moves the GRAFIX cursor to the absolute x,y position.

Parameters passed: Two ASCII numbers specifying the x and y positions (coordinates) on the screen.

Explanation: The GRAFIX cursor can be positioned anywhere within the cursor space as shown in Figure 7-1. Attempting to position the cursor outside that space generates an error.

ESC 5 R—RELATIVE POSITION GRAFIX CURSOR

Action: Moves the GRAFIX cursor relative to the current cursor position by the increments specified.

Parameters passed: Two ASCII numbers specifying the x and y increments.

Explanation: Same as Absolute Position GRAFIX Cursor function except the cursor is positioned relative to its prior location.

ESC 5 S—SAVE WINDOW ON DISK

Action: Saves the contents of the defined window from the Work Screen in the named disk file with extension ".SCR".

Parameters passed: Up to 64 ASCII characters specifying a valid pathname.

Explanation: Before calling this function, move the cursor to the upper left corner of the desired window. Then specify the extent of the window using the Define Screen Window function.

A whole screen can be defined as the window and saved on disk by using this function, but the window must be entirely on the screen.

ESC 5 T—LOAD WINDOW FROM DISK

Action: Loads the contents of the named file onto the currently enabled screen. The upper left corner of the window coincides with the current cursor position. The file can contain a whole screen or a partial screen. If the file exceeds the screen boundary, it is clipped.

Parameters passed: Up to 64 ASCII characters specifying a valid pathname.

Explanation: The file name must not contain an extent. The named file must be on a specified drive and must have the extension ".SCR". The cursor must be on the screen when this routine is called.

ESC 5 U—DRAW LINE (ABSOLUTE)

Action: Draws a line from the cursor position to the specified x and y coordinates using the current combination rule, line width, and line type. The cursor is moved to the new x,y position.

Parameters passed: Two ASCII numbers corresponding to the absolute x and y position on the screen.

Explanation: This function draws a line from the previous cursor position to the absolute x and y coordinates specified, and moves the cursor to the end of the line.

ESC 5 V—MOVE WINDOW

Action: Moves the window using the currently enabled combination rule.

Parameters passed: An ASCII number specifying the destination screen.

Explanation: Moves a region from the Work Screen (defined by the enabled window) to the destination screen, using the cursor position as the upper left corner of the window. The function selects the specified screen as the Work Screen when the move is done. The window can be moved from one location to another on the same screen and from one screen to another. The current combination rule remains in effect.

ESC 5 W—MOVE SCREEN

Action: Moves the current Work Screen to the screen specified, using the current combination rule. Selects specified screen as the Work Screen when move is done.

Parameters passed: An ASCII number specifying the destination screen.

Explanation: This is a special case of Move Window, where the window is the whole screen. The move is accomplished faster than with the Move Window function.

ESC 5 X—SET COMBINATION RULE

Action: Stores the currently selected combination rule.

Parameters passed: An ASCII number corresponding to one of 16 combination rules. These are described in Chapter 9.

Explanation: This rules applies to all moves, character prints, and line draws, but not to arc draws, circle draws, or fills.

ESC 5 Y—SET LINE WIDTH

Action: Stores specified line width value.

Parameters passed: An ASCII number corresponding to the dot width (y units) desired for the line drawing routine. The supported numbers are 1, 2, 4, and 6. Lines other than horizontal lines are automatically scaled to give equal widths.

Explanation: The specified value is used in all line, circle, and arc drawings.

ESC 5 Z—SET LINE TYPE

Action: Stores line type value in table.

Parameters passed: An ASCII number corresponding to one of five types of lines:

- Type 1—Solid line.
- Type 2—Dashed line: 4 dots ON, 4 dots OFF.
- Type 3—Dashed line: 8 dots ON, 4 dots OFF.
- Type 4—Dot, dash: 4 dots ON, 2 dots OFF, 2 dots ON, 2 dots OFF.
- Type 5—Dot, dash: 8 dots ON, 3 dots OFF, 3 dots ON, 3 dots OFF.

Explanation: The line type applies to all line, circle, and arc drawing.

ESC 5 a—GET CHARACTER WIDTH

Action: Reads the width of the specified character and returns it as an ASCII number.

Parameters passed: A single ASCII character.

Explanation: This is used with proportional printing.

ESC 5 b—GET DOT

Action: Returns the attribute of the dot pointed to by the GRAFIX cursor.

Parameters passed: None.

Explanation: If the dot at the cursor position is ON, the value returned is 1; if it is OFF, the value returned is 0.

ESC 5 c—SET DOT

Action: Specifies the source dot, and combines it with the dot pointed to by the cursor, according to the set rule.

Parameters passed: A single ASCII character (1 or 0) specifying the source dot as ON or OFF.

Explanation: If an ASCII 1 is passed, the source dot is specified as being ON; if an ASCII 0 is passed, the source dot is OFF. (See Chapter 9.)

ESC 5 d—INITIALIZE

Action: Resets all functions to their initial values.

Parameters passed: None.

Explanation: The initial parameter values are:

Cursor Position	0,0
Window Position	0,0
Screen Window	800,400
Character Set	NORMAL
Combination Rule	7 (OR)
Work Screen	0
Display Screen	0

Line Type	SOLID
Line Width	1
Left Margin	0
Shadow Print	OFF
Underline	OFF
Reverse Video	OFF
Double Size	OFF
Print Direction	NORMAL
Cursor	ON
Cursor Type	ARROW
Cursor Saved	0,0
User Cursor x Offset	0
User Cursor y Offset	0
Fill Pattern	0

ESC 5 e—GET WINDOW

Action: Returns four ASCII numbers specifying the current GRAFIX window. The first pair of numbers specifies the x and y coordinates of the upper left corner of the window; the next pair specifies the width and height of the window.

Parameters passed: None.

ESC 5 f—DRAW LINE (RELATIVE)

Action: Draws a line from the cursor position to the dot specified by the increment arguments, x and y.

Parameters passed: Two ASCII numbers specifying the x and y increments.

ESC 5 h—DRAW ARC

Action: Draws a circular arc between two endpoints in a counterclockwise direction.

Parameters passed: Five ASCII numbers specifying the radius (in y units) and the x and y coordinates of the two endpoints of the arc.

Explanation: An arc with the specified radius is drawn between the two endpoints. An error is generated if the x and y coordinates for the endpoints do not fall on a circle of the specified radius.

ESC 5 i—SELECT CHARACTER SET

Action: Selects the named character set for all subsequent Hi-Res print.

Parameters passed: File name of the desired character set.

Explanation: If the character set is not in memory, it is loaded from disk. The least recently accessed character set is overwritten if all allocated set locations are used. The system character set is protected from overwriting and is accessed by selecting NORMAL. The system graphics character set (ALT) is also protected.

A drive may be specified when you enter the file name. If no drive is specified, the default drive is assumed. File names are assumed to have a .CHR extension which can be neither specified nor overridden. The same character set name should not be used from more than one disk drive.

ESC 5 j—GET WORK SCREEN NUMBER

Action: Returns the ASCII number of the currently selected Work Screen.

Parameters passed: None.

ESC 5 k—GET DISPLAY SCREEN NUMBER

Action: Returns the ASCII number (0 or 1 for current hardware) indicating the current Display Screen.

Parameters passed: None.

ESC 5 l—GET CHARACTER HEIGHT

Action: Returns the ASCII value specifying the height of the currently selected character set.

Parameters passed: None.

Explanation: All characters in the character set have the same height.

ESC 5 m—DEFINE USER CURSOR

Action: Stores the specified pattern as user cursor in the table for future use.

Parameters passed: 34 ASCII numbers. The first two numbers represent the desired x and y offset from the upper left corner to the cursor dot within the pattern. The last 32 numbers represent the cursor pattern ($0 \leq n \leq 255$).

Explanation: The GRAFIX cursor, whatever its shape, always points to a single dot on a screen. The cursor character is a pattern 16 dots wide by 16 dots high. The cursor dot is specified by the x and y offsets with respect to the upper left corner of the cursor pattern.

Each of the last 32 ASCII numbers defines 8 dots of the cursor pattern. Each pair of numbers defines a row of the cursor pattern. For example, if you want to define an arrow, specify the following pattern:

	Number Pair	Pattern	
Row 1	255,255	11111111	11111111
Row 2	192,0	11000000	00000000
Row 3	160,0	10100000	00000000
Row 4	148,0	10010000	00000000
Row 5	136,0	10001000	00000000
Row 6	132,0	10000100	00000000
Row 7	2,0	00000010	00000000
Row 8	1,0	00000001	00000000
Row 9	0,128	00000000	10000000
Row 10	0,64	00000000	01000000
Row 11	0,32	00000000	00100000
Row 12	0,0	00000000	00000000
Row 13	0,0	00000000	00000000
Row 14	0,0	00000000	00000000
Row 15	0,0	00000000	00000000
Row 16	0,0	00000000	00000000

ESC 5 n—DEFINE USER FILL PATTERN

Action: Stores the specified fill pattern in the appropriate location in the table.

Parameters passed: 33 ASCII numbers. The first number specifies the fill number this pattern will be accessed by; the remaining 32 specify the 16-by-16-dot pattern used as the fill pattern.

Explanation: A fill pattern is defined as a 16 × 16 array of dots. Each ASCII number defines 8 dots, the first two numbers define the first row of dots, the second two define the second row, and so on, until the sixteenth row. See the explanation under "Define User Cursor."

Fill patterns 0–7 are programmed with increasing density as the number increases. Fill pattern 8 is completely blank. Any of the nine fill patterns can be defined and specified.

ESC 5 o—GET CHARACTER SET TYPE

Action: Returns an ASCII number specifying the selected character set type as follows:

0—Normal character set booted with the system (10 dots wide, 16 dots high).

4—Alternate set booted with the system.

8—Horizontal, Non-proportional.

9—Horizontal, Proportional.

10—Vertical, Non-proportional.

11—Vertical, Proportional.

12—Special, Horizontal, Non-proportional.

13—Special, Horizontal, Proportional.

14—Special, Vertical, Non-proportional.

15—Special, Vertical, Proportional.

Parameters passed: None.

ESC 5 p—HI-RES PRINT

Action: Prints the characters at the current graphics cursor position and updates the cursor position.

Parameters passed: Printable ASCII characters terminated by carriage return/line feed.

ESC 5 q—ENABLE CURSOR

Action: Displays the currently selected cursor at the cursor position.

Parameters passed: None.

ESC 5 r—DISABLE CURSOR

Action: Does not display the cursor.

Parameters passed: None.

Explanation: Even though the cursor is not visible, you can position it anywhere within the cursor positioning space.

ESC 5 s—ENABLE SHADOW PRINT

Action: Shadow prints all characters using Hi-Res print.

Parameters passed: None.

Explanation: Shadow printing is accomplished by printing a character twice. On the second printing, the character is moved one dot to the right.

ESC 5 t—DISABLE SHADOW PRINT

Action: Returns to normal print.

Parameters passed: None.

ESC 5 u—GET GRAFIX CURSOR

Action: Returns two ASCII numbers. The first corresponds to the absolute x position of the cursor, the second to the absolute y position.

Parameters passed: None.

ESC 5 v—ENTER REVERSE VIDEO MODE

Action: Causes all further Hi-Res print to be in reverse video.

Parameters passed: None.

ESC 5 w—RESET REVERSE VIDEO MODE

Action: Returns to normal print.

Parameters passed: None.

ESC 5 x—SELECT CURSOR TYPE

Action: Uses the specified cursor.

Parameters passed: An ASCII number as follows:

0—Block
1—Cross-hair
2—Arrow (the default)
3—User-defined

ESC 5 y—SET UNDERLINE MODE

Action: All characters printed from this point are underlined. (The second row of dots from the bottom of the character cell is turned ON.)

Parameters passed: None.

ESC 5 z—RESET UNDERLINE MODE

Action: Returns to normal print.

Parameters passed: None.

ESC 5 0—SAVE GRAFIX CURSOR POSITION

Action: Saves the current GRAFIX cursor position for subsequent return.

Parameters passed: None.

ESC 5 1—RETURN GRAFIX CURSOR TO PREVIOUSLY SAVED POSITION

Action: GRAFIX cursor returns to position saved when Esc 5 0 was executed and stays on the screen.

Parameters passed: None.

ESC 5 2—CLEAR SCREEN

Action: Sets all dots on the Work Screen to zero.

Parameters passed: None.

ESC 5 9—TOGGLE TEXT WINDOW SCREEN

Action: Successively enables and disables printing to the text window.

Parameters passed: None.

ESC 5 ?—SCREEN DUMP TO DOT MATRIX PRINTER

Parameters passed: None.

Action: Prints the Work Screen on the printer.

Explanation: The type of printer, if any, is identified at installation. You must install the printer correctly.

6

HI-RES PRINTING

CHARACTER SETS 6.1

The dot pattern for any character is contained in a 16-dot-wide by 16-dot-high matrix. Each character set has two dimension attributes: height and width.

Character height (H) specifies the height (in dots) of the field to be covered by the characters. Character height is the same for all 128 characters in the set.

Character width (W) is the same for all characters in the set (non-proportional printing), or depends on the width of each character (proportional printing). In proportional printing, for example, the letter W is wider than the letter I. Each character in a set is defined by a window starting at the lower left of the 16 × 16 dot character array. This window has a width specified for that character and a height specified for the whole character set. Essentially, any given character is a "small window" which is being transferred to the selected screen.

Since all characters within any given set have the same height, the cursor proceeds along the top of the boundary defined for the characters as they are printed (if you're using only one character set).

The Hi-Res print function accepts a string of characters to be printed and automatically allocates (from left to right) the amount of space needed for each character. If the character set is proportionally spaced, the printing on the screen is also proportionally spaced (the cursor automatically advances to the next character position each time). You must manage the line width—automatic end-of-line wrap-around is not implemented.

If you execute a Hi-Res carriage return/line feed (ASCII code 160, 161) within a string, the Hi-Res cursor advances down and to the left, and then prints the rest of the string. (A normal return/line feed terminates

the Hi-Res print function.) The specific number of dots the cursor moves down is determined by the height parameter for the character set. The cursor movement to the left is limited by the current left margin.

In contrast to printing on a terminal, the cursor is not a block equal to the size of a character. Instead, the cursor points to a single dot. This dot is at the upper left corner of the character field.

The Hi-Res print routine prints any character between 32 and 127 when it receives the corresponding ASCII code. Characters 0 through 31 within a character set are accessed by sending ASCII codes of 128 through 159.

6.2 THE HI-RES PRINT FUNCTION

The Hi-Res print function lets you print character strings on a graphics screen. Place the cursor at the spot where you want the upper left corner of the first character in the string. Then call the Hi-Res print function, with the string of characters you want to print as an argument. Terminate the function with a carriage return/line feed.

Before executing the print function you can specify:

- The type font
- Shadow printing
- Reverse video
- Underlining
- Subscript and superscript printing
- Double character size
- The left margin

These features are described in the following examples. (g$ is equivalent to the escape character plus "5".)

1. To print with a different type font (character set), use the Select Character Set function (Esc 5 i). For example, if you want to load a set called "script" located on drive B, execute this command:

   print g$ + "ib:script"

2. With shadow printing, a character is printed a second time by ORing-in the same character one dot to the right. To shadow print a string of characters, use the Enable Shadow Print function (Esc 5 s). For example, if you want to shadow print the phrase "this is important," execute Esc 5 s, print the string using the print function, then turn off shadow print with the Disable Shadow Print function (Esc 5 t):

   print g$ + "s"
   print g$ + "p" + "this is important"
   print g$ + "t"

3. To print a word in reverse video, use the Set Reverse Video Mode function (Esc 5 v). For example, to print the word "Warning" in reverse video, execute the reverse video function, print the word, and then exit reverse video:

   print g$ + "v"
   print g$ + "pWarning"
   print g$ + "w"

4. To underline a character string, use the Set Underline Mode function (Esc 5 y). For example:

   print g$ + "y"
   print g$ + "p" + the_string$
   print g$ + "z" 'to reset underline mode

5. Superscript and Subscript modes move the cursor up or down a distance equal to one-half the character height. To print a superscript, use the Set Superscript Shift Mode function (Esc 5 C). To print a subscript, use the Set Subscript Shift Mode function (Esc 5 E).

For example:

Superscript:

```
print g$ + "C"
print g$ + "p2"
print g$ + "D"
print g$ + "p + c"
print g$ + "C"
print g$ + "p2"
print g$ + "D"
print g$ + "p = 0"
```

Subscript:

```
print g$ + "pH"
print g$ + "E"
print g$ + "p2"
print g$ + "F"
print g$ + "pO"
```


If you change the character font size, you must position the cursor exactly if you print superscripts or subscripts.

6. When you call the function Set Double Character Size Mode (Esc 5 G) all subsequent graphics characters are printed double their normal width and height. You set and reset (Esc 5 H) the mode as described in the preceding explanations.

7. Esc 5 J, Set Invert Character And Print Direction, causes all subsequent graphics characters to be printed upside-down and from right to left. (If you're using a vertical character set, then characters are flipped over and printed from top to bottom.) Set and reset (Esc 5 K) the mode as described in the preceding explanations.

PROPORTIONAL PRINTING 6.3

If you use a proportionally spaced character set, GRAFIX automatically provides the correct amount of space between characters. (You don't need a "print proportional" function.) However, you must keep track of the location of the right margin.

Use the Get Character Width function (Esc 5 a) to keep a running total of the width of the characters, and compare it to the coordinate of the right margin.

You can do microspacing by using the Relative Position Cursor function (ESC 5 R).

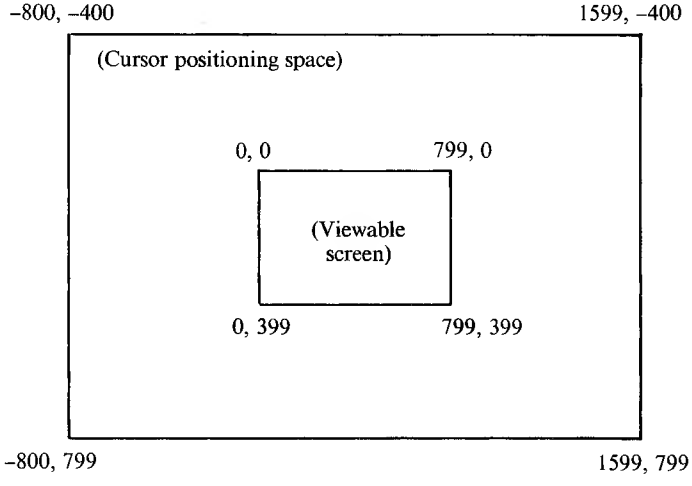
7

THE CURSOR

POSITIONING THE CURSOR 7.1

You can position the cursor anywhere within the actual screen—that is, in the space between the limits of −800 and +1599 for x and −400 and +799 for y. The total cursor space is shown in Figure 7-1. All cursor positioning refers to the currently selected screen.

Figure 7-1: Total Cursor Space



The cursor may be selected for printing (cursor ON) or non-printing (cursor OFF) within the actual screen, but it defaults to non-printing if you position it outside the actual screen. To move the cursor from one screen to another, simply change the Work Screen.

7.2 POINTING THE CURSOR

The GRAFIX cursor always points at a single dot on the screen. Table 7-1 indicates the target dots of the different cursors.

Table 7-1: Targets of GRAFIX Cursors

TYPE OF CURSOR	TARGET DOT
Block Cursor	The upper left corner of the block is the absolute x and y position of the cursor.
Arrow Cursor	The dot corresponding to the tip of the arrow is the absolute x and y location of the cursor.
Cross-hair Cursor	The dot in the middle of the cross-hair specifies the absolute x and y position of the cursor.
User-defined Cursor	The absolute location of the dot pointed to is the upper left corner of a 16 × 16 dot array specified by the x and y cursor offsets.

7.3 MULTIPLE-CURSOR PROBLEMS

GRAFIX supports two cursors: the Text cursor and the GRAFIX cursor. You can position the GRAFIX cursor anywhere within the cursor positioning space; you can position the Text cursor anywhere within the text window.

When a GRAFIX command to print or draw is issued, the GRAFIX cursor is removed from the screen, the command is executed, and the GRAFIX cursor is then XORed back into the screen. When the cursor is moved, it is XORed out of the current position and XORed into the new position. The GRAFIX cursor never disturbs any of the graphics on the screen. The Text cursor, a block, is also XORed into the screen and XORed out when it is moved.

Two kinds of screen erase are possible. The GRAFIX Clear Screen function (Esc 5 2) erases the entire Work Screen (which might not be the current Display Screen). The Standard Clear function (Esc E) clears the text window only.

If the GRAFIX cursor is enabled when a GRAFIX Clear Screen is executed, the cursor is removed, the screen is cleared, and the cursor is restored. Similarly, if the Text cursor had been enabled when the Standard Clear function was called, the Text cursor is removed, the text window is cleared, and then the Text cursor is restored.

The GRAFIX Clear Screen manages only the GRAFIX cursor; it does not manage the Text cursor. The Standard Clear function manages only the Text cursor, not the GRAFIX cursor. Multiple cursors will result if either of the screen clear functions affects a portion of the screen on which the other type of cursor resides. For example:

- If the Text cursor is enabled and a GRAFIX Clear Screen is called, the Text cursor will be cleared. The next time the Text cursor is moved it will be restored to both old and new positions (resulting in two Text cursors).
- If a Clear Text Window function is called while the GRAFIX cursor is within the text window, the GRAFIX cursor will disappear. The next time the GRAFIX cursor is moved it will reappear at both the old and new positions.

These problems can be avoided if you follow two simple rules:

1. Disable the Text cursor before clearing any portion of the GRAFIX screen.
2. Disable the GRAFIX cursor before clearing any portion of the text window.

8

USING WINDOWS AND SCREENS

CREATING A WINDOW 8.1

A window is a rectangular portion of the screen of any size (up to the whole screen). To create a window on the Work Screen:

1. Place the cursor at the desired location of the upper left corner of the window, and

2. Enter the extent of the window by using the Define Window function, specifying the width and height of the window in dots.

Windows can be saved on disk, loaded from disk, moved around on a screen, or moved from one screen to another.

MOVING A WINDOW 8.2

The Move Window function lets you use one of sixteen possible combination rules to move a window from screen to screen or into different places on the same screen.

For example, if you want to move the upper left quarter of Screen 3 to the lower right quarter of Screen 0, follow this sequence of calls:

1. Enable Screen 3 as the Work Screen.

2. Position cursor to: x = 0, y = 0.

3. Call the window function, with x = 399 and y = 199.

4. Move cursor to: x = 399, y = 199.

5. Specify the combination rule if it needs to be changed.

6. Call the Move Window function, specifying Screen 0.

This moves the window from the upper left corner of Screen 3 to the lower right corner of Screen 0, and changes the Work Screen from Screen 3 to Screen 0.

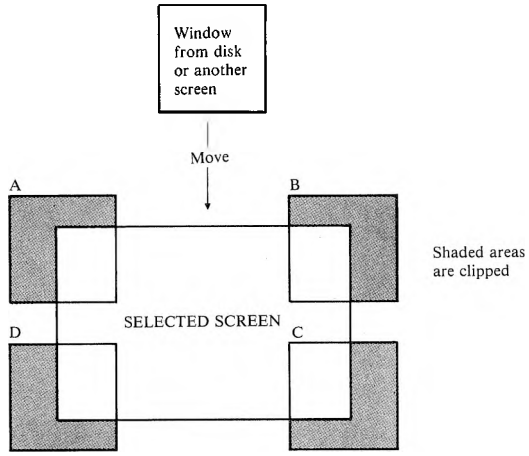
8.3 AUTOMATIC CLIPPING

Any point within the cursor positioning space ($-800 \leq x \leq 1599$ and $-400 \leq y \leq 799$) can be used for any of the drawing routines (see Figure 7-1).

Graphics are automatically clipped; the viewable screen acts as a clipping window. If you draw a line from the origin (x = 0 and y = 0) to x = 1599 and y = 799, the visible line goes diagonally across the screen, ending at x = 799 and y = 399. The cursor is at position 1599,799. If you attempt to position the cursor outside the cursor space, an error will result.

When you move a window that just been defined or loaded from disk, place the GRAFIX cursor where you want it on the destination screen. (Be sure to invoke the correct combination rule.) If you position the GRAFIX cursor outside the range of the viewable screen, the window will be "clipped." Figure 8-1 shows how a screen is clipped if the window does not fit in the viewable area of the screen.

Figure 8-1: Moving a Window



If the cursor is positioned at the point labeled A (above and to the left of the selected screen), only the portion of the window that coincides with the screen is combined with the screen. The shaded area of the window is discarded.

If the cursor is within the selected screen and the window falls entirely within that screen, all of the window is transferred. When loading a window from disk, be sure the cursor is within the viewable screen.

The Move Window function need not actually perform a move. For example, assume that you have placed the cursor at the upper left corner of a window and that you've specified the x and y extents of that window. If you invoke a Move and specify the screen on which the window has been defined, without moving the cursor, the window does not move: however, the combination rule is applied, as if there has been a move.

The portion of the screen defined as the window is combined using the set combination rule. If the rule in effect is D' = NOT D, the portion of the screen defined by the window turns into reverse video. Similarly, the portion of the screen defined by the window can be erased by invoking the appropriate combination rule (D' = 0).

8.4 SAVING TIME

You can save time in two ways when redrawing graphics figures. First, any number of graphics features can be generated just once and saved on disk. After placing the cursor at the upper left corner of your chosen region, call the Load Window From Disk function.

Second, you can build a set of graphics figures on a background (non-Display) screen, and then recall the set (move it to the Display Screen) when you need it.

Moving a window or a screen does not destroy the original image. If you move a window from Screen 3 to Screen 0, the contents of the window are still available on Screen 3.

8.5 USING SCREENS

You can save a screen, or a portion of a screen, on disk. In applications where numerous frames are to be displayed, you should use Screens 0 and 1. Follow these steps:

1. Load the first frame from disk to Screen 1 while viewing Screen 0.
2. Change the Display Screen to 1.
3. Load the next frame from disk to Screen 0.
4. Change the Display Screen to 0.

Then, the data is retrieved from disk and loaded onto the screen rapidly.

Multiple screens can be used when the graphics to be displayed on the screen are generated during the execution of the program. You can build complete screens (or partial screens called windows) on a non-Display Screen and then bring those windows onto the Display Screen.

Multiple screens also save time in applications where you need several graphics screens concurrently. You can build or bring successive frames from disk once—and then select or move them to the viewable screen as desired.

9

USING THE COMBINATION RULES

You can use the sixteen combination rules to logically combine windows with screens or other windows, and screens with other screens. The first window or screen is called the "destination" (D). The second window or screen, called the "source" (S), is the window or screen you are defining. The rules specify how the destination is transformed by the source. The transformed destination is labeled "D'".

Figure 9-1 shows a possible destination and a possible source combined under the each of the sixteen rules. Using the combination rules, you can:

1. Turn ON all the dots in the destination. (D' = 1)
2. Turn OFF all the dots in the destination. (D' = 0)
3. Invert all the dots in the destination. (D = not D)
4. Move the source into the destination. (D = S)
5. Move the inverted source into the destination. (D = NOT S)
6. Combine the source and destination or invert either one through the logical ANDing, ORing, or XORing of the two.

You need only specify the dimensions of the source. The destination automatically assumes the same dimensions. The source and the destination can be:

- On separate screens
- On different portions of the same screen
- Identical (on the same portion of the same screen)

The combination rules furnish multiple means for achieving the same end. They apply to all window and screen moves, high resolution printing, the set dot function, draw line, draw circle, and draw arc functions. All sixteen rules apply to window and screen moves and the set dot function; only three apply to the Hi-Res print function.

9.1 EXCHANGE OF SCREENS

Assume that Screen 0 is the Display Screen and that you want to view Screen 3, while preserving the contents of Screen 0. If you invoke the Move Screen function, Screen 3 will be transferred to Screen 0, but the contents of Screen 0 will be lost (the contents of both screens are now the same).

If you want to exchange the contents of the two screens without having to write one to a buffer (a 40,000-byte buffer is needed), follow these instructions (the contents of Screen 0 are labeled S0, the contents of Screen 3 are labeled S3).

1. Set the combination rule to: D' = S XOR D.

2. Set the Work Screen to 0. This places the cursor on Screen 0.

3. Move Screen to Screen 3. This XORs the contents of Screen 0 with Screen 3 and leaves the results in Screen 3. The Work Screen is now Screen 3.

4. Move Screen to Screen 0. This XORs the contents of Screen 3 with Screen 0 and leaves the results in Screen 0. The original contents of Screen 3 are now in Screen 0. The Work Screen is now Screen 0.

5. Move Screen to Screen 3. This XORs the contents of Screen 0 (S3) with Screen 3 and leaves the result in Screen 3. The original contents of Screen 0 are now in Screen 3.

BLEND OR FADE EFFECT 9.2

To fade the contents of another window or screen (called PIC3) into the viewed screen, follow these instructions:

1. Specify the screen containing PIC3 as the Work Screen.
2. Set the combination rule to: OR (D' = S OR D).
3. Move Screen (or Move Window) to the Display Screen. This ORs-in the desired graphics onto the Display Screen.
4. Set the combination rule to: D' = D.
5. Specify the screen containing PIC3 as the Work Screen.
6. Move Screen (or Move Window) to the Display Screen. This places PIC3 on the viewed screen.

HI-RES PRINTING 9.3

The combination rules that apply to Hi-Res printing are:

- Hard print the character. (D' = S)
- OR-in the character. (D' = D OR S)
- XOR-in the character. (D' = D XOR S)

If a combination rule other than one of these three is in effect when Hi-Res print is activated, the OR rule is automatically implemented as a default.

9.4 LINE DRAW FUNCTIONS

The sixteen combination rules can be summarized in the following four rules for all the Line Draw functions. In this case, the line is the source and all the dots in the source are ON.

- Hard write (or OR-in) the line. (D' = S)
- Erase a line. (D' = NOT S)
- XOR-in the line. (D' = D XOR S)
- No operation. (D' = D)

9.5 THE COMBINATION RULES

The portion of a screen (or a whole screen) to be moved is called the source and is labeled "S". The portion of the screen (or a whole screen) targeted as the location of the move is called the destination. Prior to your move, it is labeled "D'"; after the move, the destination is labeled "D'".

The following rules specify the possible transformations of the destination by the source:

0. **D' = 0:** The destination RAM contains all zeros.

1. **D' = S AND D:** The destination is formed by ANDing the corresponding bits in the original destination and the source.

2. **D' = S AND NOT D:** The destination is formed by ANDing the complemented original destination with the source.

3. **D' = S:** The source is moved to the destination.

4. **D' = NOT S AND D:** The destination is formed by ANDing the inverted source with the destination.

5. **D' = D:** No operation; no change in the destination results regardless of the source.

6. **D' = S XOR D:** The destination is formed by XORing the corresponding bits in the original destination and the source.

7. **D' = S OR D:** The destination is formed by ORing the corresponding bits in the original destination and the source. This is the starting combination rule.

8. **D' = NOT S AND NOT D:** The destination is formed by ANDing the complemented destination and the complemented source.

9. **D' = NOT S XOR D:** The destination is formed by XORing the corresponding bits in the original destination and the complemented source.

10. **D' = NOT D:** The destination is formed by complementing all the bits in the original destination.

11. **D' = S OR NOT D:** The destination is formed by complementing the original destination and ORing the corresponding bits with the source.

12. **D' = NOT S:** The complemented source is moved to the destination.

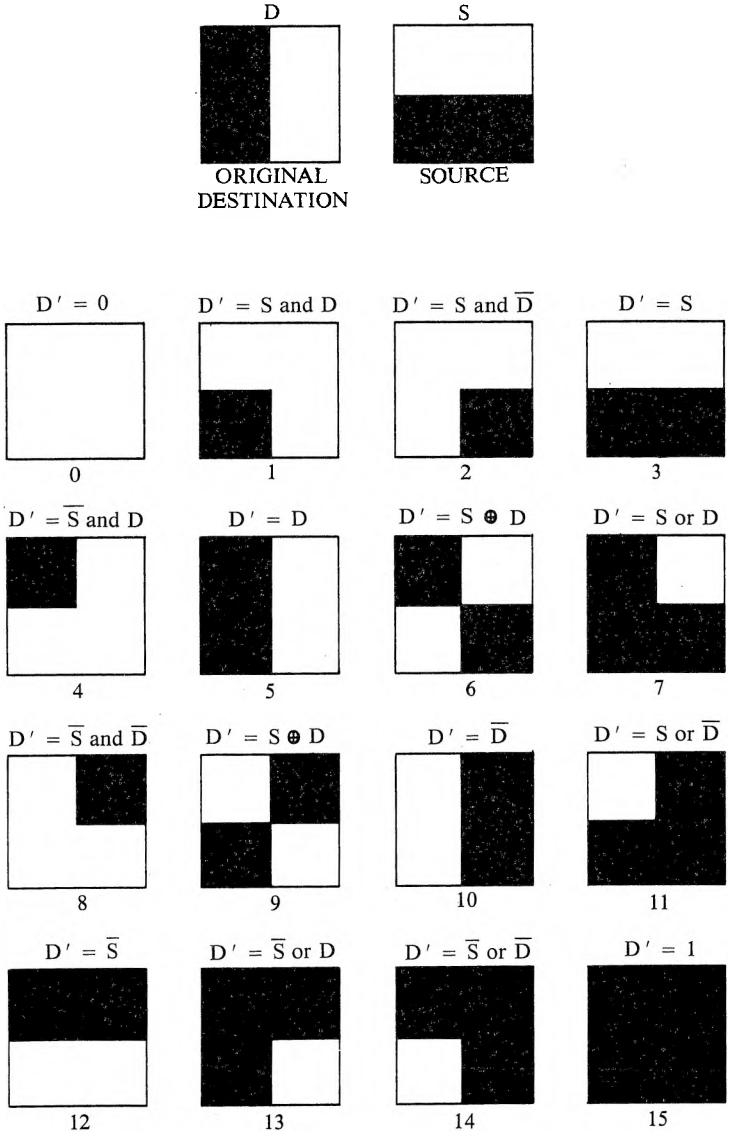
13. **D' = NOT S OR D:** The destination is formed by ORing the corresponding bits in the original destination and the complemented source.

14. **D' = NOT S OR NOT D:** The destination is formed by ORing the complemented original destination and the complemented source.

15. **D' = 1:** The destination contains all ones.

The combination rules provide flexibility in graphics presentation, including simulated motion on the screen. Figure 9-1 shows the results of applying the combination rules to a simple destination and source.

Figure 9-1: Combination Rules



For your convenience, here are some simple examples of using combination rules.

Inverting a Screen: If you select combination rule **D' = NOT D** and specify a move from Screen N back to Screen N, each dot on Screen N is inverted. Since only Screen 0 or Screen 1 can be selected as the Display Screen, the function allows Screen N (if N > 1) to be moved to the screen selected for display. This lets you display Screen N.

Erasing a Screen: When used with a combination rule, the Move function allows any screen to be erased by setting that screen to all ones or all zeros. If you set a screen to all ones and then XOR characters into that screen, a black-on-green output results. Invoking **D' = NOT D** creates the normal green-on-black display.

Scratchpad: You can select a given screen as a scratchpad where drawings, graphs, and characters are formed and then brought onto the Display Screen. Simply invoke a move using any of the combination rules.

A

ESCAPE AND CONTROL SEQUENCES

TERMINAL FUNCTIONS A.1

ESC H	Moves the cursor to the first character position on the defined top line.
ESC C	Moves the cursor one character position to the right. If the cursor is at the right end of the line, it remains there.
ESC D	Moves the cursor one character position to the left. If the cursor is at the start (left end) of a line, it remains there.
ESC B	Moves the cursor down one line without changing columns. If the cursor reaches the bottom line, it remains there and no scrolling occurs. No action is taken on bottom line + 1.
ESC A	Moves the cursor up one line without changing columns. If the cursor reaches the top line, it remains there and no scrolling occurs. No action is taken on bottom line + 1.
ESC I	Moves the cursor up one line, remaining in the same column. If the cursor is on the top line, a scroll down is performed. No action is taken on bottom line + 1.
ESC n	The operating system's Console In function reports the cursor position in the form of ESC Y line # column #.
ESC h	Moves the cursor left to next mod 8 position. Stops at left side of screen.
ESC j	The present cursor position is saved so the cursor can be returned to it after a Set Cursor To Saved Position command (ESC k).
ESC k	Returns the cursor to the position where the Save Cursor Position command was last executed. (See ESC Y.)

ESC Y l# c# Moves the cursor to the position you indicate on the screen by entering the escape code, the character representing the line number, and the character representing the column number.

Line and column numbers should be offset by 32. Thus, to move to the (1,1) position of the screen, follow ESC Y by two occurrences of the character representing the line number, "!".

If the line number entered is smaller than the defined top line, the cursor moves to the top line. If the line number entered is greater than the defined bottom line + 1, the cursor does not move from its present line. If the column number is too high, the cursor moves to the end of the line.

ESC E Erases the screen from the defined top line to the defined bottom line. If the cursor is on the bottom line + 1, the function erases bottom line + 1 only. The cursor remains in the home position. Places the cursor in the home position.

ESC b Erases from the start of screen to the cursor, including the cursor position.

ESC J Erases the screen from the cursor (including the cursor position) to the end of the defined bottom line. If on bottom line + 1, then erases to end of line only.

ESC o Erases from the beginning of the line to the cursor, including the cursor position.

ESC K Erases from the cursor (including the cursor position) to the end of the defined bottom line. If on bottom line + 1, erases to end of line only.

ESC L Inserts a new blank line by moving the line that the cursor is on (and all the following lines) down one line, to the defined bottom line. Then the cursor is moved to the beginning of the new blank line. No action is taken on bottom line + 1.

ESC M Deletes the contents of the line that the cursor is on. Then places the cursor at the beginning of the line, moves all the following lines up one line, and adds a blank line at the defined bottom line. No action is taken on bottom line + 1.

ESC N	Deletes the character at the cursor position and shifts any text located to the right of the cursor one character position to the left.
ESC Z	Responds with "ESC/K" to indicate that it can perform as VT52.
ESC @	Enters Insert Character mode, allowing insert into text on the screen. As new characters are typed in, text to the right of the cursor shifts to the right and the character at the end of the line is lost.
ESC O	Exits from the Insert Character mode.

CONTROL CODES A.2

CTRL G Bell (07H).
Generates a bell sound.

CTRL H Backspace (08H).
Moves the cursor back one column. If wrap-around mode is enabled and the cursor was at column 1, then the cursor is positioned at last column of previous row (unless at column 1, row 1, in which case the cursor is positioned at last column in row 1). If in discard mode, the cursor does not move from column 1.

CTRL I Horizontal Tab (09H).
Moves the cursor forward to the next tab stop. Tab stops are fixed at columns 9, 17, 25, 33, 41, 49, 56, 65, 73, and 81. If the cursor is at the last column, it remains there.

CTRL J Line Feed (0AH).
Moves the cursor down one line at same horizontal position, scrolls the screen up if a line feed occurs on the bottom line. If the cursor is on the bottom line + 1, then no action is taken.

CTRL M Return (0DH).
Moves cursor to the leftmost column of the same line.

CTRL X Cancel (18H).
Aborts any escape sequence in progress. Starts displaying characters as normal ASCII.

CTRL [Escape (1BH).
Starts an escape sequence.

CTRL O Shift In (0EH). (Not implemented.)
Switches the character cell size to 10 by 16, resulting in a display of 80 columns by 25 lines. The top of the screen is set to line 1 and the bottom of the screen is set to line 24. The cursor homes.

CTRL N Shift Out (0FH). (Not implemented.)
Switches the character cell size to 6 by 10, resulting in a display of 133 columns by 40 lines. The top of the screen is set to line 1 and the bottom of the screen is set to line 39. The bottom line + 1 is line 40, similar to VT52's 25th line. The cursor homes.

A.3 CONFIGURATION FUNCTIONS

ESC x Ps—Sets the following modes, where Ps equals:

1 = Enable bottom line
4 = Block cursor
5 = Cursor off
8 = Automatic line feed on receipt of CR
9 = Automatic CR on receipt of line feed
A = Send to VT52
B = Send to VT52
C = Send to VT52

ESC y Ps—Resets the following modes, where Ps equals:

1 = Disables bottom line
4 = Underscore cursor
(OK if character height >9)

5 = Cursor on

8 = No automatic line feed

9 = No automatic CR (carriage return)

A = Send to VT52

B = Send to VT52

C = Send to VT52

OPERATION MODE FUNCTIONS A.4

ESC [	Sets hold mode. (Not implemented.)
ESC /	Clears hold mode. (Not implemented.)
ESC p	Enters reverse video mode. Characters are displayed as black characters on a green background.
ESC q	Exits reverse video mode.
ESC F	VT52 graphics characters appear in character numbers 94 to 127 of the ASCII character set.
ESC G	Exits GRAFIX mode. Normal lowercase characters appear in character numbers 94 to 127.
ESC t	Enter keypad shift mode. (Not implemented.)
ESC u	Exit keypad shift mode. (Not implemented.)
ESC =	ALT. keypad on. (Not implemented.)
ESC >	ALT. keypad off. (Not implemented.)
ESC {	Inhibits the output of the keyboard. Pass through to VT52.
ESC }	Enables the keyboard after a keyboard disable command. Pass through to VT52.
ESC v	A print to the last column of the line positions to the first column of the next line. The page scrolls up if necessary.

ESC w — A print to the last column of the line does not change the cursor position and overprinting occurs. Therefore, only the last character received is displayed in the last column position.

ESC z — Reset back to 80-column mode. (Not implemented.)

ESC (— Simulates high-intensity mode by shadow printing. (Not implemented.)

ESC) — Prints normal characters. (Not implemented.)

ESC # — Will transmit only a RETURN (0DH) LINE FEED (0AH).

ESC $ — Will transmit only a RETURN (0DH) LINE FEED (0AH).

ESC] — Will transmit only a RETURN (0DH) LINE FEED (0AH).

ESC _ — Turns the debug mode on or off. In debug mode, the bottom line + 1 displays the hex codes for the print stream.

ESC 0 — Sets the underline mode.

ESC 1 — Resets the underline mode.

ESC 4 — Sets key value. Five characters are passed through to VT52 to set new key values.

ESC 8 — Displays next character literally.

ESC m2 p1p2 — Redefines the text window. p1 defines the first line of the text window; p2 defines the last line of the text window. p1 and p2 are the ASCII representations of the row numbers, as described in Figure A-1.

Figure A-1: Window Redefinition Parameters

row 0	(space)	row 9	)	row 17	1
row 1	!	row 10	*	row 18	2
row 2	"	row 11	+	row 19	3
row 3	#	row 12	,	row 20	4
row 4	$	row 13	–	row 21	5
row 5	%	row 14	.	row 22	6
row 6	&	row 15	/	row 23	7
row 7	'	row 16	0	row 24	8
row 8	(				

B

ERROR MESSAGES

If you give an invalid command, GRAFIX generates one of the following error messages:

- Invalid Command: This error occurs when you follow ESC 5 by a symbol which is not interpreted by GRAFIX.
- Parameter(s) Missing: This error occurs when you invoke an escape sequence which expects parameters to be passed, and parameters have been omitted or an insufficient number has been passed.
- Invalid Parameter: This error occurs when you execute any invalid command. For example, the error message appears if you try to move a window to Screen 4 when GRAFIX has been installed with only three screens. This error also occurs when you pass numeric parameters and alpha parameters are expected, or vice versa.
- Cursor Out of Range: This error occurs when you attempt to move the cursor out of the cursor positioning range (between −800 and 1599 in the x direction, and −400 and 799 in the y direction). This message also occurs if you call a function which expects the cursor to be in the screen range and it is not.
- File Not Found: This error occurs if you try to enable a character set that does not exist on the specified or default drive. It is also generated when a Load Window From Disk function is accessed and the file name passed does not exist.
- Disk Full: This error occurs if you execute the Save Window On Disk command and there is not enough space to save the file on the specified (or default) drive.
- Invalid File: This error occurs if the header sector does not match the format required by a Load Window or Select Character Set command. This occurs if the file has length "zero" or if the file was not generated correctly.
- Table Overflow: This error occurs if you try to use the Fill function in a polygon whose shape is too complex.

- Out of Character Sets: This error occurs if there are no character sets available to load when you do a Select Character Set. This can occur if you invoke GRAFIX with a maximum of two character sets, or if all the loadable sets have headers designating them as system character sets.

C

GRAFIX FUNCTIONS REFERENCE LIST

PARAMETER INPUT COMMANDS C.1

Reset underline mode	ESC 5 z
Save GRAFIX cursor position	ESC 5 0

C.2 PARAMETER RETURN COMMANDS

Get enabled screen number	ESC 5 j
Get displayed screen number	ESC 5 k
Get window parameters	ESC 5 e
Get dot	ESC 5 b
Get character width	ESC 5 a
Get character height	ESC 5 l
Get character type	ESC 5 o
Get GRAFIX cursor	ESC 5 u

C.3 ACTION COMMANDS

Fill a region	ESC 5 M
Fill a bar	ESC 5 N
Draw a circle	ESC 5 P
Draw an arc	ESC 5 h
Absolute line draw	ESC 5 U
Relative line draw	ESC 5 f
Move a window or a screen	ESC 5 V
Move a screen	ESC 5 W
Initialize	ESC 5 d
Hi-Res print	ESC 5 p
Clear screen	ESC 5 2
Return GRAFIX cursor to previously saved position	ESC 5 1
Print a screen	ESC 5 ?
Toggle text window screen	ESC 5 9

FILE MAINTENANCE COMMANDS C.4

Save a window on disk ..... ESC 5 S
Load a window from disk ..... ESC 5 T
Select character set (if that character
set is not already in memory) ..... ESC 5 i

INDEX

BUSIGRAF

VICTOR Publications
380 El Pueblo Road
Scotts Valley, California 95066
(408) 438-6680

Second VICTOR printing November, 1983.

ISBN 0-88182-098-9 Printed in U.S.A.

CONTENTS

APPENDIXES

FIGURES

CHAPTERS

1

INTRODUCTION

BUSIGRAF is a business graphics package. With BUSIGRAF, you can make and edit:

- Pie charts
- Bar graphs
- Line plots
- Organization charts

And, with BUSIGRAF, you can print these charts and graphs or organize them into a "slideshow."

BUSIGRAF is easy to use. No prior knowledge of computers or computer programming is necessary. The program is menu-driven—it leads you through the program automatically, so you may select both the format for a graphic, and what information that graphic will include.

This section is written in a tutorial format. You should read the entire section and Appendix A before attempting to use the program. Going through the five examples takes less than an hour and will save you time in the long run.

Each type of chart or graph is covered in a separate chapter. The type of chart is briefly described, as are its program's menus. After going through each chapter, you should be ready to create a graph on your own.

2

ENTERING DATA AND ANSWERING BUSIGRAF QUESTIONS

You use four types of data entry with BUSIGRAF:

1. Selecting from a menu
2. Answering questions
3. Entering parameters and values
4. Specifying names and titles

Each type of data entry requires a different response from you. The easiest are selecting from a menu and answering questions—you press only one key. With the other two types you enter data and then press the carriage return key, shown in the examples as (cr).

SELECTING FROM A MENU 2.1

A menu is a list of options that lets you choose which action you want to take. When a menu is presented, press the key corresponding to your choice. If you press a number that is not presented in the menu, you hear a beep; then the program asks you to re-enter your choice.

2.2 ANSWERING QUESTIONS

If you want to answer a question with yes, simply press the Y key. Press the N key to answer no. Striking any other key causes a beep.

2.3 ENTERING PARAMETERS AND VALUES

When a menu or prompt asks you to enter a parameter or value, press the key corresponding to the numeric value you want. When you finish, press (cr). If you make a mistake, use the Backspace key to correct it. **Note:** Sometimes you won't have to press (cr) after entering a value. If you are in doubt, make an entry and see if the system responds. If it doesn't, press (cr).

2.4 SPECIFYING NAMES AND TITLES

When you are asked to specify a name or title, the maximum length of the name or title is shown by a reverse video field. You can type any characters you want into that field. When you reach the end of the field, you hear a beep.

If you make a mistake, use the Backspace key to back up to the point of the mistake. The characters you backspace over are erased.

If you are editing an existing chart, you are given the old entry in the length field. If you do not want to change that entry, press (cr). Otherwise, you can edit the value using the Backspace key. **Note:** You do not need to fill in the entire field. The value you enter is centered automatically.

3

STARTING THE PROGRAM

AUTOMATIC BOOT-UP OF GRAFIX 3.1

The system supplied on your program diskette should automatically boot up the GRAFIX package with the appropriate options. If it does not, ask your dealer for assistance.

If you need to boot the GRAFIX package manually, the appropriate command is:

grafix $S1C7P*(cr)

The * specifies the printer:

F or **M**	Epson
T	Tally
C or **S**	C. Itoh
O	Okidata
N	No printer

Note: you must have all the data files for BUSIGRAF in the currently active directory on drive B.

To load the MS-BASIC Interpreter and run BUSIGRAF, type:

mbasic busigraf(cr)

BUSIGRAF MAIN MENU 3.2

The first menu that you will see when the program begins—the main BUSIGRAF menu—is shown in Figure 3-1. The choices on the menu are explained in the following chapters.

Figure 3-1: The BUSIGRAF Main Menu

BUSINESS GRAPHICS

1 .	PIE	used to make and edit PIE CHARTS
2 .	BAR	used to make and edit BAR GRAPHS
3 .	PLOT	used to make and edit LINE PLOTS
4 .	ORGANIZATION	used to make and edit ORGANIZATION CHARTS
5 .	MAKE SLIDES	used to make and edit SLIDES
6 .	SLIDESHOW	used to show a SLIDE PRESENTATION
7 .	END	exit to SYSTEM

MAKE YOUR SELECTION (1 - 7) ?

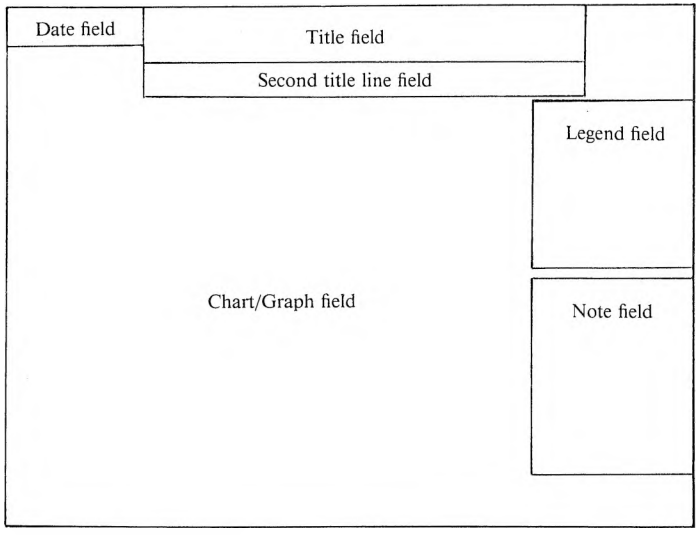
3.3 GRAPH AND CHART SCREEN ZONES

The standard BUSIGRAF chart or graph screen has six fields:

- Date
- Title
- Second title
- Legend
- Note
- Chart, Graph, or Plot

Pie charts, bar graphs, and line plots all use this standard screen. Bar graphs and line plots have vertical and horizontal axis titles and unit labels. The pies, bars, lines, and axes are automatically organized and scaled to fit into the sixth field. Refer to Figure 3-2.

Figure 3-2: Screen Zone

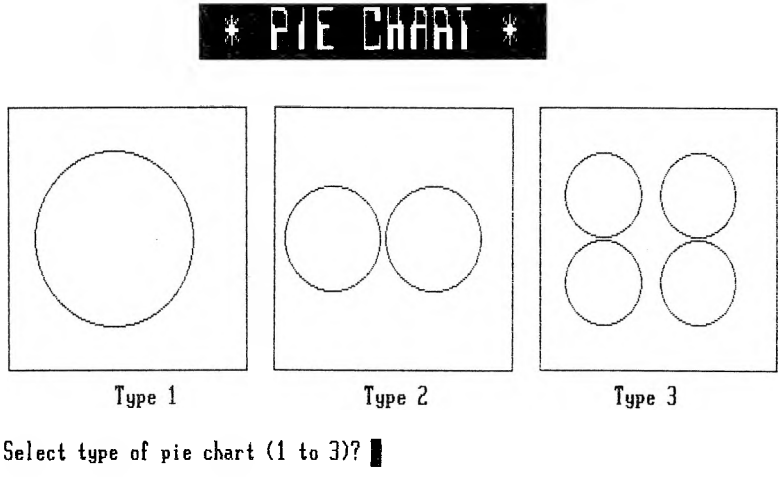


4

MAKING A PIE CHART

The Pie Chart program supports three types of pie charts, as shown in Figure 4-1.

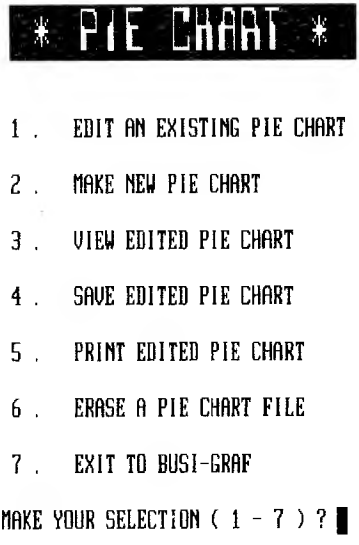
Figure 4-1: Types of Pie Charts



4.1 THE MAIN MENU

After you select the Pie Chart program at the BUSIGRAF main menu, the following menu appears on your screen.

Figure 4-2: Pie Chart Main Menu



These options are available to you from the Pie Chart main menu:

- Edit an Existing Pie Chart—displays a directory of the current pie chart files on the (data) disk in drive B, and asks you which one you want to edit.
- Make a New Pie Chart—makes a new pie chart from scratch. If you choose this selection, you are led through a data entry procedure until you form a complete pie chart.

- View Edited Pie Chart—displays the pie chart in memory. To view a chart, you must first have chosen one of the two options for loading a chart into memory.
- Save Edited Pie Chart—saves a just-created or just-edited pie chart. The pie chart will be saved on the diskette in drive B.
- Print Edited Pie Chart—used to print a hard copy of the pie chart in memory. To print a chart, you must first have chosen one of the first two options.
- Erase a Pie Chart File—deletes a particular pie chart file from the data diskette.
- Exit to BUSIGRAF—returns to the main BUSIGRAF menu.

MAKING A NEW PIE CHART 4.2

To make a new pie chart, you press "1" at the main menu. Then follow these steps (your entries are shown in boldface):

1. Enter the date. You are allowed 12 characters.

   For this example, enter **July 8, 1982.**

2. Enter the title of the pie chart. You are allowed two lines for the title. The first line of the title can be 24 characters long, the second can be 50 characters long. (Figure 4-2 shows where the title appears on the chart.)

   Enter **The Decline** for the first line, and **of the American Widget Industry** for the second line.

3. If you want notes, press the Y key; otherwise, press "N". Next, enter the number of notes (you are allowed five), and then enter the notes.

   There are two notes in the example. The first one is **1979- Western Widgets enters agreement with Renault Tools**. The second one is **1980- US Government breaks up International Widget Corp.**

4. You are now asked to choose the type of pie chart you want. (See Figure 4-1 for the types available.)

   Choose Type 3. You will chart four different years.

5. Then enter the number of slices you want in each pie. Choose a number between 2 and 8.

   Choose **4** slices. You'll want one for the USA, one for Japan, one for Germany, and one for Other Countries.

6. For each pie slice, enter the name of the slice, and the corresponding value for each pie.

   Enter **USA** for the name of the first parameter. Then enter the following USA values for each pie: **76, 65, 54, 42**. For the other slices, enter: **Japan** and **8, 12, 19, 36**; **Germany** and **12, 14, 13, 12**; **Other Countries** and **8, 9, 14, 10**.

7. If you want to select the shading patterns, press "Y"; otherwise press "N".

   Press **Y**.

8. If you struck the Y key to select the patterns, enter the patterns you want for each slice.

   In the example, enter **8** for the USA, **5** for Japan, **4** for Germany, and **2** for Other Countries.

9. If you want any pie slices exploded (pulled out slightly from the rest of the pie), press "Y"; otherwise, press "N".

   Press **Y**.

10. If you struck the Y key to explode a pie slice, enter a number between 0 and 4 to indicate how many slices you want exploded.

    Enter **1**.

11. Then, enter the parameter number of the slice you want exploded.

    In the example, choose **1** for USA.

12. Then enter the subtitle for each pie in the chart. The subtitles appear under each pie in your chart. Only 16 characters are allowed.

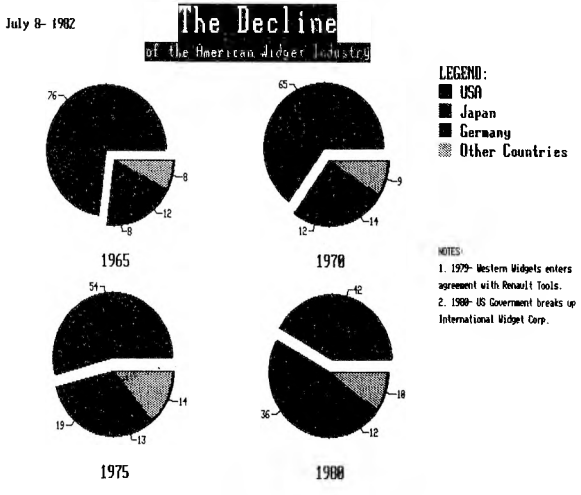
    In the example, enter **1965** for Pie 1, and **1970**, **1975**, and **1980**, respectively, for the other pies.

13. Indicate whether you want the percentages calculated for you. If you don't have them calculated, BUSIGRAF simply displays the values (slices) in relation to the other values in the pie.

    Press the N key. (If you had known only the dollar sales of each country, instead of the percentage shares of the market, you would have struck "Y" to let BUSIGRAF determine the percentages.)

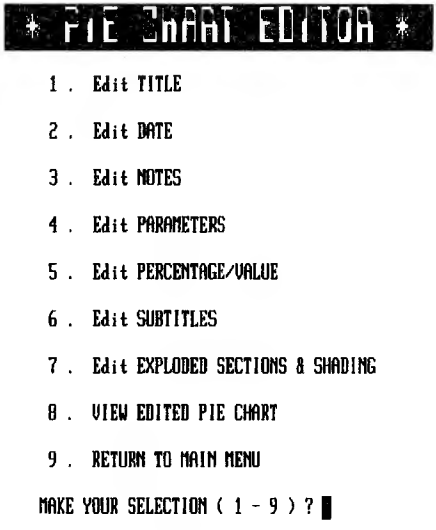
14. The pie chart is displayed on the screen. It should look like the chart in Figure 4-3. Note that the comma in the date was changed to a hyphen. Commas are not allowed in any entries, and are always converted to hyphens.

Figure 4-3: Example Pie Chart



15. Press the space bar to continue. The Pie Chart Editor menu will be displayed (see Figure 4-4). If there are any mistakes in the displayed chart, select the option number for the incorrect data and follow the directions to fix the error. If there are no errors, select option 9 to return to the main Pie Chart menu. If you have trouble correcting the errors, go on to the next chapter, and return to this one later.

Figure 4-4: Pie Chart Editor



16. When you are returned to the main Pie Chart menu, you can save the chart, print the chart, edit a chart, make a new chart, erase a chart, or view a chart.

    Note: If you want to keep the chart you just made, be sure to save it. Otherwise you'll have to re-enter all the data.

There is an example of a type 1 Pie Chart in Appendix A.

5

MAKING A BAR GRAPH

The Bar Graph program allows you to make six types of bar graphs. These are listed in Figure 5-1.

Figure 5-1: Types of Bar Graphs

Make a selection of the type of graph. Choices 1 through 5 handle the labelling of the horizontal axis automatically, choice 6 allows you to enter the desired labels. The choices are:

1 . To plot a single value for each year

2 . To plot a single value for each month

3 . To plot a single value for each week

4 . To plot a single value for each day (all seven days)

5 . To plot a single value for each day (Mon to Fri only)

6 . To label each value individually

MAKE YOUR SELECTION (1 - 6) ?

THE MAIN MENU 5.1

The Bar Graph main menu is nearly identical to the Pie Chart program's main menu. All selections function like the similar selection in the Pie Chart program.

5.2 BAR GRAPH EXAMPLE

Select option 2 at the main menu. Then, use the same data you used in the pie chart, but represent it in a bar graph. Your entries are shown in boldface:

1. Enter the date: **8 July 1982.**
2. Enter the first line of the title: **The Decline**. Note that you are allowed only 20 characters, instead of 24 as in the pie chart.

   Enter the second line of the title: **of the American Widget Industry**. Note that you are allowed only 40 characters, instead of 50.
3. There will be two notes. Note 1 is **1979–Western Widgets enters agreement with Renault Tools**. Note 2 is **1980–US Government breaks up International Widget Corp.**
4. Enter the number of bars: **4**.
5. Enter the names of the bars: **USA**, **Japan**, **Germany**, and **Other Countries**.
6. Enter the vertical axis title: **% of US Sales**.
7. Since you are charting sales percentages rather than actual sales, don't enter anything for Vertical Axis Units. Instead, press (cr). (**Note:** You could have used this field to continue the vertical axis title.)
8. Choose the type of bar graph. You want selection 6 (to label each value individually).
9. Enter **Year** for the horizontal axis label.
10. You don't want to list anything for units on the horizontal axis, so press (cr).
11. You want to plot four years, so enter **4** for the number of data points per parameter (bar) to be plotted.
12. Since the horizontal axis label is not numeric (the values are for years, not numbers), press "N".

(If you want to use numeric axis labels, press "Y", and enter the starting value, followed by the step value. The step value is the difference between the starting value and the next value marked on the axis. If you enter 1000 as the starting value and 5 as the step value, BUSIGRAF automatically calculates the rest of the data point labels for you: 1005, 1010, 1015 and so on.)

13. Now enter the numeric value that each parameter will have at a particular year (data point). For parameter 1 (USA), the value at data point 1 is **76**. The next question requests the label for data point 1. Enter **1965**. Then enter **65** followed by **1970**, **54** followed by **1975**, and **42** followed by **1980**. Since you've already entered the labels for the data points, you won't have to enter them again for the other parameters. For Japan, enter **8**, **12**, **19**, **36**. For Germany, enter **12**, **14**, **0**, **12**. (The zero in the third year is an intentional mistake. It will be fixed later.) For Other Countries, enter **8**, **9**, **14**, **10**.

14. Press "N" to let the program automatically select the shading patterns.

15. Look at the chart. You want to: (1) change the shading pattern of Japan to a brighter pattern; and (2) fix the error you made in step 13. Press the space bar to continue. The Bar Graph Editor menu will be displayed.

16. Press "5" to change the shading patterns and "Y" to indicate that you want to select the patterns.

17. Select pattern 8 for the USA, 4 for Japan, 2 for Germany, and 5 for Other Countries.

18. Press "4" to edit the parameters. The parameter names are correct, so press the N key. You want to change a parameter value, so press "Y". You made a mistake when you entered the data point for Germany, however, so press "3" to indicate parameter Germany. Then press "Y" to indicate that you want to make a change.

19. Press "3" to indicate you want to change Germany's third data point. Enter the correct value, **13**. That is the only change you want to make, so press the N key.

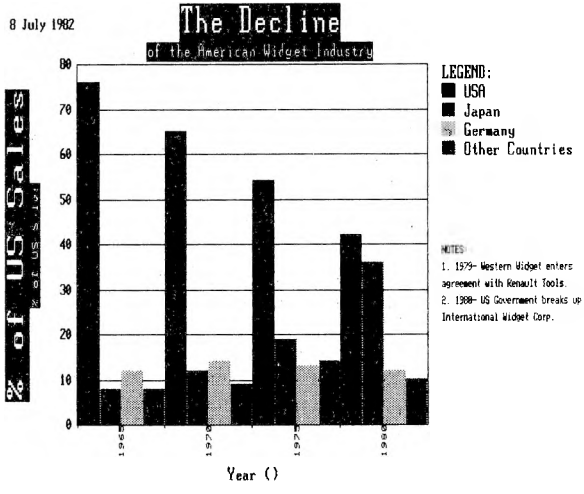
20. Press the N key again to return to the Bar Graph Editor menu. Press "8" to view the chart. Your bar graph should look like the one in Figure 5-2.

21. Return to the main Bar Graph menu to save the table.

If you want to make one of the other five types of bar graphs, follow the line plot example—the types of line plots correspond to the types of bar graphs.

Appendix A includes an example of a type 2 Bar Graph.

Figure 5-2: Example Bar Graph



6

MAKING A LINE PLOT

Line plots are similar to bar graphs, except the data points are connected by lines instead of represented by bars. The following example is almost identical to the previous bar graph example. The data, however, will be displayed as a type 1 Line Plot (corresponding to a type 1 Bar Graph).

THE MAIN MENU 6.1

The Line Plot main menu is similar to the previous main menus.

LINE PLOT EXAMPLE 6.2

After you press "2" at the main menu:

1. Enter the date: **8 July 1982**.
2. Enter the title, **The Decline**, then press "Y" to indicate a second line. Enter the second line of the title: **of the American Widget Industry**.
3. There will be two notes: **1979 -- Western Widgets enters agreement with Renault Tools**; and **1980 -- US Government breaks up International Widget Corp.**
4. Enter the number of lines to plot: **4**.
5. Enter the names of the lines: **USA, Japan, Germany,** and **Other Countries.**

6. Enter the vertical axis title: **US Sales**.

7. You will now plot actual sales, not percentages. Enter **Thousands** for the unit.

8. Choose the type of line plot. You want selection 1 (to plot a single value for each year).

9. Enter **1970** as the starting year.

10. Enter **10** as the number of years.

11. Enter the values for each year for each country. For the USA, enter **618, 623, 632, 630, 628, 623, 616, 602, 585**, and **555**. For Japan, enter **145, 151, 192, 212, 239, 255, 280, 301, 345**, and **406**. For Germany, enter **112, 114, 115, 116, 113, 110, 115, 119, 121,** and **118**. For Other Countries, enter **70, 72, 76, 74, 75, 79, 79, 77, 78,** and **80**.

    Note: Because of the line type you requested, the next nine years (after the first year 1970) are 1971 through 1979. Years are incremented by one automatically.

12. Press "N" to let the program automatically select the line types.

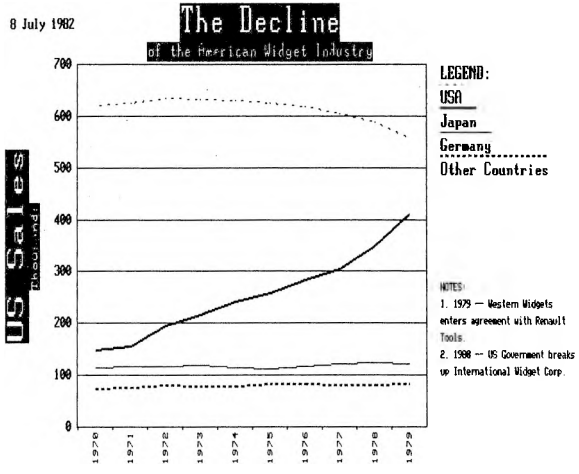
13. Look at the chart. You want to change the line types so that the lines for Japan and Other Countries are more distinct. Therefore, hit the space bar to continue on to the Line Plot Editor.

14. Press the 5 key and "Y". Choose types that will make the lines more distinct. For example, choose 3 for USA, 5 for Japan, 1 for Germany, and 6 for Other Countries.

15. Press the 8 key to view the plot again. Your line plot should look like the graph in Figure 6-1.

There is a sample of another line plot in Appendix A.

Figure 6-1: Example Line Plot



6

7

MAKING ORGANIZATION CHARTS

The Organization Chart program allows you to make a simple organization chart: a main block with a tree of single-level secondary blocks (as few as two or as many as eleven).

Although the format for the organization chart may seem restrictive since only a single level of secondary blocks is supported, you can create multiple organization charts for more complex organizations.

UNDERSTANDING ORGANIZATION CHARTS 7.1

You should understand the concept of organization charts before proceeding further. Look at the group of sample organization charts in Appendix A. Notice how each of the three charts is related. Figure A-4 is the main chart; it is the only level 1 chart. Level 2 (Figure A-5) continues the main chart by further describing one of the nodes of the main chart, "VP Marketing." One of the nodes of the secondary chart, "International Marketing Manager," is further described in the third chart (Figure A-6). The main menu's functions are identical to those of the other main menus.

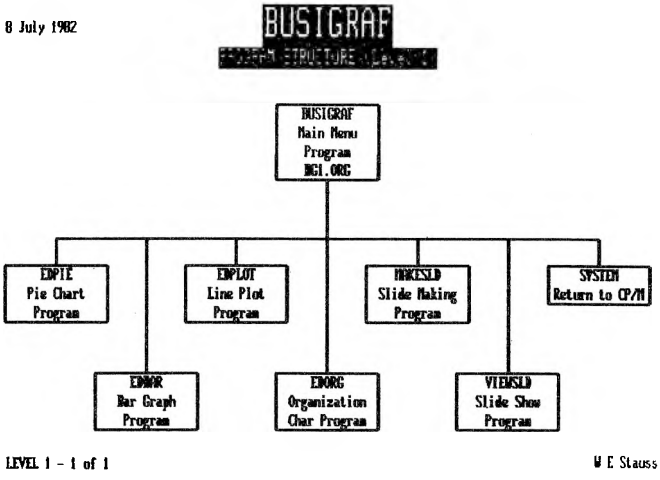
7.2 ORGANIZATION CHART EXAMPLE

The following example shows how to produce the organization chart in Figure 7-1.

After pressing "2" at the main menu:

1. Enter the date: **8 July 1982**.
2. Enter the organization chart title: **BUSIGRAF**.
3. Enter the second title line: **PROGRAM STRUCTURE (Level 1)**.
4. Enter the number of lines of text for the main block: **4**.
5. Enter the first line: **BUSIGRAF**. Enter the second, third, and fourth lines: **Main Menu**; **Program**; and **BG1.ORG**.
6. Enter the number of secondary blocks: 7.
7. Secondary block 1 will have three text lines, so press the 3 key and enter the three lines of text. Repeat this process for each of the remaining six blocks displayed in Figure 7-1.
8. Enter the control code: **LEVEL 1 - 1 of 1**.
9. Enter your name as the operator identification.
10. The organization chart appears on the screen. Check to see that it is identical to the chart in Figure 7-1. The only difference should be the name in the lower right corner.
11. Press the space bar to continue on to the Organization Chart Editor. If there are mistakes, press the appropriate key, and correct the mistakes as before. If there are none, press "8" to return to the main menu. Save the chart.

Figure 7-1: Example Organization Chart



8

MAKING SLIDES AND SLIDESHOWS

You can organize a group of pie charts, bar graphs, line plots, or organization charts into a more understandable presentation which automatically brings charts and graphs to the screen. To do this, you must make a "slide" out of each chart or graph and then enter the order you want the slides to appear in your "slideshow."

UNDERSTANDING THE SLIDESHOW 8.1

A slideshow consists of a series of graphs or charts that is displayed automatically. To make one, you first decide which slides should appear, and in which order, and then run the Slide Maker program as described in Chapter 8.2.

A slideshow works as follows. The screen is replaced by the first slide in the slideshow (any type of chart or graph). When finished looking at the first slide, the viewer presses the space bar. The slide disappears and is replaced by the next slide, another chart or graph. This continues until the viewer has looked at each slide in the slideshow.

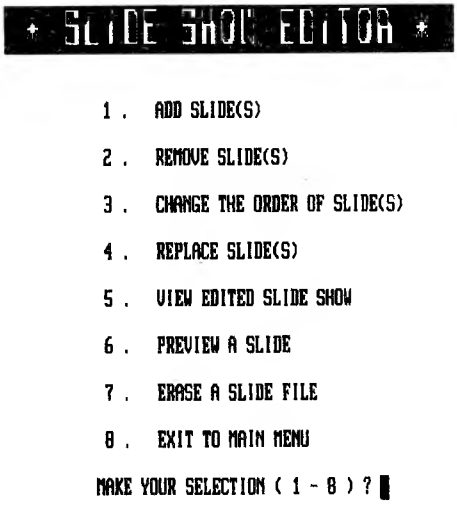
The slideshow is useful for demonstrations and for presentations of chartable data. You can design or develop a slideshow, and then save the show on disk for viewing at any time.

8.2 SLIDESHOW EXAMPLE

The following example illustrates how to make a slideshow of the charts and graphs you have created in the previous examples. (If you didn't save the graphs, you'll need to create some new ones.) At the main Slide Maker menu:

1. Press the 1 key.
2. Choose the pie chart you want to convert into a slide and enter its name. Be sure to enter the name exactly as you assigned it, with upper- and lowercase letters identical to those in the original chart name.
3. Then choose a name for the slide. Try **SLIDE1**.
4. Since no other pie charts should be converted to slides, press "N".
5. Press "2" to create a slide from a bar graph.
6. Repeat steps 2 through 5, but choose a bar graph, and name the slide **SLIDE2**.
7. Press "3" to create a slide from a line plot.
8. Repeat steps 2 through 5, but choose a line plot, and name the slide **SLIDE3**.
9. Press "4" to create a slide from an organization chart.
10. Repeat steps 2 through 5, but choose an organization chart, and name the slide **SLIDE4**.
11. Now put together the slideshow. Press the 5 key.
12. Give the show a name.
13. Enter the names of the slides that correspond to the slide numbers in the show. For example, enter **SLIDE1** as Slide 1.
14. Press "Y" to continue, and enter the names of the other three slides: **SLIDE2**, **SLIDE3**, and **SLIDE4**.
15. When all the slide names have been entered, press "N". The following menu appears.

Figure 8-1: Slide Show Editor



16. Press "5" to view the slideshow. When you are finished viewing a slide, press the space bar to continue on to the next slide in the show.

17. When you are finished with the last slide, you are returned to the main Slide Maker menu. If there are mistakes in the show, press the 6 key to go to the Slide Show Editor.

    For example, if you want to move the organization chart from the last slide to the first slide, follow these steps:

    a. Press the 3 key.

    b. Enter **4** to move the organization chart.

    c. Enter **1** to move the slide to the first position.

    d. Press the N key to return to the Slide Show Editor.

    e. Press the 5 key to view the reorganized slideshow.

After viewing the show, you will be returned to the main Slide Maker menu. The slideshow is saved automatically.

Note: If you change any of the charts that have appeared in a slideshow, you must adjust the slideshow by deleting the old chart and replacing it with the revised chart. This is not done automatically.

8.3 VIEWING A SLIDESHOW

If you want to show a slide presentation, press "6" when you're at the main BUSIGRAF menu. Select the name of the show you'd like to view, and enter its name. Be sure to observe the case of the letters.

A

SAMPLE CHARTS AND GRAPHS

Figure A-1: Sample Pie Chart

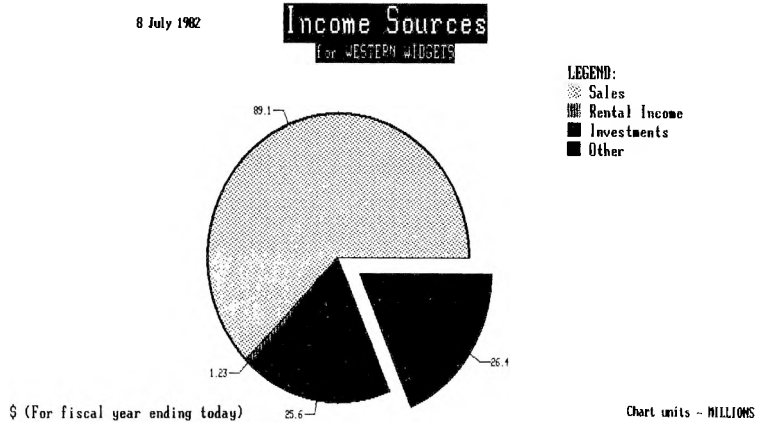
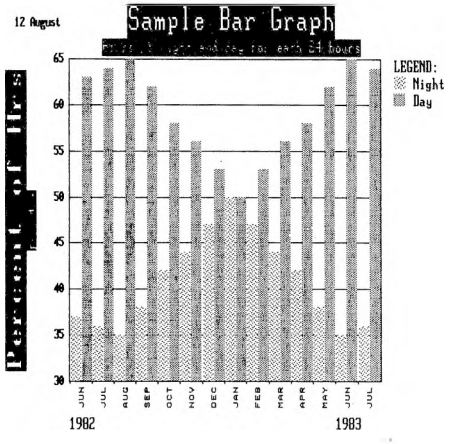


Figure A-2: Sample Bar Graph



A

Figure A-3: Sample Line Plot

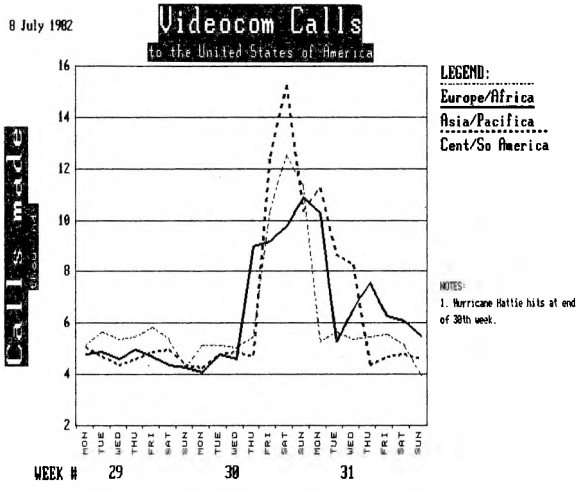


Figure A-4: Sample Organization Chart A

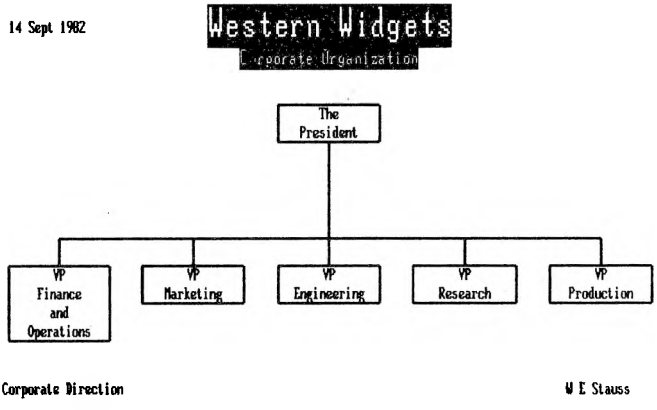


Figure A-5: Sample Organization Chart B

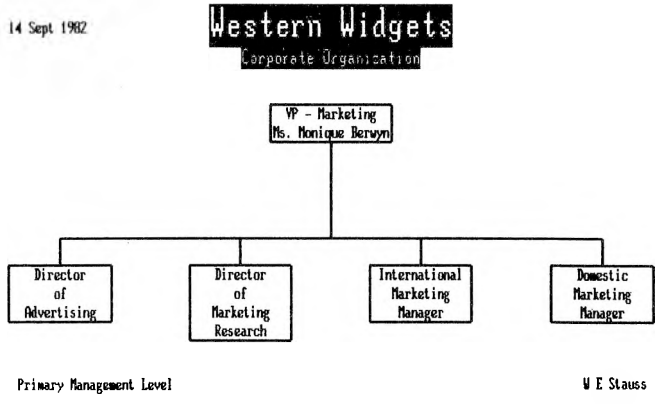
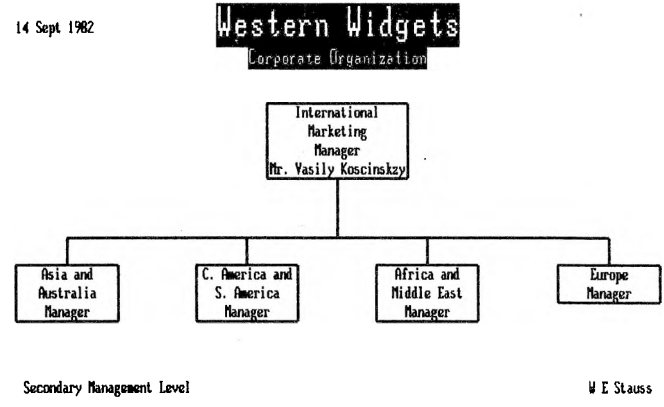


Figure A-6: Sample Organization Chart C



A

B

DATA FILE MANIPULATION

CHANGING THE DATA DISKETTE B.1

If you want to change the data diskette, you can stop the program with an ALT-C. (**Note:** MS-BASIC does not always accept ALT-C, so you may need to enter more than one to stop a program.) The following message appears on the screen:

```
BREAK IN LINE xxxx
Ok
```

(where xxxx is the line number.) Now another data diskette can be inserted into drive B. To log the new diskette, type:

reset(cr)

To restart the program, type:

run(cr)

DIRECTORY OF THE DATA DISKETTE B.2

You can examine the contents of the data diskette partially by using the BUSIGRAF EDPIE subprogram. Ask to edit an existing file to see all the available .PIE files. To see the contents of the whole data diskette, stop the program with an ALT-C and then type:

files "b:*.*"(cr)

B.3 DATA FILE TYPES

Source files for editing are:

.BAR for Bar Charts
.PIE for Pie Charts
.PLT for Line Plots
.ORG for Organization Charts
.CTL for Slideshows

All of these files are created by BUSIGRAF and can be edited.

Compiled files are:

.SLD

You can group compiled files for a slideshow, but you cannot edit them. They are generated from .PIE, .BAR, .PLT, and .ORG files in the MAKESLD program. You can view them from the operating system, assuming GRAFIX has been installed, by entering:

```
type b:filename.sld(cr)
```

where FILENAME is the name of the file you want to view.

Show files are:

.CTL

These are "control" files which contain the names of .SLD files. You can edit the contents of .CTL files with the MAKESLD program.

EDITING DATA FILES B.4

There are two ways to edit BUSIGRAF files:

1. The safe method: Use the BUSIGRAF internal editors.
2. The dangerous method: Use a word processor or text editor. In this case, you must ensure that: first, the word processor does not insert control characters (use non-document or program mode); and second, you are familiar with the file format for the particular file.

RENAMING DATA FILES B.5

You cannot directly rename files in the BUSIGRAF program. But if you leave the program, there are three ways to rename files:

1. Halt the program with an ALT-C. This re-enters the MS-BASIC Interpreter and allows use of MS-BASIC's full utilities to rename files. To rename a file, type:

   name "b:oldname.ext" as "b:newname.ext"(cr)

   where OLDNAME.EXT is the current name of the file and NEWNAME.EXT is the new name for the file.

2. Return to the main BUSIGRAF menu from any of the other menus and select **7. END exit to system.** This returns control to the operating system and the A> prompt appears. You can now use system utilities to rename files. To rename, type:

   ren b:newname.ext = b:oldname.ext(cr)

   where NEWNAME.EXT is the new name of the file and OLDNAME.EXT is the old name of the file.

3. In the appropriate BUSIGRAF subprogram (PIE, BAR, LINE PLOT, ORGANIZATION CHART, or SLIDESHOW), read the current file and save it under the new name. Then, delete the old file.

INDEX

CHARGRAF

Second VICTOR printing November, 1983.

ISBN 0-88182-100-4 Printed in U.S.A.

CONTENTS

FIGURES

CHAPTERS

1

CHARACTER GRAPHICS SYSTEM (CHARGRAF)

The character graphics system (CHARGRAF) allows you to directly access the graphics characters residing between A1 hex and FE hex in the International character set. CHARGRAF allows you to bypass the usual requirement to use the CHR$ type function from within a program or the ESC/8/control-character sequence from the keyboard. Instead, you can access the graphics characters with a single keystroke using a properly configured operating system with the graphics character set provided with CHARGRAF, or a character set you create. A facility is also provided that allows a text file containing images developed through CHARGRAF to be output to any printer.

SYSTEM STARTUP 1.1

Since it is defined in the system configuration, the character graphics system is initialized during a system cold boot. Simply insert the properly configured system diskette into drive A. You can confirm that the character graphics system has been properly loaded by checking the operating system sign-on banner: "Graphic" should appear in the keyboard description.

1.2 EXAMPLE PROGRAMS

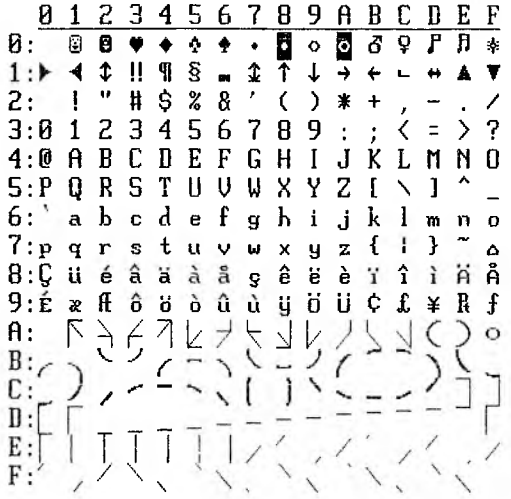
Three example programs (and their corresponding sources in BASIC and Pascal) are contained on the distribution diskette:

- BAR.EXE (executable)
  BAR.BAS (BASIC)
  BAR.PAS (Pascal)
- MAN.EXE (executable)
  MAN.BAS (BASIC)
  MAN.PAS (Pascal)
- JUGGLER.EXE (executable)
  JUGGLER.PAS (Pascal)
  This is a slightly more advanced example demonstrating character graphics animation.

1.2.1 LOW RESOLUTION GRAPHICS CHARACTER SET

The low resolution character set is comprised of single characters that are defined in a 16-by-10-dot matrix cell. The graphics characters provided with CHARGRAF are made up of circle and line characters that allow you to create fairly complex images using the graphics character set shown in Figure 1-1.

Figure 1-1: CHARGRAF Character Set



The CHARGRAF character set includes the following features:

- 13 horizontal lines located at cell matrix dots 0, 1, 2, 3, 4, 6, 7, 9, 10, 12, 13, 14 and 15.
- 8 vertical lines located at cell matrix dots 0, 1, 2, 4, 5, 7, 8 and 9.
- 45 degree and −45 degree lines.
- Special characters used to develop three-dimensional images such as bar graphs, boxes and so on.
- Circle development characters with six different radii: 1 × 1 cell, 1 × 2 cell, 2 × 2 cell, 2 × 3 cell, 2 × 4 cell and 4 × 7 cell.

These features allow the development of graphics such as organization charts, graphs and plots, pie charts and bar graphs.

1.2.2 OPERATION OF THE CHARACTER GRAPHICS SYSTEM

Operating CHARGRAF is as easy as typing on the keyboard. The graphics characters have been attached to selected keys on the keyboard, and can be accessed by typing keys in unshifted, shifted, or Alternate mode. (Each of the selected keys has the ability to display a maximum of three graphics characters.) The keys that can access graphics characters are the seven function keys, the calculator pad keys, and certain other keys.

The logical key number and their associated graphics character hex codes in unshifted, shifted and Alternate modes are listed in Figure 1-2. The logical key numbers corresponding to the keys on the keyboard are shown in Figure 1-3.

Figure 1-2: Keyboard Logical Key Numbers

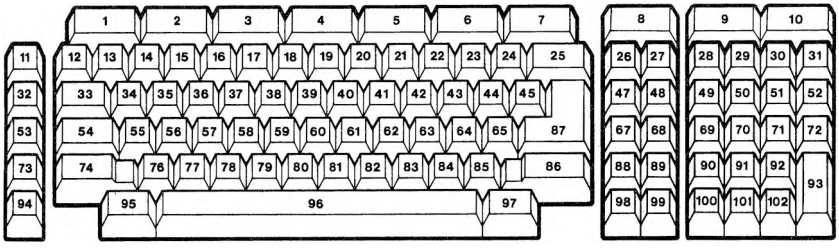


Figure 1-3: CHARGRAF Keyboard-Character Table

Logical Key	Unshifted Hex Code	Unshifted Char.	Shifted Hex Code	Shifted Char.	Alternate Hex Code	Alternate Char.
1	F1	/	F8	\	EE	
2	F2	/	F9	\	ED	/
3	F3	\	FA		EC	/
4	F4	\	FB	\	EB	
5	F5	\	FC	\	EA	/
6	F6	\	FD	\	E9	/
7	F7		FE	\	E8	/
11	-		F0	/	E7	/
12	F1	/	EF	/	E6	|
16	-		-		E5	|
18	-		-		E4	|
25	-		E2	|	E3	|
26	E0	|	E1	|	CLR SCREEN	
27	-		DE	~	DF	|
28	-		DC	~	DD	~
29	-		BA	~	DB	~
30	D7	_	D8	~	D9	~
31	-		D5	_	D6	~
32	D2	_	D3	_	D4	_
33	-		D0	└	D1	┌
44	CD	/	CE	┐	CF	┘
45	-		-		CC	/
47	C9	\	CA	\	CB	_

Logical Key	Unshifted Hex Code	Unshifted Char.	Shifted Hex Code	Shifted Char.	Alternate Hex Code	Alternate Char.
48	C6	\	C7	|	C8	)
49	-		C4	-	C5	`
50	-		C2	,	C3	'
51	-		C0	_	C1	(
52	-		BE	\	BF	_
53	-		-		BD	\
64	-		-		BC	~
65	-		-		BB	'
67	B9	(	BA	'	-	
68	B7	\	B8	_	-	
69	-		B5	~	B6	\
70	-		B3	(	B4	'
71	-		B1	\	B2	\
72	-		AF	o	B0	'
73	-		-		AE	)
83	-		-		AD	-
84	-		-		AC	|
85	-		-		AB	\
98	-		A9	|	AA	(
91	-		A7	|	A8	|
92	-		A5	|	A6	|
93	-		A3	|	A4	|
100	-		A1	|	A2	|

2

THE CHRPRINT FACILITY

CHRPRINT is a stand-alone program run at the system level that outputs a text file containing character graphics images to a dot-matrix printer. To use CHRPRINT, first be certain of the following:

- The text file you want to print is saved on diskette.
- A printer is properly connected to your computer and the correct I/O channel is set.
- **Important:** The operating system's graphics character set is the same as the graphics character set under which the file was created.

You can then invoke CHRPRINT with the command:

CHRPRINT pathname

The 25th line of the screen will be replaced by a menu of printer choices:

```
EPSON MX   EPSON FX   TALLY   C.ITOH   C.ITOH S   OKIDATA
```

When you press the function key corresponding to the printer of your choice, CHRPRINT prints the specified text file using the operating system graphics character set.

3

CUSTOM CHARACTER GRAPHICS SYSTEMS

DEFINING YOUR OWN CHARACTER SETS 3.1

You can create your own character set to suit any special need by using a character set editor, like EFONT. You must follow certain steps, however, to configure an operating system correctly with your customized character graphics.

The character set header (a small section of information that is written out to disk in front of any character set) contains important information used by the operating system configuration program. This information must be correct for your particular character set, or the character set won't work. Furthermore, the number of characters defined can affect software you may want to run under the new operating system. You can redefine all 256 characters of a character set (normally done only in unusual circumstances), but you should stay within the range of Al hex to FE hex.

3.2 CONFIGURING YOUR KEYBOARD

Besides creating your own character set, you can also place your character set into any position on the keyboard that you like. Using KEYGEN, you can specify the hex codes each key generates in unshifted, shifted or Alternate mode. (Figure 1-2 shows how the keyboard is configured on the distribution diskette.) When configuring a keyboard, you should consider the software that will be run under the new configuration. Some programs reserve certain keys for their own use; take steps to supply the program with the required keys if you are using that program in the new configuration.

3.3 PUTTING IT ALL TOGETHER

Once you have defined your own special graphics character set and keyboard, you can configure a functional operating system. Use the system configuration program (SYSGEN) or MODCON to do this (refer to the *Programmer's Tool Kit*). These programs allow you to combine everything you need to create your own character graphics operating system.

INDEX

GW-BASIC

VICTOR Publications
380 El Pueblo Road
Scotts Valley, CA 95066 USA
(408) 438-6680

First VICTOR printing February, 1983.
Second VICTOR printing December, 1983.

ISBN 0-88182-095-4 Printed in U.S.A.

CONTENTS

PREFACE

GW-BASIC was created to take advantage of the facilities offered by the newer 16-bit microprocessors. The extra capabilities provide:

- Advanced graphics
- Sound
- Device-independent I/O and telecommunications support
- Event trapping

GW-BASIC is an extension of MS-BASIC. Likewise, this manual is an extension of the MS-BASIC manual. *GW-BASIC* includes only those features and statements that are specific to this program, or that have different uses than they have in MS-BASIC.

Chapter 1 introduces the special features that are supported by GW-BASIC. It discusses graphics capabilities and event trapping. These features are further discussed in the appropriate statement sections. Chapter 1 also covers device-independent I/O and files.

Chapter 2 explains how to use the full screen editor to input and edit your programs. The editor provides immediate visual feedback and functions such as cursor movement, insertion, and deletion.

Chapters 3 and 4 cover GW-BASIC statements and functions. These chapters are organized with this outline:

FORMAT:

Shows the correct format for the instruction.

PURPOSE:

Tells what the instruction is used for.

REMARKS:

Describes in detail how the instruction is used.

EXAMPLE:

Shows sample programs or program segments that demonstrate the use of the instruction.

Statement and function syntax follows these rules:

1. Items in uppercase letters must be input as shown.
2. Items in lowercase letters enclosed in angle brackets (< >) are to be supplied by the user.
3. Items in square brackets ([ ]) are optional.
4. All punctuation except angle brackets and square brackets (i.e., commas, parentheses, semicolons, hyphens, equals signs) must be included where shown.
5. Items followed by an ellipsis (...) can be repeated any number of times (up to the length of the line).

Chapter 5 describes the communications option, including Communication I/O, the TTY program, and COM I/O functions.

Chapter 6 describes initialization and printer configuration.

GW-BASIC has two appendixes: Appendix A describes error messages; Appendix B is a complete summary of statements and functions available in GW-BASIC.

1

GW-BASIC FEATURES

This chapter describes the special features that are part of GW-BASIC. It includes graphics, event trapping, and device-independent I/O.

GRAPHICS 1.1

GW-BASIC can create high-resolution graphics on the standard screen. The program supports two screen modes:

- SCREEN 0: a text-only video display mode nearly identical to the MS-BASIC display mode.
- SCREEN 2: a high-resolution mode for black and white graphics and text.

The screen mode is chosen by the mode parameter in the SCREEN statement.

Table 1-1 further describes the screen modes.

Table 1-1: Screen Modes

		CHARACTER		GRAPHICS (PIXELS)		COLOR/
	MODE	COLS	ROWS	HORIZ	VERT	B&W
Standard	0	40/80	25	No Graphics		B&W
	2	80	25	800	400	B&W

The screen in mode 2 produces the highest resolution and the sharpest images for both graphics and text. Mode 2 is, however, black and white only.

The GW-BASIC statements used to draw and manipulate images are:

PSET	CIRCLE	PAINT
PRESET	GET	DRAW
LINE	PUT	

You can also use the POINT function in graphics. The SCREEN statement describes how to choose a mode, and the COLOR statement discusses the use of character attributes, such as reverse video. For more information on each of these commands, see the appropriate section in Chapter 3.

All the graphics commands have been fully optimized to take advantage of the 8086-88. They run significantly faster on these machines than on others.

1.1.1 THE COLOR ATTRIBUTE

The graphics statements DRAW, CIRCLE, PSET, PRESET, LINE, and PAINT let you specify a color attribute. You can select black or white. The range is 0 to 3. In screen mode 2, 0 or 2 selects black, and 1 or 3 selects white.

The drawing statements PSET, PRESET, LINE, CIRCLE, GET, PUT and PAINT require screen locations as pairs of (x,y) coordinates. The format is (< x > , < y >) where < x > and < y > are numeric expressions.

The screen coordinates are:

Mode	x (horizontal)	y (vertical)
2 (standard screen)	0–799	0–399

Point (0,0) is the upper left corner.

You can specify any integer coordinate value (in the range − 32768 to 32767) for < x > and < y > . Values outside this range, however, specify points not on the screen.

The statements PSET, PRESET, LINE, and CIRCLE also let you specify relative coordinates. In these cases you can write:

STEP (< x offset > , < y offset >)

< x offset > and < y offset > are numeric expressions. Their values are added to the current graphics cursor to determine the coordinate. The graphics cursor is the point on the screen where the last graphics point was referenced.

All of the graphics statements (excluding the POINT function) update the most recent point used. If you use the relative form on the second coordinate, it is relative to the first coordinate.

When you clear the screen with either the SCREEN or CLS statement, GW-BASIC sets the graphics cursor to the middle of the screen. On a standard screen that point is (400,200).

1.2 EVENT TRAPPING

Event trapping lets a program transfer control to a specific program line when a certain event occurs. Control transfers as if a GOSUB statement had been executed to the trap routine starting at the specified line number.

After completing the event, the trap routine executes a Return statement that continues program execution at the place where it was when the event trap occurred.

1.2.1 EVENT SPECIFIERS

The following are defined as "event specifiers":

- **COM (n):** where n is the number of the communications channel (1 or 2).

  Typically, the COM trap routine reads an entire message from the COM port before returning. Using the COM routine for single-character messages is problematic. At high baud rates, the overhead of trapping and reading for each character might let the interrupt buffer for COM overflow.

- **KEY (n):** where n is a function key number 1–11. 1 through 7 are the soft keys. 8 through 11 are the cursor direction keys, as follows: 8–Up, 9–Left, 10–Right, 11–Down. A KEY (0) ON, OFF, or STOP enables, disables, or stops all 11 key events.

  Note that KEY (n) ON is not the same statement as KEY ON. KEY ON displays the values of all the function keys on the twenty-fifth line of the screen.

When a key is trapped, that occurrence of the key is destroyed. Therefore, you cannot use the INPUT or INKEY$ statements to find out which key caused the trap. To assign different functions to particular keys, you must set up a different subroutine for each key; you cannot assign the various functions with a single subroutine.

CONTROLLING EVENT TRAPPING 1.2.2

Event trapping is controlled by the following statements:

< event specifier > ON

< event specifier > OFF

< event specifier > STOP

When an event is ON and a non-zero number is specified for the trap, GW-BASIC checks before starting each new statement to see if the specified event occurred. (It checks to see, for example, whether a function key was struck or whether a COM character came in.) If the event did occur, GW-BASIC performs a GOSUB to the line specified in the ON statement.

When an event is OFF, no trapping takes place and the event is not remembered even if it takes place.

When you specify STOP no trapping can take place. But if the event happens, it is remembered and an immediate trap takes place when an < event > ON is executed.

When a trap is made for a particular event, the trap automatically causes a "stop" on that event so recursive traps can never take place. The "return" from the trap routine turns that event trap back on unless an explicit OFF has been performed inside the trap routine. When an error trap takes place, it disables all trapping. Trapping never takes place when GW-BASIC is not in Direct mode executing a program.

1.2.3 ADDITIONAL CONTROLS

Event trapping includes the following statements:

ON < event specifier > GOSUB < line number >

This sets up an event trap line number for the specified event. A < line number > of 0 disables trapping for this event.

RETURN < line number >

This optional form of RETURN is primarily intended for use with event trapping. The event trap routine might want to go back into the GW-BASIC program at a fixed line number while still eliminating the GOSUB entry that the trap created.

Use this non-local RETURN with care. Any other GOSUB, WHILE, or FOR that was active at the time of the trap remains active. If the trap comes out of a subroutine, any attempt to continue loops outside the subroutine results in the "NEXT without FOR" error.

1.3 I/O

GW-BASIC allows device-independent I/O (input/output) files. Consequently, any type of input/output can be treated like I/O to a file, whether you are using a diskette or a printer, or are communicating with a device.

The following statements, commands, and functions support device-independent I/O. (For more information, see the individual descriptions in Chapters 3 and 4.)

BLOAD	INPUT$	LPOS	PRINT USING
BSAVE	KILL	LPRINT	PUT
CHAIN	LINE	MERGE	RESET
CLOSE	LIST	NAME	RUN
EOF	LLIST	OPEN	SAVE
FILES	LOAD	OPEN COM	WIDTH
GET	LOC	POS	WRITE
INPUT	LOF	PRINT	

There are several ways to save and retrieve file information, such as with SAVE, LOAD, and LIST.

Filenames

The physical file is described by its file specification (filespec).

The filespec is a string expression of the form:

[< device >][< filename >]

< device > is a physical device:

KYBD:	Keyboard	Input only
SCRN:	Video Display	Output only
LPT1:	First Line Printer	"
LPT2:	Second Line Printer	"
LPT3:	Third Line Printer	"
COM1:	RS-232-C Port A	Input/Output
COM2:	RS-232-C Port B	"
A: to O:	Disk Drives	"

< filename > is the name given to the file. The name conforms to the MS-DOS filename conventions. The name consists of two parts separated by a period:

< filename > .[< extension >]

The filename can be from 1 to 8 characters and the extension from 0 to 3 characters.

A default extension of .BAS is used on LOAD, SAVE, MERGE, RUN, CHAIN, BLOAD, and BSAVE commands if no period appears in the filename and if the filename has less than 9 characters. If the device is not specified, the current MS-DOS default disk drive is assumed.

File specification for communications devices is slightly different. The filename is replaced with a list of options specifying such items as baud rate and parity.

2

THE FULL SCREEN EDITOR

Using the Full Screen Editor for program development is a big time-saver. This chapter describes how to use the editor most effectively.

Because editing is a dynamic process, it is difficult to provide clear text examples of how edit commands work. The best way of understanding the editing process is to try editing a few lines while reading the edit commands in this chapter.

WRITING PROGRAMS 2.1

Any line of text that you type while GW-BASIC is in Direct mode is processed by the Full Screen Editor. GW-BASIC is always in Direct mode after the prompt Ok and until a RUN command is given.

GW-BASIC processes program statements in one of four ways:

1. Add a new line to the program. This occurs if the line number is legal (range is 0 through 65529) and at least one non-blank character follows the line number in the line.
2. Modify an existing line. This occurs if the line number matches the line number of an existing line in the program. GW-BASIC replaces the existing line with the text of the newly entered line.
3. Delete an existing line. This occurs if the line number matches the line number of an existing line and the entered line contains ONLY a line number.
4. Produce an error. If you try to delete a non-existent line you see the "Undefined line number" error message. If program memory is exhausted and you try to add a new line to the program, the "Out of memory" error message displays.

A GW-BASIC program line always begins with a line number, ends with a carriage return, and can contain a maximum of 250 characters. Note that any tabs that are embedded in multiple lines are replaced by spaces.

You can place more than one GW-BASIC statement on a line, but you must separate each statement from the last with a colon (:).

You can extend a logical line over more than one physical line by using the linefeed key. A linefeed sends subsequent text to the next line, without inserting a Return at the end of the previous line. Remember to signal the end of a logical line with a Return.

2.2 EDITING PROGRAMS

Begin modifying text by using the LIST command to display an entire program or range of lines on the screen. Then use the arrow keys and ALT-B (previous word), ALT-F (next word), and ALT-N (end of line) to move the cursor to the place requiring change. Finally, use one or more of the special function keys described in Chapter 2.3 to perform one of the following functions:

1. Overtype characters.
2. Delete characters to the left of the cursor.
3. Delete words or characters to the right of the cursor.
4. Insert characters at the cursor.
5. Add or append characters to the end of the current logical line.

A Return stores the modified lines in your program. You do not, however, have to move the cursor to the end of the logical line before you press Return. The Screen Line Editor remembers where each logical line ends and transfers the whole line, regardless of the cursor position.

You can move around the screen and make corrections to several lines at once. Then go back to the beginning of each line you changed and press Return.

You do not need to modify lines that contain GW-BASIC messages such as "OK". These lines are automatically erased if your cursor lands on them while you are editing. (The interpreter recognizes its own messages because they are terminated internally by FF hex to distinguish them from user text.)

FUNCTION KEYS 2.3

The Full Screen Editor recognizes the arrow keys, the Backspace, the Tab, the Return, plus 15 Alternate keys for moving the cursor on the screen, inserting characters, or deleting words or characters. The keys and their names are described in Table 2-1.

In addition, GW-BASIC supports seven special function keys. These keys display on the 25th line of your screen. Their functions are explained in the KEY statement.

Table 2-1: GW-BASIC Function Keys

INTERNAL CODE HEX	DEC	KEY	FUNCTIONAL NAME
01	01	ALT-A	Edit line buffer
02	02	ALT-B	Previous word
03	03	ALT-C	Break (stop program)
05	05	ALT-E	Erase to end of line
06	06	ALT-F	Next word
08	08	ALT-H	Destructive backspace
09	09	ALT-I	Tab (modulo 8)
0A	10	ALT-J	Linefeed
0B	11	ALT-K	Home
0C	12	ALT-L	Clear screen
0D	13	ALT-M	Carriage return (enter logical line)
0E	14	ALT-N	Append to end of line
12	18	ALT-R	Toggle insert/overtype mode
14	20	ALT-T	Refreshes the 25th line
15	21	ALT-U	Clear logical line
17	23	ALT-W	Delete word
1A	26	ALT-Z	Clear to end of window
1C	28	→	Cursor right
1D	29	←	Cursor left
1E	30	↑	Cursor up
1F	31	↓	Cursor down
7F	128	DEL	Delete character

Note: The VT-52 Escape sequences are not supported to maintain compatibility with IBM BASIC A.

Function Explanations

ALT-K — Moves the cursor to the upper left corner of the screen.

ALT-L — Clears the screen and positions the screen cursor in the upper left corner of the screen.

↑ — Moves the cursor up one line.

↓ — Moves the cursor down one line.

← Moves the cursor one column left. When the cursor is advanced beyond the left of the screen, it moves to the right side of the screen on the preceding line until it reaches the beginning of the screen.

→ Moves the cursor one position right. When the cursor is advanced beyond the right of the screen, it moves to the left side of the screen on the next line down until it reaches the end of the screen.

ALT-F Moves the cursor to the beginning of the next word. A word is defined as the characters A–Z, a–z, or 0–9 and is delineated by space characters. The next word is defined as the next character to the right of the cursor in the set [A..Z] or [0..9].

ALT-B Moves the cursor to the beginning of the previous word.

ALT-N Moves the cursor to the end of the logical line. GW-BASIC appends to the line any characters typed from this position.

ALT-T Refreshes the 25th line: if the function keys are displayed (KEY ON), ALT-T rewrites them. If the keys are not displayed (KEY OFF), ALT-T clears the 25th line.

ALT-E Erases to the end of the logical line from the current cursor position. All physical lines are erased up to the terminating carriage return.

ALT-R Toggles Insert/Overtype mode. Pressing this key changes mode to the other mode. Insert mode is automatically toggled to Overtype mode when you press any cursor movement key or Return.

When in Insert mode, GW-BASIC inserts typed characters at the cursor position and all characters on the physical line move to the right. Wrap-around is in effect: characters advanced off the right edge of the screen appear from the left edge of the screen on the following line.

When out of Insert mode, characters typed replace existing characters on the line.

TAB	When in Insert mode, pressing the Tab key inserts blanks from the current cursor position to the next tab stop. When out of Insert mode, pressing Tab moves the cursor right modulo 8 until the end of the screen.
DEL	Deletes one character under the cursor for each depression. All characters to the right then move one position left to fill in the space. If a logical line extends beyond one physical line, characters on subsequent lines move left one position to fill in the previous space, and the character in the first column of each subsequent line moves up to the end of the preceding line.
BS	Backspace. Deletes the last character typed, or deletes the character to the left of the cursor. All characters to the right of the cursor move left one position. Subsequent characters and lines within the current logical line move up as with the DEL key.
ALT-U	Erases the entire logical line.
ALT-C	Returns to Direct mode, without saving any changes that were made to the current line being edited.
ALT-A	Enters the line buffer at the current cursor position for editing.
ALT-J	Moves to the next physical line; scrolls if necessary.
ALT-W	Deletes characters up to the next word.
ALT-Z	Clears the screen to spaces from the cursor position to the end of the screen.

SYNTAX ERRORS 2.4

When GW-BASIC encounters a syntax error during program execution, it automatically enters EDIT at the line containing the error. For example:

```
10 A = 2$5      (you meant 10 A = 2^5)
RUN
Syntax Error in 10
10 A = 2$5
```

The Screen Line Editor displays the line in error and puts the cursor under the digit 1. You move the cursor right to the dollar sign ($) and change it to an up-arrow (^), then press Return. The corrected line is then stored in the program.

You destroy variables whenever you change a program line. If you want to examine the contents of a variable before you make a change, press ALT-C instead of moving the cursor. This command returns you to Direct mode, and preserves the variables.

3

GW-BASIC STATEMENTS

Table 3-1 is a summary of GW-BASIC statements. The sections that follow explain the statements in detail.

Table 3-1: GW-BASIC Statements

STATEMENT	PURPOSE
BEEP	Sounds the speaker
BLOAD	Loads a memory image into memory
BSAVE	Saves memory on a device
CALL	Interfaces 8086 machine language with GW-BASIC
CHAIN	Calls a program and passes variables to it from current program
CIRCLE	Draws circles or ellipses
CLS	Clears active screen
COLOR	Selects character attributes (reverse, underline, etc.)
COM	Enables or disables trapping
DATE$	Sets or retrieves current date
DEF SEG	Assigns segment address for referencing 8086 memory locations
DRAW	Draws a complex object
EDIT	Displays specified line for editing
GET and PUT (COM Files)	Allows fixed length I/O for COM
GET and PUT (Graphics)	Reads and writes pixels to screen
KEY	Designates soft function keys
KEY(n)	Activates and deactivates trapping
LCOPY	Dumps contents of screen to printer
LINE	Draws or removes lines, rectangles, and filled rectangles
LIST	Lists a program to screen or device
LOAD	Loads a program from device into memory, and optionally runs it
LOCATE	Moves cursor to specified position
MERGE	Merges lines from an ASCII program file into program in memory
ON COM	Sets up line number for trapping
ON KEY(n)	Sets up function key to designate line number for trapping
OPEN	Establishes addressability between device and I/O buffer in data pool
OPENING A COM FILE	Activates serial I/O communications
OUT	Sends a byte to output port
PAINT	Fills in area on screen with selected color (black or white)
PLAY	Plays music

STATEMENT	PURPOSE
PSET	Displays or removes pixel
PRESET	Removes or displays pixel
RETURN	Returns from GOSUB
SAVE	Saves program on specified device
SCREEN	Sets screen attributes
SOUND	Generates sound at specified frequency for specified duration
TIME$	Sets or retrieves current time
WAIT	Suspends program execution
WIDTH	Sets printed line width

3.1 BEEP

FORMAT:

BEEP

PURPOSE:

Sounds the speaker at 800 Hz for 1/4 second.

REMARKS:

Both BEEP and PRINT CHR$(7); have the same effect.

EXAMPLE:

```
2430 IF X < 20 THEN BEEP 'if X is out of
                         'range, complain.
```

BLOAD 3.2

FORMAT:

BLOAD <**filespec**> [,<**offset**>]

PURPOSE:

Loads a memory image into memory.

REMARKS:

<filespec> is a string expression returning a valid file specification. This file specification must follow the format described in Chapter 1.3; the extension, however, must be different. The only valid extensions are:

(none)	(no extension)
.B	for GW-BASIC programs in the internal format (created with the SAVE command).
.P	for protected GW-BASIC programs in the internal format (created with SAVE ,P command).
.A	for GW-BASIC programs in ASCII format (created with SAVE ,A command).
.M	for memory image files (created with BSAVE command).
.D	for data files (created by OPEN followed by output statements).

If you do not put an extension in the filename, GW-BASIC assigns the default extension .BAS.

If you omit the device name, GW-BASIC assumes the current diskette drive.

< offset > is a numeric expression in the range 0 to 65535. This is the address at which the loading starts, specified as an offset into the segment declared by the last DEF SEG statement. If you omit the < offset >, GW-BASIC assumes the < offset > specified in the last BSAVE.

WARNING: BLOAD does not perform address range checking. That is, it is possible to BLOAD anywhere in memory. You should be absolutely sure you are not overwriting the operating system, GW-BASIC, or your own program.

EXAMPLE:

```
10 'Load an assembly program into GW-BASIC
20 'assuming no program has been loaded.
30 DEF SEG = &H16F0 'set to previously
40 'determined location
50 BLOAD "MOVE",0 'load the CALLable program
```

Note: See the CALL statement for details on safely loading a module.

3.3 BSAVE

FORMAT:

BSAVE < **filespec** >, < **offset** >, < **length** >

PURPOSE:

Saves portions of the computer's memory on the specified device.

REMARKS:

< filespec > is a string expression returning a valid file specification as described in BLOAD.

< offset > is a numeric expression in the range 0 to 65535. This is the address at which the saving starts, specified as an offset into the

segment declared by the last DEF SEG statement.

< length > is a valid numeric expression returning an unsigned integer in the range 1 to 65535. This is the length of the memory image you want to save.

EXAMPLE:

```
10 'save the first 100 bytes of memory located
20 'at the start of GW-BASIC's Data Segment.
30 DEF SEG
40 BSAVE "PROGRAM.M",0,100
```


CALL 3.4

FORMAT:

CALL < variable name > [(< argument list >)]

PURPOSE:

Interfaces 8086 machine language programs with GW-BASIC. Use CALL instead of the old-style user call (x = USR(n)).

REMARKS:

< variable name > contains the address that is the starting point in memory of the subroutine being CALLed.

< argument list > contains the variables or constants, separated by commas, that you want to pass to the routine.

The CALL statement conforms to the INTEL PL/M-86 calling conventions outlined in Appendix H of the *INTEL PL/M-86 User's Guide* (#121636-02). GW-BASIC follows the rules described for the Medium case.

When you invoke the CALL statement, the following occurs:

1. For each parameter in the argument list, the 2-byte offset into the data segment [DS] of the parameter's location is pushed onto the stack.
2. The return address code segment [CS] and offset [IP] are pushed onto the stack.
3. Control is transferred to your routine via the segment address given in the last DEF SEG statement and offset given in < variable name >.

Your routine now has control. Parameters can be referenced by moving the stack pointer [SP] to the base pointer [BP] and adding a positive offset to [BP].

RULES:

The assembly language subroutine must follow these rules to work correctly:

1. It must be declared FAR.
2. Segment registers DS and ES must be restored to their entry values before returning to GW-BASIC.
3. The general purpose registers (AX, BX, CX, DX, SI, DI, and BP) can have any value when returning to GW-BASIC.
4. The assembly language routine **must not** change the length of any GW-BASIC strings.
5. The assembly language routine must perform a RET (n) (where n = 2 times the number of parameters) to restore the stack pointer to its proper value.
6. Values can be returned to GW-BASIC by passing a parameter that the result will be returned in.

GW-BASIC Data Types

To manipulate data passed to an assembly language subroutine, you must understand how the various data types are represented in memory. When a subroutine is called, GW-BASIC passes the address of one of the following data representations:

1. Integer: two-byte two's-complement number.

2. Single-Precision Number: four-byte binary floating-point quantity. The most significant byte contains the value of the exponent minus 127. The remaining three bytes contain the mantissa. The most significant byte of the mantissa contains the sign bit, followed by the seven highest bits of the mantissa. A positive number is represented with a 0 as the sign bit, and a negative number with a 1 as the sign bit. The binary point is to the left of the most significant bit of the mantissa. A 1 is always assumed to exist immediately to the right of the mantissa, although it is not represented. Thus the number is represented as:

   ((sign) (1.(mantissa)*2)^(exponent − 127)

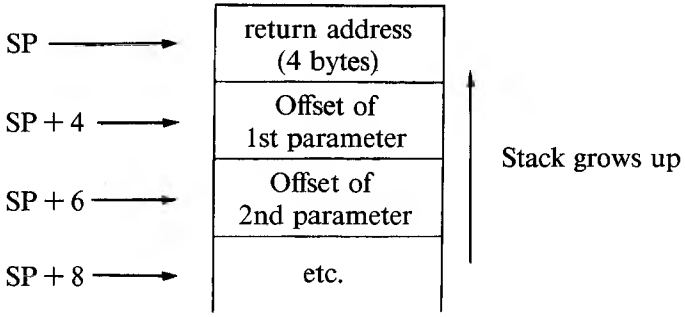
3. Double-Precision Number: eight-byte binary floating-point quantity. It is represented exactly the same as a single-precision number, except that the mantissa consists of 55 bits (7 bytes less the sign bit).

4. String: GW-BASIC passes the offset address of a "string descriptor" which is a three-byte data structure. The first byte of the string descriptor contains the length of the string. The last two bytes contain the address where the actual ASCII string is located. The assembly language subroutine is allowed to modify the string (within the number of bytes specified by the descriptor), but it must not change the string descriptor.

5. Array: arrays are made up of sequential elements of the array type. For example, an integer array containing twenty elements is represented as twenty sequential integers in memory.

Passing Parameters

GW-BASIC passes all subroutine parameters by reference. In a CALL statement, the offset of each parameter's address is pushed onto the stack in the same order that the parameters are listed in the procedure call. Note that all parameters to the assembly language subroutine must be variables. Upon entry to the subroutine, the stack is arranged as follows:



The parameters can then be referenced by using the BP register to get their address off of the stack.

EXAMPLE:

This example shows how to load an assembly language subroutine from a GW-BASIC program. The assembly language routine performs modulo arithmetic on two integers, returning the remainder that results when the first integer is divided by the second. In this example, the assembly language module is loaded at address 1664:0 Hex, but this address will be different for different applications. The method of determining this address is explained after the example.

```
10 '
20 ' load the MODULO routine
30 '
40 DEF SEG = &H1664
50 BLOAD "MODULO",0
60 MODULO = 0
70 '
80 ' call the MODULO routine with some
90 ' sample data
100 A% = 140
110 B% = 11
120 REMAINDER% = 0
130 CALL MODULO(A%,B%,REMAINDER%)
140 PRINT A%;"modulo";B%;"is";REMAINDER%
150 END
```

The assembly language module you should use with the CALL statement is:

```
name        modulo

code        segment public 'code'
assume      cs:code,ds:code

modulo      proc        far

; This module is called from GW-BASIC with 3
; parameters, using the CALL statement. It
; divides the first parameter by the second
; and returns the remainder in the third.
```

```
mov     bp, sp      ;BP used to get parameters
mov     bx, [bp+8]  ;BX=pointer to dividend
mov     ax, [bx]    ;AX=value of dividend
mov     bx, [bp+6]  ;BX=pointer to divisor
mov     cx, [bx]    ;CX=value of divisor
mov     dx, 0       ;DX:AX=dividend
idiv    cx          ;AX=quotient,DX=remainder
mov     bx, [bp+4]  ;BX=address of result
mov     [bx], dx    ;return result to GW-BASIC
ret     2*3         ;no. of parameters

modulo  endp
code    ends
        ends
```


Loading the Assembly Language Module

To call the assembly language module, you must know its location (address). The BLOAD statement allows you to load the module at any physical address. However, to use the BLOAD statement to load a module, you must first create the disk file containing the module using MS-LINK and DEBUG and the BSAVE statement, as follows:

1. After assembling your module to create the object file, use the linker to create the .EXE file. Use the /HIGH switch when linking so that the module will load in high address memory.
2. Use the debugger to load the .EXE file produced in step 1.
3. Display the register values (with the R command) to determine where the subroutine was loaded. Write down the values contained in the CS:IP register pair and the CX register. The CS:IP register pair contains the starting address of the subroutine and the CX register contains its length.

4. Load and execute GW-BASIC from DEBUG with this sequence of commands:

```
NGWBASIC.EXE
L
N
G
```

Note that your assembly language module is still loaded in high address memory.

5. Set the segment value in GW-BASIC with a DEF SEG statement:

```
DEF SEG = (value in CS register)
```


6. Save the module with a BSAVE statement:

```
BSAVE "module_name", (value in IP reg.),
(value in CX reg.)
```

The assembly language subroutine is now ready to be called from your GW-BASIC program. The following statements are required in your GW-BASIC program before the subroutine can be called:

```
DEF SEG = (value in CS register)
BLOAD "module_name", (value in IP register)
SUBROUTINE = (value in IP register)
```

The subroutine can then be called with statements of the form:

```
CALL SUBROUTINE(PARAMETER1, PARAMETER2, ...)
```

3.5 CHAIN

FORMAT:

CHAIN [MERGE] < filespec > [,[< line number exp >]
[,ALL][,DELETE < range >]]

PURPOSE:

Calls a program and passes variables to it from the current program.

3

REMARKS:

< filespec > is a string expression returning a valid file specification as described in Chapter 1.3.

< filespec > is the only difference between MS-BASIC and GW-BASIC. Refer to your MS-BASIC manual for a complete description.

EXAMPLE:

```
995 CHAIN "PROG1", 1000
```

3.6 CIRCLE

FORMAT:

CIRCLE (< xcenter > , < ycenter >), < radius >
[, < attribute > [, < start > , < end > [, < aspect >]]]

PURPOSE:

Draws an ellipse with center (< xcenter > , < ycenter >) and radius < radius > .

REMARKS:

< xcenter > is a numeric expression in the integer range used as the x-coordinate of the center of the ellipse.

< ycenter > is a numeric expression in the integer range used as the y-coordinate of the center of the ellipse.

< radius > is a numeric expression in the integer range used as the radius of the ellipse.

< attribute > is an expression returning the value 0 to 3, which is used to determine the color of the ellipse. An attribute of 0 or 2 draws an ellipse of the background color.

< start > and < end > are angle parameters expressed as radian arguments between 0 and 2*PI that allow you to specify where drawing of the ellipse begins and ends.

< aspect > is the ratio of the x radius to the y radius. The default aspect ratio is 5.25/8.00 in hi-res, and gives a visual circle (assuming a standard monitor screen aspect ratio).

The CIRCLE statement draws an ellipse with a center and radius, as indicated by the first of its arguments. The default attribute is 1 (white). If the start or end angle is negative, the ellipse is connected to the center point with a line, and the angles are treated as if they are positive (note that this is different from adding 2*PI).

If the aspect ratio is less than one, then the radius is given in x-pixels. If it is greater than one, the radius is given in y-pixels. You can use the standard relative notation to specify the center point.

For more information, see Chapter 1.1.

EXAMPLE:

```
10 CIRCLE (399,199),100,1   'circle at center
                            'of screen
```

3.7 CLS

FORMAT:

CLS [<n>]

PURPOSE:

Erases the current active screen page.

REMARKS:

<n> is a number from 0 to 2.

1. If the screen is in Alpha mode, it clears to white or underlined, depending on the current foreground and background colors (see Chapter 3.8). If the screen is in Graphics or Hi-res mode, the entire screen buffer clears to black.
2. You can also clear the screen with ALT-L.
3. **Note:** The SCREEN and WIDTH statements force a screen clear if the resultant Screen mode created is different from the current mode.
4. Your computer does not have the capability to clear the text or graphics screens separately. Use CLS with a valid parameter to clear both screens.

EXAMPLE:

```
10 CLS          'Clears the screen.
15 CLS 2        'Clears the screen also.
```

COLOR 3.8

FORMAT:

COLOR [<foreground>, <background>]

PURPOSE:

Selects the foreground and background colors.

REMARKS:

<foreground> is an unsigned integer from 0 to 15 that determines the color of the character.

<background> is an unsigned integer from 0 to 15 that determines the color the character is placed over.

The COLOR statement is valid only in Screen 0 (Alpha mode). Since GW-BASIC has no colors available, use this statement to select reverse-video characters (black on white), underlined, or highlighted characters.

You can obtain the following effects with the specified combinations of foreground and background colors.

FOREGROUND	BACKGROUND	EFFECT
7	0	Normal, white on black
1	0	Underlined, white on black
0	7	Reverse, black on white
15	0	Highlight, white on black
9	0	Highlight, underlined white on black
8	7	Reverse highlight
0	0	Invisible (black)

1. Any values that you enter outside these ranges result in the "Overflow" or "Illegal Function Call" error. Previous values are retained.
2. You can omit any parameter. Omitted parameters assume the old value.

EXAMPLE:

```
100 COLOR 0,7 'reverse video
110 COLOR ,0  'invisible characters
```


3.9 COM

FORMAT:

COM(< n >) ON

COM(< n >) OFF

COM(< n >) STOP

PURPOSE:

Enables or disables trapping of communications activity to the indicated serial port.

0 = both ports
1 = port A
2 = port B

REMARKS:

You must execute a COM(< n >) ON statement to allow trapping by the ON COM(< n >) statement. The combination of COM(< n >) ON and a non-zero line number specified in the ON COM(< n >) statement tells GW-BASIC to check for new characters from the serial port before starting any new statements.

If COM(< n >) is OFF GW-BASIC does not execute event traps and does not remember the specified event even if it does take place.

COM(< n >) STOP also inhibits trapping. If any communications activity takes place, however, GW-BASIC remembers and executes an immediate trap at the next COM(< n >) ON.

EXAMPLE:

```
10 PORT.A = 1
20 COM(PORT.A) ON   'enable com trapping on
                    'port a
    .
    .
    .
100 COM(PORT.A) STOP 'temporarily disable
                     'trapping
    .
    .
    .
300 COM(PORT.A) ON 'enable trapping
                   'immediately
    .
    .
    .
500 COM(PORT.A) OFF 'disable trapping &
                    'forget events
```


DATE$ 3.10

FORMAT:

DATE$ = < string expr >

To set the current date.

< string expr > = DATE$

To get the current date.

PURPOSE:

Sets or retrieves the current date.

REMARKS:

< string exp > is a valid string literal or variable.

The current date is fetched and assigned to the string variable if DATE$ is the expression in a LET or PRINT statement.

The date is stored if DATE$ is the target of a string assignment.

RULES:

1. If < string exp > is not a valid string, GW-BASIC generates the "Type mismatch" error. Previous values are retained.
2. For < string var > = DATE$, DATE$ returns a 10-character string in the form "mm-dd-yyyy", where mm is the month (01 to 12), dd is the day (01 to 31) and yy is the year (1980 to 2099).
3. For DATE$ = < string expr >, < string expr > may take one of the following forms:

   "mm-dd-yy"
   "mm/dd/yy"
   "mm-dd-yyyy"
   "mm/dd/yyyy"

   If any of the values are out of range or missing, GW-BASIC generates the "Illegal Function Call" error. Any previous date is retained.

EXAMPLE:

```
DATE$ = "10-21-82"
Ok
PRINT DATE$
10-21-1982
Ok
```

DEF SEG 3.11

FORMAT:

DEF SEG [= <address>]

PURPOSE:

Assigns the current segment address that the subsequent CALL or POKE statement, or USR or PEEK functions will reference.

REMARKS:

<address> is a valid numeric expression returning an unsigned integer in the range 0 to 65535. The <address> specified is saved for use as the segment required by the PEEK, POKE and CALL statements.

RULES:

1. Any value entered outside of this range results in an "Illegal Function Call" error. The previous value is retained.
2. If you omit the address option, DEF SEG uses GW-BASIC's data segment. This is the initial default value.
3. If you supply this address option, GW-BASIC uses the value as the segment portion of the offset address specified by the PEEK, POKE, or CALL.
4. **Note:** DEF and SEG **must** be separated by a space. Otherwise, GW-BASIC interprets the statement DEFSEG = 100 to mean: "assign the value 100 to the variable DEFSEG".

EXAMPLE:

```
10 DEF SEG=0  'Set segment to interrrupt table
20 DEF SEG    'Restore segment to BASIC's DS.
30 DEF SEG=&H8112 'Set segment
40 A=PEEK (100) 'A=byte at 8112:100 or 81220
                'absolute
```

3.12 DRAW

FORMAT:

DRAW <**string exp**>

PURPOSE:

Draws a complex object as specified by <string exp>. <string exp> is a string expression returning a valid formatted string, using the movement commands.

The DRAW verb combines most of the capabilities of the other graphics statements into an easy-to-use object definition language called "Graphics Macro Language." A GML command is a single character within a string, optionally followed by one or more characters.

MOVEMENT COMMANDS:

Each of the following movement commands begin movement from the current graphics position. This is usually the coordinate of the last graphics point plotted with another GML command, LINE, or PSET. The current position defaults to the center of the screen (400,200) when a program is run.

U [<n>]	Move up (scale factor * n) points
D [<n>]	Move down
L [<n>]	Move left
R [<n>]	Move right
E [<n>]	Move diagonally up and right
H [<n>]	Move diagonally up and left
G [<n>]	Move diagonally down and left
F [<n>]	Move diagonally down and right

The commands move one unit if no argument is supplied.

M <x,y>	Move absolute or relative. If x is preceded by a "+" or "−", x and y are added to the current position; the new point is connected with the current position by a line. Otherwise, GW-BASIC draws a line from the current position to the point x,y.

The following prefix commands can precede any of the movement commands:

B	Move but don't plot any points.
N	Move but return to original position when done.
A <n>	Set angle n. n can range from 0 to 3, where 0 is zero degrees, 1 is 90, 2 is 180, and 3 is 270. Figures rotated 90 or 270 degrees are scaled so that they appear the same size as with 0 or 180 degrees on a monitor screen with the standard aspect ratio of 3 to 2.
C <n>	Set attribute n. n can range from 0 to 3; even values remove the dot and odd values display the dot.
S <n>	Set scale factor. n can range from 1 to 255. The scale factor is multiplied by the distances given with U, D, L, R, or relative M commands to get the actual distance traveled.
X <string>	Execute substring. This powerful command allows you to execute a second substring from a string, much like GOSUB in BASIC. You can have one string execute another, which executes a third, and so on. Numeric arguments can be constants like "123" or "=variable;", where variable is the name of a variable.

EXAMPLE:

To draw a box:

```
10 SCREEN 2    'must be in graphics mode
20 SIDE.LEN = 50    'set length of each side
30 DRAW "C1;U=SIDE.LEN;R=SIDE.LEN;D=SIDE.LEN;
   L=SIDE.LEN;"
```

To draw a triangle:

```
10 SCREEN 2      'must be in graphics mode
20 DRAW "C1;E15;F15;L30"
```


3.13 EDIT

FORMAT:

EDIT <**line number**>

EDIT <.>

PURPOSE:

Displays the line specified and positions the cursor under the first digit of the line number. You can then modify the line using the Full Screen Editor.

REMARKS:

<line number> is the program line number of a line existing in the program. If there is no such line, GW-BASIC displays the "Undefined Line Number" error message.

<.> always gets the last line referenced by an EDIT statement, LIST command, or error message.

GET AND PUT FOR COM FILES 3.14

FORMAT:

GET < **file number** > , < **nbytes** >

PUT < **file number** > , < **nbytes** >

PURPOSE:

Allows fixed-length I/O for COM.

REMARKS:

< file number > is an integer expression returning a valid file number.

< nbytes > is an integer expression returning the number of bytes to transfer into or out of the file buffer. nbytes cannot exceed the value set by the /S: switch when BASIC was invoked.

Because of the low performance associated with telephone line communication, you should not use GET and PUT in such applications.

EXAMPLE:

```
10   '***  Program to Send an ASCII file over Port A
20 INPUT "Enter ASCII file to transmit:";FILNME$
30 OPEN "COM1:9600,O,7,1" AS #1     'init PORT A
40 OPEN "Output" ,#2, FILNME$    'open ASCII file
50 WHILE NOT(EOF(2))
60 GET #2      'load the file buffer with a rec
70 PUT #1,128 'send the record out port A
80 WEND
90 CLOSE #1, #2
100 END
```

3.15 GET AND PUT FOR GRAPHICS

FORMAT:

GET (<x1-coord>,<y1-coord>)-(<x2-coord>,<y2-coord>), <array name>

PUT (<x1-coord>,<y1-coord>),<array>[,<action verb>]

PURPOSE:

Reads (GET) or writes (PUT) pixels to or from an area of the screen.

REMARKS:

<x1-coord> and <y1-coord> are numeric expressions returning a value in the integer range that specifies one corner of the rectangular area.

<x2-coord> and <y2-coord> are numeric expressions returning a value in the integer range which specifies the opposite corner of the rectangular area.

<array name> is a previously dimensioned array to receive the graphics points.

<array> is an array containing graphics information.

<action verb> is one of:

PSET, PRESET, AND, OR, XOR

Use the PUT and GET statements to transfer graphics images to and from the screen. PUT and GET make animation and high-speed object motion possible in either Graphics mode.

The GET statement transfers the screen image inside a rectangle into the array. You define the rectangle by specifying the coordinates in the same way you would using the line statement with the ',B' or ',BF' option.

The array is only a place to hold the image and can be of any type except string. It must be dimensioned large enough to hold the entire image. After a GET the contents of the array are meaningless (unless the array is of type integer).

The PUT statement transfers the image stored in the array onto the screen. The specified point is the coordinate of the top left corner of the image. If the image to be transferred is too large to fit on the screen, GW-BASIC displays the "Illegal Function Call" error.

Use the action verb to interact the transferred image with the image already on the screen. PSET transfers the data onto the screen verbatim.

PRESET is the same as PSET except that PRESET produces a negative image (black on white). Use AND to transfer the image only if an image already exists under the transferred image.

Use OR to superimpose the image onto the existing image.

XOR inverts the points on the screen where a point exists in the array image. This behavior is exactly like the cursor on the screen. XOR has a unique property that makes it especially useful for animation: when an image is PUT against a complex background twice, the background is restored unchanged. This allows you to move an object around the screen without obliterating the background.

Note: The default action mode is XOR.

It is possible to GET an image in one mode and PUT it in another, although the effect might be unusual because of the way points are represented in each mode.

AND, OR and XOR have the following effects on color:

AND

array attr	screen attrib 0	1	2	3
0	0	0	0	0
1	0	1	0	1
2	0	0	2	2
3	0	1	2	3

OR

array attr	screen attrib 0	1	2	3
0	0	1	2	3
1	1	1	3	3
2	2	3	2	3
3	3	3	3	3

XOR

array attr	screen attrib 0	1	2	3
0	0	1	2	3
1	1	0	3	2
2	2	3	0	1
3	3	2	1	0

You can animate an object in the following way:

1. PUT the object(s) on the screen.
2. Recalculate the new position of the object(s).
3. PUT the object(s) on the screen a second time at the old location(s) to remove the old image(s).
4. Return to step 1, this time PUTting the object(s) at the new location.

This movement leaves the background unchanged. You can cut down flicker by minimizing the time between steps 4 and 1, and by making sure that there is enough time delay between steps 1 and 3. If you are animating more than one object, you should process all objects at once, one step at a time.

You don't have to preserve the background; you can animate by using the PSET action verb. Leave a border around the image when you first get it as large or larger than the maximum distance the object will move. Then, when the object is moved, the border effectively erases any points. This method might be somewhat faster than the method using XOR since you need only one PUT to move an object (although you must PUT a larger image).

The storage format in the array is as follows:

2 bytes giving x dimension in bits
2 bytes giving y dimension
The array data itself

The data for each row of pixels is left-justified on a byte boundary, so if you store less than a multiple of 8 bits, the rest of the byte is filled with zeros. The required array size in bytes is:

4 + INT((x* < bits/pixel > + 7)/8)*y

where < bits/pixel > is 1 in screen mode 2.

The bytes per element of an array are:

2 for integer
4 for single precision
8 for double precision

For example, if you want to GET a 10-by-12 image into an integer array, the number of bytes required is 4 + INT((10*2 + 7)/8)*12, or 40 bytes. So you need an integer array with at least 20 elements.

You can examine the x and y dimensions and even the data itself if you use an integer array. The x dimension is in element 0 of the array, and the y dimension is in element 1. Remember that integers are stored low byte first, then high byte, and that the data is transferred high byte first (leftmost) and then low byte.

3.16 KEY

FORMAT:

KEY <**key number**>,<**string expression**>

KEY LIST

KEY ON

KEY OFF

3

PURPOSE:

Designates function keys as "soft keys." You can assign any one or all of the seven special function keys a 15-byte string. When you press the key, the string is input to GW-BASIC.

Initially, the soft keys are assigned the following values:

1	RUN	5	LOAD "
2	CONT	6	FILES
3	LCOPY	7	LIST
4	SAVE "		

REMARKS:

<key number> is the key number, an expression returning an unsigned integer in the range 1 to 7.

<string expression> is the key assignment text, any valid string expression up to 15 characters in length.

KEY ON — This is the initial setting. It displays the key values on the 25th line. Only the first 9 characters of each value display. "<" in the string indicates a carriage return.

KEY OFF	Erases the soft key display from the 25th line.
KEY LIST	Lists all seven soft key values on the screen. All 15 characters of each value are displayed.
ALT-T	Refreshs the 25th line if the function keys are on. If the function keys are off, ALT-T clears line 25.

RULES:

1. If the value returned for < key number > is not in the range 1 to 7, the "Illegal Function Call" error displays. GW-BASIC retains the previous key assignment string.
2. The key assignment string may be 1 to 15 characters in length. If the string is longer, the first 15 characters are assigned.
3. Assigning a null string (string of length zero) to a soft key disables the function key as a soft key.
4. When a soft key is assigned, the INKEY$ function returns one character of the soft key string per invocation. If a soft key is disabled, INKEY$ returns the code given for that key.

EXAMPLE:

```
50 KEY ON
```

Display the soft keys on the 25th line.

```
200 KEY OFF
```

Erase soft key display.

```
10 KEY 1,"MENU"+CHR$(13)
```

Assigns the string "MENU"(cr) to soft key 1. You can use such assignments for rapid data entry. This example might be used in a program to select a menu display.

```
20 KEY 1," "
```

Erases soft key 1.

The following routine initializes the first five soft keys:

```
10 KEY OFF      'Turn off key display during init
20 DATA KEY1, KEY2, KEY3, KEY4, KEY5
30 FOR I=1 TO 5
40 READ SOFTKEYS$(I)
50 KEY I,SOFTKEYS$(I)
60 NEXT I
70 KEY ON       'now display new softkeys.
```


3.17 KEY(n)

FORMAT:

KEY(<n>) ON

KEY(<n>) OFF

KEY(<n>) STOP

PURPOSE:

Activates and deactivates trapping of the specified key.

REMARKS:

<n> is a numeric expression returning a value between 0 and 11 and indicates the key to be trapped.

0	Keys 1–11
1–7	Function keys 1 to 7
8	Up arrow
9	Left arrow
10	Right arrow
11	Down arrow

Execute a KEY(< n >) ON statement to activate trapping of function key or cursor control key activity. If you then specify a non-zero line number in the ON KEY(< n >) statement, every time GW-BASIC starts a new statement it checks to see if the specified key was pressed. If so, GW-BASIC performs a GOSUB to the line number specified in the ON KEY(< n >) statement.

If KEY(< n >) is OFF, no trapping takes place and the event is not remembered even if it does take place.

If a KEY(< n >) STOP statement has been executed, no trapping takes place. But GW-BASIC remembers if the specified key is pressed, and executes an immediate trap when the next KEY(< n >) ON executes.

KEY(< n >) ON has no effect on whether the soft key values are displayed on the 25th line.

LCOPY 3.18

FORMAT:

LCOPY

PURPOSE:

Dumps the contents of the display screen to a graphics printer.

REMARKS:

This statement allows you to get hard copies of the graphics displays generated by GW-BASIC. The program supports these printers:

Epson FX
Epson MX
Tally
C. Itoh
C. Itoh S
Okidata

You must configure GW-BASIC to use one of these printers (see Chapter 6.2, "Printer Installation"). Once configured, LCOPY can use only the printer specified in the sign-on message. If you do not configure any printer, GW-BASIC returns "Illegal Function Call."

EXAMPLE:

```
10 SCREEN 2 'set up for some graphics
20 KEY OFF  'turn off the function key display
30 CLS 2    'clear the screen
40 FOR RADIUS = 20 TO 200 STEP 10
50 CIRCLE (400,200),RADIUS  'draw some graphics
60 NEXT RADIUS
70 LCOPY      'copy the screen to the printer
80 END
```


3.19 LINE

FORMAT:

LINE [(<x>,<y>)]-(<x1>,<y1>)[[,[<attribute>][,B[F]]

PURPOSE:

Draws or removes straight lines, rectangles, and filled rectangles.

REMARKS:

<x>, <y>, <x1>, and <y1> are valid coordinates. <attribute> is an expression returning a value 0 to 3; GW-BASIC uses that value to determine the color of a line. Even attributes remove dots and odd attribute display dots. (See Chapter 1.1 for more information.)

The final argument is ",B" (box) or ",BF" (filled box).

LINE (x,y)-(x1,y1) draws a line from point (x,y) to point (x1,y1).

LINE −(x1,y1) draws a line from the previous graphics cursor to the point (x1,y1).

LINE (x1,y1)−(x2,y2),B draws a rectangle with (x1,y1) as one corner and (x2,y2) as the opposite diagonal corner. Using the B argument replaces the following four LINE commands:

```
LINE (x1,y1)−(x2,y1)
LINE (x1,y1)−(x1,y2)
LINE (x2,y1)−(x2,y2)
LINE (x1,y2)−(x2,y2)
```


",BF" draws the same rectangle as ",B" but also fills in the interior points with the selected attribute.

When you give out-of-range coordinates in the LINE command, the coordinate which is out of range takes the closest legal value—negative values become zero, y values greater than 399 become 399 and x values greater than 799 become 799.

LINE (x,y)−(x1,y1),BF draws a rectangle and fills the entire rectangle.

EXAMPLE:

Draw lines forever using random attribute:

```
10 CLS
20 LINE -(RND*799,RND*399),INT(RND*4)
30 GO TO 20
```

Draw alternating pattern—line on, line off:

```
10 FOR X = 0 TO 799
20 LINE (X,0)-(X,399),X AND 1
30 NEXT
```

Draw lines forever, using random attribute, and filling the rectangles:

```
10 CLS
20 LINE (RND*799,RND*399)-(RND*799,RND*399),
   INT(RND*4),bf
30 GO TO 20
```

3.20 LIST

FORMAT:

LIST [[<line no.> [– [<line no.>]]] [,<filespec>]]

PURPOSE:

Lists a program to the screen or other devices.

REMARKS:

<line no.> is a valid line number in the range 0 to 65529.

<filespec> is a valid string expression returning a valid file specification.

RULES:

1. If you omit the optional device specification, GW-BASIC lists the specified lines to the screen.
2. You can stop listings directed to the screen at any time by pressing ALT-C.
3. If you omit the line range, GW-BASIC lists the entire program.
4. When you use the dash (-) in a line range, you have three options:
   a. If you give only the first number, that line and all higher numbered lines are listed.
   b. If you give only the second number, all lines from the beginning of the program through the given line are listed.
   c. If you give both numbers, the inclusive range is listed.

EXAMPLES:

```
LIST ,"LPT1:"
```

List program to the Line Printer.

```
LIST 10-20
```

List lines 10 through 20 to Screen.

```
LIST 10- ,"SCRN:"
```

List lines 10 through last to Screen.


```
LIST -200
```

List first through line 200 to Screen.

```
LIST 1000-1045,"COM1:4800,O,5,2"
```

List lines 1000 through 1045 to serial port A, setting the baud rate, parity, data bits, and stop bits.

LOAD 3.21

FORMAT:

LOAD "<**filespec**>" **[,R]**

PURPOSE:

Loads a program from the specified device into memory, and optionally runs it.

REMARKS:

< filespec > is a valid string expression for the file specification.

LOAD lets you enter programs from the communications ports. GW-BASIC offers this option in addition to the MS-BASIC options. Refer to your MS-BASIC manual for additional information.

EXAMPLE:

```
LOAD "COM1:4800,0,7,1"R
```


3.22 LOCATE

FORMAT:

LOCATE [< row >] [,[< col >] [,[< cursor >][,[< start >] [, < stop >]]]]

PURPOSE:

Moves the cursor to the specified position on the active screen. Optional parameters turn the cursor on and off and define the start and stop raster lines for the cursor.

REMARKS:

LOCATE moves the cursor to the specified position. Subsequent PRINT statements begin placing characters at this location. You can also use this statement to turn the cursor on or off or to change the size of the cursor.

< row > is the screen line number, a numeric expression returning an unsigned integer in the range 1 to 25.

< col > is the screen column number, a numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80, depending on the screen width.

< cursor > is a boolean value indicating whether or not the cursor is visible; use 0 for off, non-zero for on.

< start > is the cursor starting scan line, a numeric expression returning an unsigned integer in the range 0 to 31. GW-BASIC, however, recognizes and displays only 0 to 15.

< stop > is the cursor stop scan line, a numeric expression returning an unsigned integer in the range 0 to 31. GW-BASIC, however, recognizes and displays only 0 to 15.

If you enter any values outside these ranges, GW-BASIC displays the "Illegal Function Call" error and retains previous values.

You can omit any of the parameters. Omitted parameters assume the previous value.

If you give the start scan line parameter and omit the stop scan line parameter, stop assumes the start value. If you omit both, the start and stop scan lines retain their previous values.

EXAMPLE:

```
10 LOCATE 1,1
```

Move to the home position in the upper left corner.

```
20 LOCATE ,,1
```

Make the cursor visible; position remains unchanged.

```
30 LOCATE ,,,15
```

Cursor position and visibility remain unchanged. Set the cursor to display at the bottom of the character starting and ending on scan line 15.

```
40 LOCATE 5,1,1,0,15
```

Move to line 5, column, turn cursor on; cursor covers entire character cell starting at scan line 0 and ending at scan line 15.

3.23 MERGE

FORMAT:

MERGE < **filespec** >

PURPOSE:

Merges the lines from an ASCII program file into the program currently in memory.

REMARKS:

< filespec > is a string expression that returns a valid file specification. Refer to your MS-BASIC manual for additional syntax information.

EXAMPLE:

```
MERGE "COM2:4800,0,7,1"
```

ON COM 3.24

FORMAT:

ON COM(< n >) GOSUB < line number >

PURPOSE:

Sets up a line number for GW-BASIC to trap when there is information coming into the communications buffer.

REMARKS:

< n > is the number of the communications port, where 1 is port A and 2 is port B.

< line number > is the starting line number of the routine to handle the information coming from the port. A line number of 0000 (zero) disables trapping of communication for the specified port.

You must execute a COM(< n >) ON statement to activate this statement for port < n > . After COM(< n >) ON, if you specify a non-zero line number in the ON COM(< n >) statement, then every time GW-BASIC starts a new statement it checks to see if any characters have come in to the specified port. If so, the program performs a GOSUB to the specified line number.

If COM(< n >) OFF was executed, no trapping takes place for the port and the event is not remembered even if it does take place.

If GW-BASIC executes a COM(< n >) STOP statement, no trapping can take place for the port. But if a character is received, GW-BASIC remembers this character and an immediate trap takes place when COM(< n >) ON is next executed. You can use 0 in place of a 1 or 2 to affect both ports in the COM(< n >) ON, COM(< n >) STOP, COM(< n >) OFF.

When the trap occurs, an automatic COM(< n >) STOP is executed, so that recursive traps can never take place. The RETURN from the trap routine automatically executes a COM(< n >) ON unless there is an explicit COM(< n >) OFF inside the trap routine.

Event trapping does not take place when GW-BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place, all trapping is automatically disabled (including ERROR, COM and KEY).

Typically the communications trap routine reads an entire message from the communications port before returning back to the main program. You should not use the communications trap for single-character messages. The interrupt buffer is fixed at 256K bytes, and at high baud rates the overhead of trapping and reading for each individual character might overflow the buffer.

EXAMPLE:

```
100 PORT. A = 1
110 PORT. B = 2
120 ON COM(PORT. A) GOSUB 500
        .
        .
        .
500 '*****    ROUTINE TO HANDLE PORT A CHRS
        .
        .
        .
550 RETURN
```

FORMAT:

ON KEY(< n >) GOSUB < line number >

PURPOSE:

Sets up a line number for GW-BASIC to trap when you press the specified function key or cursor control key.

REMARKS:

< n > is a numeric expression returning a value between 1 and 14 and indicates the key to be trapped.

1–7	Function keys 1 to 7
8	Up arrow
9	Left arrow
10	Right arrow
11	Down arrow

< line number > is a valid number in the range 0 to 65529. Use 0 as < line number > to disable trapping on the specified key.

You must execute a KEY(< n >) ON statement to activate trapping of function key or cursor control key activity. After KEY(< n >) ON, if a non-zero line number is specified in the ON KEY(< n >) statement, then every time GW-BASIC starts a new statement it checks to see if the specified key was pressed. If so, GW-BASIC performs a GOSUB to the line number specified in the ON KEY(< n >) statement.

If you execute a KEY(< n >) OFF statement, no trapping takes place for the specified key and the event is not remembered even if it does take place.

If you enter a KEY(< n >) STOP statement, no trapping takes place. But if you press the specified key, the program remembers so an immediate trap takes place at the next KEY(< n >) ON.

When the trap occurs an automatic KEY(< n >) STOP is executed, so that recursive traps can never take place. The RETURN from the trap routine automatically executes a KEY(< n >) ON unless you have an explicit KEY(< n >) OFF inside the trap routine.

Event trapping does not take place when GW-BASIC is not executing a program. When an error trap (resulting from an ON ERROR statement) takes place it automatically disables all trapping (including ERROR, COM, and KEY).

Key trapping might not work when you press other keys before the specified key. The key that caused the trap cannot be tested using INPUT$ or INKEY$, so the trap routine for each key must be different if you want a different function.

KEY(< n >) ON has no effect on whether the soft key values display on the 25th line.

EXAMPLE:

```
100 KEY.5 = 5
110 ON KEY(KEY.5) GOSUB 500
      .
      .
      .
500 '*****    ROUTINE TO HANDLE KEY(5)
      .
      .
      .
550 RETURN
```

FORMAT:

OPEN [<dev>] <filespec> [FOR <mode>] AS [#]
<file number> [LEN = <lrecl>]

PURPOSE:

Establishes addressability between a physical device and an I/O buffer in the data pool.

REMARKS:

<dev> is optionally part of the filename string.

<filespec> is a valid string literal or variable, as described in Chapter 1.3. In the simplest case, it is just a filename. The device can be specified separately in the <dev> field or as part of the <filespec>, or omitted entirely (in which case the default drive is assumed).

<mode> determines the initial positioning within the file and the action to be taken if the file does not exist. The valid modes and actions taken are:

- INPUT: Position to the beginning of an existing file. GW-BASIC returns the "File Not Found" error if the file does not exist.
- OUTPUT: Position to the beginning of the file. If the file does not exist, the program creates one.
- APPEND: Position to the end of the file. If the file does not exist, the program creates one.

If you omit the FOR <mode> clause, the initial position is at the beginning of the file. If GW-BASIC cannot find the file, it creates one. This feature is the Random I/O mode: you can read or write records at will from any position within the file.

< file number > is an integer expression returning a number in the range 1 through 15. GW-BASIC uses this number to associate an I/O buffer with a disk file or device. This association exists until the program executes a CLOSE < file number > or CLOSE statement.

< lrecl > is an integer expression in the range 2 to 32768. This value sets the record length to be used for random files. If you omit < lrecl >, the record length defaults to 128-byte records.

Your MS-BASIC manual has an alternate form of the OPEN statement.

When you OPEN FOR APPEND a disk file, the position is initially at the end of the file and the record number is set to the last record of the file (LOF(x)/128). PRINT, WRITE, or PUT then extend the file. You can position the program elsewhere in the file with a GET statement. If you do so, the mode changes to random and the position moves to the record indicated.

Once you move the position from the end of the file, you can execute a GET #x,LOF(x)/ < lrecl > to append additional records to the file.

3.27 OPENING A COM FILE

FORMAT:

OPEN < **COM.filename** > **AS** [#] < **file number** >

PURPOSE:

Allocates a buffer for I/O in the same fashion as OPEN does for disk files.

REMARKS:

< COM.filename > = "< dev >:< speed >,< parity >,< data >, < stop >,[RS],[,CS < n >][,DS < n >][,CD < n >][,LF][,ASC or,BIN]"

< dev > is a valid communications device. Valid devices are:

COM1: COM2:

< speed > is a literal integer specifying the transmit/receive baud rate. Valid speeds are:

50, 75, 110, 150, 200, 300, 600, 1200, 1800,
2000, 2400, 3600, 4800, 9600, 19200

If you omit the speed parameter, GW-BASIC uses the default 300.

< parity > is a one-character literal specifying the parity for transmit and receive as follows:

S	SPACE	Parity bit always transmitted and received as space (0 bit).
O	ODD	Odd transmit/receive parity checking.
M	MARK	Parity bit always transmitted and received as mark (1 bit).
E	EVEN	Even transmit/receive parity checking.
N	NONE	No transmit parity, no receive parity checking.

If you omit the parity parameter, GW-BASIC uses the default even.

< data > is a literal integer indicating the number of transmit/receive data bits. Valid values are: 5, 6, 7, 8.

If you omit the data parameter, GW-BASIC uses the default 7.

Note: 5 data bits with no parity is illegal. 8 data bits with any parity is also illegal.

< stop > is a literal integer indicating the number of stop bits. Valid values are: 1, 2.

If you omit < stop >, the program uses 2 stop bits at 75 and 110 baud; otherwise, the program uses 1 stop bit.

< file number > is an integer expression returning a valid file number. The number is then associated with the file for as long as it is OPEN and is used to refer other COM I/O statements to the file.

[RS] suppresses RTS (Request to Send). The RTS line is normally turned on when you execute Open a COM File.

[CS < n >] controls CTS (Clear To Send). If you omit CS, the default is CS1000.

If you specify RS and omit CS, CS defaults to 0.

[DS < n >] controls DSR (Data Set Ready). If you omit DS, the default is DS1000.

[CD < n >] controls CD (Carrier Detect, also referred to as RLSD, Received Line Signal Detector). If you omit CD, the default is CS0.

Normally I/O statements to a communication file fail if the CTS or DSR lines are not cabled. The CS and DS options let you avoid this problem by ignoring these lines. If you include the < n > argument, it specifies the number of milliseconds to wait for the signal before returning a "Device Timeout" error.

If you omit the argument < n > in the CS, DS, and CD options, or if you set them to 0, the program does not check that line's status. n must be greater than or equal to 0, but less than or equal to 65535.

[LF] causes a line feed character after a carriage return.

[ASC] opens a COM file in ASCII mode.

[BIN] opens a COM file in binary mode.

The speed, parity, data, and stop parameters are positional, but RS, CS, DS, and CD can appear in any order after STOP.

Note: A COM device may be OPENed to only one file number at a time.

POSSIBLE ERRORS:

Any coding errors within the filename string result in the "Bad File Name" error. GW-BASIC does not give any indication which parameter is wrong. The "Device Timeout" error occurs if data set ready (DSR) is not detected. Refer to your hardware documentation for proper cabling instructions.

EXAMPLES:


```
10 OPEN "COM1: " AS 1
```

File 1 is opened for communication with all defaults. Speed at 300 bps, even parity, and 7 data bits, one stop bit.

```
20 OPEN "COM1:2400 " AS #2
```

File 2 is opened for communication at 2400 bps. Parity and number of data bits are defaulted.

```
10 OPEN "COM1:1200,N,8" AS #1
```

File number 1 is opened for Asynchronous I/O at 1200 bits/second, no parity is to be produced or checked, and 8-bit bytes are sent and received.

3.28 OUT

FORMAT:

OUT <**port**>,<**data**>

PURPOSE:

Sends a byte to a machine output port.

REMARKS:

<port> is a numeric expression returning a value in the integer range from 0 to 65535.

<data> is a numeric expression returning a value in the integer range from 0 to 255.

In assembly language, the statement is equivalent to:

```
MOV     DX, port        ;port address
MOV     AL, data        ;byte to output
OUT     DX,AL           ;output byte in AL to port
                        ;[DX]
```

Note: Since your computer uses memory mapped I/O, this statement returns the port number.

FORMAT:

PAINT (< xstart > , < ystart >)[, < paint attribute > [, < border attribute >]]

PURPOSE:

Fills in an area on the screen with the selected color.

REMARKS:

< xstart > and < ystart > are numeric expressions that return a number in the integer range and specify screen coordinates for the origin of painting.

< paint attribute > is a numeric expression that returns a value from 0 to 3. It determines the color with which to fill the area. See Chapter 1.1 for more information.

< border attribute > is a numeric expression that returns a value from 0 to 3 and specifies the color to be searched for to delimit the area being painted.

The PAINT statement fills in an arbitrary graphics figure with the specified paint attribute. If you omit the paint attribute, it defaults to the foreground attribute (3 or 1) and the border attribute defaults to the paint attribute.

You might, for example, want to fill in a circle of attribute 3 with attribute 0. Visually, this could mean a black ball with a white border.

Since there are only two attributes in Hi-res mode, you must white out an area until white is hit, or black out an area until black is hit.

PAINT must start on a non-border point or it has no effect.

PAINT can fill any figure, but PAINTing "jagged" edges or very complex figures might result in the "Out of Memory" error. If this happens, you must use the CLEAR statement to increase the amount of stack space available.

EXAMPLE:

```
10 SCREEN 2
20 LINE (100,200)-(200,350),1,B
30 PAINT (150,225),1,1
```


3.30 PLAY

FORMAT:

PLAY <**string exp**>

PURPOSE:

Plays music as specified by <string exp>.

REMARKS:

<string exp> is a string expression returning a valid string conforming to the format described below.

PLAY implements a concept similar to DRAW by embedding a Music Macro Language into the string data type.

The single-character commands in PLAY are described in Table 3-2.

Table 3-2: PLAY Commands

COMMAND	DESCRIPTION
A–G [#, +, −]	Play the note. "#" or " + " following the note means sharp, and " − " means flat.
L <n>	Length—sets the length of each note. L4 is a quarter note, L1 is a whole note, and so on. n ranges from 1 to 64.
	The length can also follow the note when you want to change the length for only one note. In this case, A16 is equivalent to L16A.
MF	Music Foreground. Music (PLAY statement) and SOUND are to run in foreground. That is, each subsequent note or sound does not start until the previous note or sound is finished.
MB	Music Background. In this version of GW-BASIC, the effect of MB duplicates that of MF.
MN	Music Normal. Each note plays 7/8ths of the time determined by L (length).
ML	Music Legato. Each note plays the full period set by L (length).
MS	Music Staccato. Each note plays 3/4ths of the time determined by L (length).
N <n>	Play note n. n can range from zero to 84. In the seven possible octaves, there are 84 notes. N = 0 means rest.
O <n>	Octave. Sets the current octave. There are seven octaves (0–6).
P <n>	Pause. P ranges from 1 to 64.
T <n>	Tempo. Sets the number of L4's in a second. n ranges from 32 to 255. Default is 120.
	Dot. A dot after a note causes the note to play 3/2 times the period determined by L (length) times T (tempo). Multiple dots can appear after the note. The period is scaled accordingly (for example, A. 3/2, A. . 9/4, A. . . 27/8). Dots can appear after a pause (P) and scale the pause length as described.
X <string>	Execute substring.

Note: Because of the slow clock interrupt rate, some notes do not play at higher tempos—e.g., L64 at T255. You can determine these note/tempo combinations by experimenting.

EXAMPLE:

```
10 A$ = "BB-C"
20 B$ = "O4XA$;"
30 C$ = "L1CT50N3N4N5N6"
40 PLAY "P2XA$;XB$;XC$;"
```

3.31 PSET

FORMAT:

PSET (<absolute x>,<absolute y>) [,<attribute>]

PSET STEP (<x offset>,<y offset>) [,<attribute>]

PURPOSE:

Displays ("turns on") or removes ("turns off") one pixel (dot) from the video screen.

REMARKS:

<absolute x>, <absolute y>, <x offset>, and <y offset> are valid coordinates. See Chapter 1.1 for further information.

<attribute> is an expression returning a value 0 to 3. This value determines the color of the point. Even values add a dot and odd values remove a dot.

If the attribute argument is omitted, GW-BASIC uses the default 1 in screen mode 2.

EXAMPLE:

```
10 FOR I = 0 TO 100
20 PSET (I,I),1
30 NEXT '(draw a diagonal line to (100,100))
40 FOR I = 100 TO 0 STEP -1
50 PSET (I,I),0
60 NEXT '(remove the line just drawn)
```

PRESET 3.32

FORMAT:

PRESET (<absolute x>,<absolute y>) [,<attribute>]

PRESET STEP (<x offset>,<y offset>) [,<attribute>]

PURPOSE:

Removes ("turns off") or displays ("turns on") one pixel (dot) from the video screen.

REMARKS:

PRESET has a syntax identical to PSET. The only difference is that if you give no third parameter for the color, GW-BASIC selects zero. When you give a third argument, PRESET is identical to PSET.

If you give an out-of-range coordinate to PSET or PRESET, GW-BASIC takes no action and displays no error message. If you give an attribute greater than 3, you see the "Illegal Function Call" message. The program treats attribute value 2 like 0 in Hi-res and treats 3 like 1.

EXAMPLE:

```
10 FOR I = 0 TO 100
20     PSET (I,I)
30 NEXT   'draw a diagonal line to (100,100)
40 FOR I = 100 TO 0 STEP -1
50 PRESET (I,I),0
60 NEXT   'remove the line just drawn
```

3.33 RETURN

FORMAT:

RETURN [<line number>]

PURPOSE:

Returns from a GOSUB.

REMARKS:

See your MS-BASIC manual for details on RETURN.

The line number is primarily intended for use with event trapping. The event trap routine might want to go back into GW-BASIC at a fixed line number while still eliminating the GOSUB entry that the trap created.

Use the non-local return with care. Any other GOSUB, WHILE, or FOR that was active at the time of the trap remains active. If the trap comes out of a subroutine, any attempt to continue loops outside the subroutine results in the "NEXT without FOR" error.

EXAMPLE:

```
100 RETURN 900
```

SAVE 3.34

FORMAT:

SAVE "<filespec>" [,A,P]

PURPOSE:

Saves a GW-BASIC program file on diskette or another device.

REMARKS:

<filespec> conforms to the rules described in Chapter 1.3.

Refer to your MS-BASIC manual for normal syntax. The only difference in syntax is that the device specifications other than the diskette are legal.

EXAMPLE:

```
SAVE "COM1:4800,0,7,1"A
```

SCREEN 3.35

Note: This is the SCREEN statement. See Chapter 4.6 for the SCREEN function.

FORMAT:

SCREEN [<mode>] [, [<burst>] [, [<apage>][, <vpage>]]]

PURPOSE:

Sets the screen attributes.

REMARKS:

< mode > is a valid numeric expression returning an unsigned integer value. Valid modes are:

- 0—Alpha mode at current width (40 or 80).
- 2—Hi-resolution Graphics mode (800 × 400 dots).

< burst > is a valid numeric expression returning a boolean result. This parameter enables color burst. A value of 0 disables color burst (black and white images only). A non-zero value enables color burst (allows color images). Your computer accepts only zero.

< apage > —Active page. This parameter is valid in Alpha only. It is a numeric expression returning an unsigned integer in the range 0 to 7 for width 40, or 0 to 3 for width 80. < apage > selects the page to be written to. Your computer accepts only zero.

< vpage > —Visible page. Follows the same rules as < apage >, but selects which page displays on the screen. Your computer accepts only zero.

If all parameters are legal and there is a change in < mode > or < burst > from their previous values, the screen clears. The background color is reset to black, and the foreground color to white.

RULES:

1. Any values entered outside these ranges result in the "Illegal Function Call" error. Previous values are retained.
2. You can omit any parameter. Omitted parameters assume the old value.

EXAMPLE:

```
10 SCREEN 0,0,0,0     'select alpha mode
10 SCREEN 2           'select hi-res graphics
```

SOUND 3.36

FORMAT:

SOUND < **frequency** > , < **duration** >

PURPOSE:

Generates a sound of a specified frequency for a specified time duration from the speaker.

REMARKS:

< frequency > is the desired frequency in Hertz. It is a valid numeric expression returning an unsigned integer in the range 37 to 32767.

< duration > is the desired duration in clock ticks. It is a valid numeric expression returning an unsigned integer in the range 0 to 65535.

Current clock ticks occur 18.2 times per second. If the duration is zero, any current sound that is playing turns off; you see no effect if no sound is playing.

EXAMPLE:

```
2500 SOUND RND*1000+37,2   'creates random
                           'sound.
```

TIME$ 3.37

FORMAT:

TIME$ = < **string expr** >

To set the current time.

< **string expr** > = **TIME$**

To get the current time.

PURPOSE:

Sets or retrieves the current time.

REMARKS:

< string exp > is a valid string literal or variable.

The current time is fetched and assigned to the string variable if TIME$ is the expression in a LET or PRINT statement.

The current time is stored if TIME$ is the target of a string assignment.

RULES:

1. If < string expr > is not a valid string, the "Type mismatch" error results.
2. For < string var > = TIME$, TIME$ returns an 8-character string in the form "hh:mm:ss", where hh is the hour (00 to 23), mm is the minutes (00 to 59), and ss is the seconds (00 to 59).
3. For TIME$ = < string expr >, < string expr > can take one of the following forms:
   a. "hh" sets the hour. Minutes and seconds default to 00.
   b. "hh:mm" sets the hour and minutes. Seconds default to 00.
   c. "hh:mm:ss" sets the hour, minutes and seconds.

If any of the values are out of range, the statement returns the "Illegal Function Call" error. The previous time is retained.

EXAMPLE:

```
TIME$ = "08:00"
Ok
PRINT TIME$
08:00:04
Ok
```

The following program displays the current date and time on the 25th line of the screen, and chimes on the half hour and hour.

```
10 KEY OFF:SCREEN 0:WIDTH 40:CLS
20 LOCATE 25,5
30 PRINT DATE$,,TIME$
40 SEC = VAL(MID$(TIME$,7,2))
50 IF SEC = SSEC THEN 20 ELSE SSEC = SEC
60 IF SEC = 0 THEN 1010
70 IF SEC = 30 THEN 1020
80 IF SEC < 57 THEN 20
1000 SOUND 1000,2:GOTO 20
1010 SOUND 2000,8:GOTO 20
1020 SOUND 400,4 :GOTO 20
```


WAIT 3.38

FORMAT:

WAIT <**port**>,<**and byte**> [,<**xor byte**>]

PURPOSE:

Suspends program execution while monitoring the status of a machine port.

REMARKS:

<port> is a numeric expression returning a number in the integer range 0 to 65535.

<and byte> is a numeric expression returning a number in the integer range from 0 to 255 and is used to match a byte coming in from the <port>.

< xor byte > is a numeric expression returning a number in the integer range from 0 to 255 and is used to check a byte coming in from the < port > .

Note: Because your computer uses memory mapped I/O, this statement stops execution of a program indefinitely. ALT-C aborts the statement.

3.39 WIDTH

FORMAT:

WIDTH < **size** >

WIDTH < **file no.** > , < **size** >

WIDTH < **dev** > , < **size** >

PURPOSE:

Sets the printed line width in number of characters for the screen and line printer.

REMARKS:

< size > is a valid numeric expression returning an integer result in the range 0 to 255. This integer is the new width.

< file no. > is a valid numeric expression returning an integer in the range 1 to 4. This integer is the number of the file OPENed.

< dev > is a valid string expression returning the device identifier. Valid devices are described in Chapter 1.3.

Depending upon the device specified, the following actions are possible:

WIDTH < **size** >
WIDTH "**SCRN:**", < **size** >

These commands set the screen width. You can use 40- or 80-column width only.

Note: Changing the screen width clears the screen. Screen mode 0 can be 40 or 80 columns. Screen mode 2 is always 80 columns.

WIDTH "**LPT1:**", < **size** >

Use this form as a deferred width assignment for the line printer. This form of WIDTH stores the new width value without changing the current width setting. A subsequent OPEN "LPT1:" FOR OUTPUT AS < number > uses this value for width while the file is open.

WIDTH < **file no.** >, < **size** >

If the file is open to LPT1:, the line printer's width is immediately changed to the new size specified. This allows you to change the width while the file is open. This form of WIDTH has meaning only for LPT1:.

RULES:

1. Valid widths for the screen are 40 and 80. Valid width for the line printer is 1 to 255.

   Any value entered outside these ranges results in the "Illegal Function Call" error. The previous value is retained.

2. WIDTH has no effect on the keyboard (KYBD:).

3. WIDTH for printers does not check against the physical printer's valid width range. A value from 1 to 255 is valid for all physical printers.

4. Specifying WIDTH 255 for the line printer (LPT1:) disables line folding; the effect is infinite width.

EXAMPLE:

```
10 WIDTH "LPT1:",75
20 OPEN "LPT1:" FOR OUTPUT AS #1
        .
        .
        .
6020 width #1,40
```

In this example, line 10 stores a line printer width of 75 characters per line.

Line 20 opens file #1 to the line printer and sets the width to 75 for subsequent PRINT #1,... statements.

Line 6020 changes the current line printer width to 40 characters per line.

4

GW-BASIC FUNCTIONS

Table 4-1 is a summary of GW-BASIC functions. Each function is explained in detail in the sections that follow.

Table 4-1: GW-BASIC Functions

FUNCTION	PURPOSE
CSRLIN	Returns current line position of cursor
INP	Returns byte read from port
INPUT$ for COM Files	Returns a string of characters read from the keyboard or from a file
LOF	Returns number of bytes allocated to the file
POINT	Returns the attribute of pixel at specified coordinates
SCREEN	Returns the ordinal of character at specified row and column
VARPTR	Returns the first byte of the file control block for the opened file

CSRLIN 4.1

FORMAT:

```
x = CSRLIN
```

PURPOSE:

Returns the current line (or row) position of the cursor.

REMARKS:

The value returned is in the range from 1 to 25.

x = POS(0) returns the column location of the cursor.

The LOCATE statement explains how to set the cursor line.

EXAMPLE:

```
10 ROW = CSRLIN      'Record current line.
20 COL = POS(0)      'Record current column.
30 LOCATE 24,1
40 PRINT "HELLO"     'Print HELLO on last line
50 LOCATE ROW,COL    'Restore pos. to old line,
                     'column
```


4.2 INP

FORMAT:

X = INP(<n>)

PURPOSE:

Returns the byte read from port <n>.

REMARKS:

INP is the complementary function to the OUT statement. This function translates to the 8086 assembly language statements:

```
MOV         DX, n          ;n = the port number
IN          AL, DX
```

Note: Because your computer uses memory mapped I/O, this function returns the port number.

INPUT$ FOR COM FILES 4.3

FORMAT:

x$ = INPUT$(< num chars > [,[#] < file number >])

PURPOSE:

Returns a string of < n > characters, read from the keyboard (default) or from the < file number > .

REMARKS:

< num chars > is the number of characters to be input from the file.

< file number > is the number of a previously opened file for input.

The INPUT$ function is better than the INPUT and LINE INPUT statements when you are reading COM files, since all ASCII characters might be significant in communications. INPUT stops at commas or carriage returns and LINE INPUT terminates at carriage returns.

INPUT$ allows all characters read to be assigned to a string. If you are using the terminal for input, the characters do not echo on the console. The function reads all characters except ALT-C. ALT-C interrupts the execution of the INPUT$ function.

The following statements are most efficient for reading a COM file:

```
10 WHILE NOT EOF (1)
20 A$=INPUT$(LOC (1),#1)
     .
     .
Process data returned in A$
     .
     .
60 WEND
```

This sequence means: "while there is something in the input queue, return the number of characters in the queue and store them in A$. If there are more than 255 characters, only 255 are returned at a time to prevent string overflow. If this is the case, EOF(1) is false and input continues until the input queue is empty."

4.4 LOF

FORMAT:

LOF(< **file number** > **)**

PURPOSE:

Returns the number of bytes allocated to the file.

REMARKS:

< file number > is associated with a currently open file.

For diskette files, LOF returns a multiple of 128. For example, if the actual file length is 257 bytes, the number 384 is returned.

For communications, LOF returns the amount of free space in the input buffer. That is, size-LOC(filnum), where size is the size of the communications buffer, defaults to 256.

EXAMPLE:

```
10 OPEN "DATA.FIL" AS #1
20 GET #1, LOF(1)/128
```

These statements get the last record of the file, assuming the record length is 128 bytes.

POINT 4.5

FORMAT:

x = POINT (<absolute x>,<absolute y>)

PURPOSE:

Returns the attribute of the pixel at the specified coordinates. <absolute x> and <absolute y> are valid expressions returning a value in the screen range.

EXAMPLE:

```
10 FOR I = 1 TO 400
20 IF POINT(I,I) <> 0 THEN GOTO 50 'a dot ?
30 PSET(I,I) 'put a dot if not one here
40 GOTO 60
50 PRESET(I,I)     'remove a dot if one here
60 NEXT I
```


SCREEN 4.6

Note: This is the SCREEN function. See Chapter 3.35 for the SCREEN statement.

FORMAT:

x = SCREEN(<row>,<col> [,<boolean>])

PURPOSE:

Returns the ordinal of the character from the screen at the specified row (line) and column.

REMARKS:

x is a numeric variable receiving the ordinal returned.

< row > is a valid numeric expression returning an unsigned integer in the range 1 to 25.

< col > is a valid numeric expression returning an unsigned integer in the range 1 to 40 or 1 to 80 depending upon the width.

< boolean > is a valid numeric expression returning a boolean result.

The ordinal of the character at the specified coordinates is stored in the numeric variable. If you give a non-zero optional parameter < boolean >, the color attribute for the character returns instead.

RULE:

Any values entered outside these ranges result in the "Illegal Function Call" error.

EXAMPLE:

```
100 X = SCREEN (10,10)     'Returns 65 if
                           'character at 10,10 is 'A'.
110 X = SCREEN (1,1,1)     'Returns color
                           'attribute of character in
                           'the upper left corner of
                           'screen.
```

4.7 VARPTR

FORMAT:

x = VARPTR(< file number >)

PURPOSE:

Returns the address of the first byte of the file control block (FCB) for the opened file.

REMARKS:

< file number > is tied to a currently open file. Offsets to information in the FCB from the address returned by VARPTR are described in Table 4-2.

Table 4-2: Offsets to FCB Addresses

OFF	SIZE	CONTENTS	MEANING
0	1	Mode	The mode in which the file was opened: 1 Input only 2 Output only 4 Random I/O 16 Append only 32 Internal use 64 Future use 128 Internal use
1	38	FCB	Disk File Control Block. Refer to the *MS-DOS User's Guide* for contents.
39	2	CURLOC	Number of sectors read or written for sequential access. For random access, it contains the last record number + 1 read or written.
41	1	ORNOFS	Number of bytes in sector when read or written.
42	1	NMLOFS	Number of bytes left in input buffer.
43	3	***	Reserved for future use.
46	1	DEVICE	Device number: 0–9 Disks A: thru J: 255 KYBD: 254 SCRN: 253 LPT1: 251 COM1: 250 COM2: 249 LPT2: 248 LPT3:
47	1	WIDTH	Device width.

OFF	SIZE	CONTENTS	MEANING
48	1	POS	Position in buffer for PRINT.
49	1	FLAGS	Internal use during LOAD/SAVE. Not used for data files.
50	1	OUTPOS	Output position used during tab expansion.
51	128	BUFFER	Physical data buffer. Used to transfer data between DOS and GW-BASIC. Use this offset to examine data in sequential I/O mode.
179	2	VRECL	Variable-length record size. Default is 128. Set by length option in OPEN statement.
181	2	PHYREC	Current physical record number.
183	2	LOGREC	Current logical record number.
185	1	***	Reserved for future use.
186	2	OUTPOS	Disk file only. Output position for PRINT, INPUT, and WRITE.
188	<n>	FIELD	Actual FIELD data buffer. Size is determined by /S: switch. VRECL bytes are transferred between BUFFER and FIELD on I/O operations. Use this offset to examine file data in Random I/O mode.

EXAMPLE:

```
10 OPEN "DATA.FIL" AS #1
20 FCBADR = VARPTR(#1) 'FCBADR contains start of FCB
30 DATADR = FCBADR+188 'DATAR contains address
40 'of data buffer.
50 A$ = PEEK (DATADR) 'A$ contains 1st byte
60 'in data buffer.
```

5

THE COMMUNICATION OPTION

This chapter describes the GW-BASIC statements required to support RS-232-C asynchronous communications with other computers and peripherals.

COMMUNICATION I/O 5.1

Since the communications port is opened as a file, all Input/Output statements that are valid for disk files are valid for COM.

COM sequential input statements are the same as those for disk files:

```
INPUT #<file number>
LINE INPUT #<file number>
INPUT$
```

COM sequential output statements are the same as those for disk:

```
PRINT #<file number>
PRINT #<file number> USING
```

Refer to the "INPUT" and "PRINT" statements for details of coding syntax and usage.

GET and PUT are only slightly different for COM files. See "GET and PUT Statements for COM Files."

The TTY program that follows enables your (GW-BASIC) computer to be used as a conventional terminal. Besides full-duplex communication with a host, the TTY program allows data to be downloaded to a file. Conversely, a file can be "uploaded" (transmitted) to another machine.

In addition to demonstrating the elements of asynchronous communications, this program is useful in transferring GW-BASIC programs and data to and from your computer.

Note: The TTY program is set up to communicate using XON and XOFF. You might want to modify it for your environment.

5.2 THE TTY PROGRAM

(An Exercise in Communication I/O)

```
10 SCREEN 0,0:WIDTH 80
15 KEY OFF:CLS:CLOSE
20 DEFINT A-Z
25 LOCATE 25,1
30 PRINT STRING$(60," ")
40 FALSE=0:TRUE= NOT FALSE
50 MENU=5 ' Value of MENU key (ctrl-E)
60 XOFF$=CHR$(19):XON$=CHR$(17)

100 LOCATE 25,1:PRINT "Async TTY Program ";
110 LOCATE 1,1:LINE INPUT "Speed? ";SPEED$
120 COMFIL$="COM1:"+SPEED$+",E,7"
130 OPEN COMFIL$ AS #1
140 OPEN "OUTPUT", #2, SCRN:"

200 PAUSE=FALSE
210 A$=INKEY$: IF A$="" THEN 230
220 IF ASC(A$)=MENU THEN 300 ELSE PRINT #1,A$;
230 IF EOF(1) THEN 210
240 IF LOC(1)>128 THEN PAUSE=TRUE: PRINT #1,XOFF$;
250 A$=INPUT$(LOC(1),#1)
260 PRINT #2,A$;:IF LOC(1)>0 THEN 240
270 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
280 GOTO 210

300 LOCATE 1,1:PRINT STRING$(30," "):LOCATE 1,1
310 LINE INPUT"File? ";DSKFIL$
```

```
400 LOCATE 1,1:PRINT STRING$(30," "):LOCATE 1,1
410 LINE INPUT"(T)ransmit or (R)eceive? ";TXRX$
415 CLOSE #2
420 IF TXRX$="T" THEN OPEN DSKFIL$ FOR INPUT
    AS #2:GOTO 1000
430 OPEN "OUTPUT", #2, DSKFIL$
440 PRINT #1,CHR$(13);

500 IF EOF(1) THEN GOSUB 600
510 IF LOC(1)>128 THEN PAUSE=TRUE: PRINT #1,XOFF$;
520 A$=INPUT$(LOC(1),#1)
530 PRINT #2,A$;:IF LOC(1)>0 THEN 510
540 IF PAUSE THEN PAUSE=FALSE:PRINT #1,XON$;
550 GOTO 500

600 FOR I=1 TO 5000
610 IF NOT EOF(1) THEN I=9999
620 NEXT I
630 IF I>9999 THEN RETURN
640 CLOSE #2:CLS:LOCATE 25,10:PRINT
    "* Download complete *";
650 END

1000 WHILE NOT EOF(2)
1010 A$=INPUT$(1,#2)
1020 PRINT #1,A$;
1030 WEND
1040 PRINT #1,CHR$(26); 'CTRL-Z to make close file.
1050 CLOSE #2:CLS:LOCATE 25,10:PRINT
     "** Upload complete **";
1060 END

9999 CLOSE:KEY OFF
```

5.3 NOTES ON THE TTY PROGRAM

LINE NO.	COMMENTS
10	Sets the screen to Alpha mode; sets the width to 80.
15	Turns off the soft key display, clears the screen, and makes sure that all files are closed. **Note:** Asynchronous implies character I/O as opposed to line or block I/O. Therefore, all PRINTs (either to the COM file or to the screen) are terminated with a semicolon (;). This retards the carriage return/line feed normally issued at the end of a PRINT statement.
20	Defines all numeric variables as INTEGER. Primarily for the subroutine at 600–620.
25–30	Clears the 25th line starting at column 1.
40	Defines boolean TRUE and FALSE.
50	Defines the ASCII (ASC) value of the MENU key.
60	Defines the ASCII XON, XOFF characters.
100–130	Prints program-id and asks for baud rate (speed). Opens communications to file number 1, even parity, 7 data bits. Programmer exercise: Modify this section to check for valid baud rates.
200–280	This section performs full-duplex I/O between the screen and the device connected to the RS-232-C connector, as follows: 1. Read a character from the keyboard into A$. Note that INKEY$ returns a null string if no character is waiting. 2. If no character is waiting then go see if any characters are being received. If a character is waiting at the keyboard then: 3. If the character is the MENU key, then the user is ready to download a file, so get the filename. 4. If character (A$) is not the MENU key then send it by writing to the communication file (PRINT #1...). 5. At 230 see if any characters are waiting in COM buffer. If not, then go back and check keyboard. 6. At 240, if more than 128 characters are waiting then set PAUSE flag saying we are suspending input. Send XOFF to host, stopping further transmission.

LINE NO.	COMMENTS
	7. At 250–260, read and display contents of COM buffer on screen until empty. Continue to monitor size of COM buffer (in 240). Suspend transmission if we fall behind.
	8. Finally, resume host transmission by sending XON only if suspended by previous XOFF. Repeat process until MENU key is struck.
300–320	Get disk filename we are downloading to. Open the file as device number 2.
400–420	Asks if file named is to be transmitted uploaded) or received (downloaded).
430	Sends a carriage return to the host to begin the download. This program assumes that the last command sent to the host was to begin such a transfer and was missing only the terminating carriage return. If a DEC system is the host, then such a command might be: **COPY TTY: = MANUAL.MEM < MENU key >** where the MENU key was struck instead of Return.
500	When you are receiving no more characters (LOC(x) returns 0), then perform a time-out routine (explained later).
510	Again, if more than 128 characters are waiting, signal a pause and send XOFF to the host while we catch up.
520–530	Read all characters in COM queue LOC(x)) and write them to disk (PRINT #2..) until we are caught up.
540–550	If a pause was issued, restart host by sending XON and clear the pause flag. Continue process until no characters are received for a pre-determined time.
600–650	This is the time-out subroutine. The FOR loop count was determined by experimentation. In short, if no character is received from the host for 17–20 seconds, then transmission is assumed complete. If any character is received during this time (line 610) then set I well above FOR loop range to exit loop and then return to caller. If host transmission is complete, close the disk file and return to being a terminal.
1000–1060	Transmit routine. Until end of disk file do: Read one character into A$ with INPUT$ statement. Send character to COM device in 1020. Send a Control-Z at end of file in 1040 in case receiving device needs one to close its file. Finally, in lines 1050 and 1060, close our disk file, print completion message and go back to conversation mode in line 200.
9999	Presently not executed. As an exercise, add some lines to the routine 400–420 to exit the program via line 9999. This line closes the COM file left open and restores the soft key display.

5.4 THE COM I/O FUNCTIONS

The most difficult aspect of asynchronous communications is processing characters as fast as they are received. At rates above 2400 bps, character transmission must be suspended from the host long enough to catch up. This suspension can be done by sending XOFF (ALT-S) to the host and XON (ALT-Q) when ready to resume.

GW-BASIC provides three functions which help in determining when an "over-run" condition is imminent:

- **LOC(x):** Returns the number of characters in the input queue waiting to be read. The input queue can hold only 255 characters.
- **LOF(x):** Returns the amount of free space in the input queue—that is, 256—LOC(x). You can use LOF to detect when the input queue is getting full. LOC is adequate for this purpose, as shown in the programming example.
- **LOF(x):** If true (− 1), indicates that the input queue is empty. Returns false (0) if any characters are waiting to be read.

The following errors are possible:

1. "Communication Buffer Overflow" occurs if a read is attempted after the input queue is full (i.e., LOC(x) returns 0).
2. "Device I/O Error" occurs if any of the following line conditions are detected on receive: Overrun Error (OE), Framing Error (FE), or Break Interrupt (BI). The error is reset by subsequent inputs but the character causing the error is lost.
3. "Device Fault" occurs if data set ready (DSR) is lost during I/O.

6

GW-BASIC INITIALIZATION AND PRINTER CONFIGURATION

GW-BASIC INITIALIZATION 6.1

FORMAT:

GW-BASIC [<filename>] [/F:<number of files>] [/S:<lrecl>] <buffer size>] [/M:<highest memory location>]

REMARKS:

GW-BASIC is loaded and executed by typing:

GW-BASIC

at the MS-DOS command line prompt.

Upon loading, GW-BASIC responds with the banner:

```
GW-BASIC Rev. x.x
LCOPY Print to xxxxxxxxx
[MS-DOS Version]
Created: DD-Month-YY
(c) Microsoft 1982
nnnn Bytes Free
Ok
```

You can alter the GW-BASIC operating environment somewhat by specifying option switches following GW-BASIC on the command line. <filename> is the filename of a GW-BASIC program. If <filename> is present, GW-BASIC proceeds as if a RUN "<filename>" command had been given after initialization.

GW-BASIC assumes a default file extension of .BAS if you give no other. This default allows GW-BASIC to run programs in batch by putting this form of the command line in an AUTOEXEC.BAT file. Programs run in this manner must exit via the SYSTEM command so that the next command in the AUTOEXEC.BAT file can be executed.

/F: < number of files > If present, sets the maximum number of files that can be opened simultaneously during the execution of a GW-BASIC program. Each file requires 62 bytes for the File Control Block (FCB) plus 128 bytes for the data buffer. You can alter the data buffer size via the /S: option switch. If you omit the /F: option, the number of files defaults to 3.

/S: < lrecl > If present, sets the maximum record size allowed for use with random files. **Note:** The record size option to the OPEN statement cannot exceed this value. If you omit the /S: option, the record size defaults to 128 bytes.

/M: < highest memory location > When present, sets the highest memory location that will be used by GW-BASIC. GW-BASIC attempts to allocate 65K of memory for the data and stack segments. If you are using machine language subroutines with GW-BASIC programs, use the /M: switch to reserve enough memory for them.

Note: < number of files >, < lrecl >, < buffer size >, and < highest memory location > can be decimal, octal (preceded by &O), or hexadecimal (preceded by &H).

EXAMPLES:

GW-BASIC PAYROLL

Use all of memory and 3 files, load and execute PAYROLL.BAS.

GW-BASIC INVENT/F:6

Use all of memory and 6 files, load and execute INVENT.BAS.

GW-BASIC /M:32768

Use only the first 32K of memory.

GW-BASIC /F:4/S:512

Use 4 files and allow a maximum record length of 512 bytes.

PRINTER INSTALLATION 6.2

The GW-BASIC Interpreter is capable of outputting graphics or text to a dot-matrix printer via the LCOPY verb. To configure GW-BASIC to drive your system printer correctly, you must use the GWCONF program. GW-BASIC cannot perform the LCOPY screen dump without one of the following printers on your computer system:

EPSON FX
EPSON MX
TALLY
C. ITOH
C. ITOH S
Okidata

Before running the GWCONF program, back up your GW-BASIC issue disk, and work from a copy. To execute the program, type:

GW-BASIC GWCONF

GW-BASIC displays its sign-on banner and then the GWCONF screen. Follow the directions on the screen either to choose a printer or abort the program. If you select the ABORT option, your GW-BASIC is not modified.

If you select one of the printers, a highlighted counter digit displays in the upper left corner of the screen. When it reaches zero, the screen clears and the program returns to MS-DOS; your GW-BASIC is modified. The next time you load GW-BASIC the new printer selection is listed in the sign-on banner.

A

ERROR MESSAGES

GW-BASIC provides the following error messages. These messages supplement the MS-BASIC error messages.

24 Device Timeout

GW-BASIC did not receive information from an I/O device within a predetermined amount of time.

25 Device I/O Error

Fault status returns from the parallel and serial devices. Usually indicates a hardware error in the printer or serial communications channel.

57 Device I/O Error

Formerly "Disk I/O Error". Generalized to include all I/O devices.

68 Device Unavailable

You attempted to open a file to a non-existent device. Hardware might not exist to support the device (for example, LPT2: or LPT3:) or might be disabled by the user.

69 Communication Buffer Overflow

GW-BASIC executed a communication input statement and the input queue was already full. Use an ON ERROR GOTO statement to retry the input. Subsequent inputs attempt to clear this fault unless characters are received faster than the program can process them. In this case these options are available:

1. Implement a "hand-shaking" protocol with the host/satellite such as XON/XOFF (as demonstrated in the TTY programming example), to turn transmit off long enough to catch up.
2. Use a lower baud rate for transmit and receive.

70 Disk Write Protect

This is one of three "hard" disk errors returned from the diskette controller. It occurs when you attempt to write to a write-protected diskette. Use an ON ERROR GOTO statement to detect this situation and request user action.

Other possible "hard" disk errors are:

71 Disk Not Ready

The diskette drive door is open or a diskette is not in the drive. Again use an ON ERROR GOTO statement to recover.

72 Disk Media Error

The FDC controller detected a hardware or media fault. This usually indicates harmed media. Copy any existing files to a new diskette and reformat the damaged diskette. FORMAT flag.

B

BASIC STATEMENTS AND FUNCTIONS

The following is a complete list of statements and functions available in GW-BASIC.

If a statement or function is documented in both the GW-BASIC and the MS-BASIC manuals, refer first to the GW-BASIC manual to find out what new features have been added.

NAME	STATEMENT TYPE	WHERE DOCUMENTED	
ABS	function		MS-BASIC
ASC	function		MS-BASIC
ATN	function		MS-BASIC
AUTO	statement		MS-BASIC
BEEP	statement	GW-BASIC	
BLOAD	statement	GW-BASIC	
BSAVE	statement	GW-BASIC	
CALL	statement	GW-BASIC	MS-BASIC
CDBL	function		MS-BASIC
CHAIN	statement	GW-BASIC	MS-BASIC
CHR$	function		MS-BASIC
CINT	function		MS-BASIC
CIRCLE	statement	GW-BASIC	
CLEAR	statement		MS-BASIC
CLOSE	statement		MS-BASIC
CLS	statement	GW-BASIC	
COLOR	statement	GW-BASIC	
COMMON	statement		MS-BASIC
COM	statement	GW-BASIC	
CONT	statement		MS-BASIC
COS	function		MS-BASIC
CSNG	function		MS-BASIC
CSRLIN	variable	GW-BASIC	
CVI, CVS, CVD	function		MS-BASIC
DATA	statement		MS-BASIC
DATE$	function		MS-BASIC
DEF FN	statement		MS-BASIC

NAME	STATEMENT TYPE	WHERE DOCUMENTED	
DEF USR	statement		MS-BASIC
DEF SEG	statement	GW-BASIC	MS-BASIC
DEFINT, DEFSNG	statement		MS-BASIC
DEFDBL, DEFSTR	statement		MS-BASIC
DELETE	statement		MS-BASIC
DIM	statement		MS-BASIC
DRAW	statement	GW-BASIC	
EDIT	statement	GW-BASIC	MS-BASIC
END	statement		MS-BASIC
EOF	function		MS-BASIC
ERASE	statement		MS-BASIC
ERR and ERL	variable		MS-BASIC
ERROR	statement		MS-BASIC
EXP	function		MS-BASIC
FIELD	statement		MS-BASIC
FIX	function		MS-BASIC
FOR...NEXT	statement		MS-BASIC
FRE	function		MS-BASIC
GET and PUT	statement	GW-BASIC	MS-BASIC
GOSUB...RETURN	statement		MS-BASIC
GOTO	statement		MS-BASIC
HEX$	function		MS-BASIC
IF	statement		MS-BASIC
INKEY$	function		MS-BASIC
INP	function	GW-BASIC	MS-BASIC
INPUT	statement		MS-BASIC
INPUT#	statement		MS-BASIC
INPUT$	function	GW-BASIC	
INSTR	function		MS-BASIC
INT	function		MS-BASIC
KEY	statement	GW-BASIC	
KEY(n)	statement	GW-BASIC	
KILL	statement		MS-BASIC
LCOPY	statement	GW-BASIC	
LEFT$	function		MS-BASIC
LEN	function		MS-BASIC
LET	statement		MS-BASIC
LINE	statement	GW-BASIC	
LINE INPUT	statement		MS-BASIC
LINE INPUT$	statement		MS-BASIC
LIST	statement	GW-BASIC	MS-BASIC
LLIST	statement		MS-BASIC

NAME	STATEMENT TYPE	WHERE DOCUMENTED	
LOAD	statement	GW-BASIC	MS-BASIC
LOC	function		MS-BASIC
LOCATE	statement	GW-BASIC	
LOF	function	GW-BASIC	
LOG	function		MS-BASIC
LPOS	function		MS-BASIC
LPRINT	statement		MS-BASIC
LSET and RSET	statement		MS-BASIC
MERGE	statement	GW-BASIC	MS-BASIC
MID$	statement		MS-BASIC
MID$	function		MS-BASIC
MKI$, MKS$, MKD$	function		MS-BASIC
NAME	statement		MS-BASIC
NEW	statement		MS-BASIC
NULL	statement		MS-BASIC
OCT$	function		MS-BASIC
ON	statement		MS-BASIC
ON COM	statement	GW-BASIC	
ON ERROR	statement		MS-BASIC
ON KEY	statement	GW-BASIC	
OPEN	statement	GW-BASIC	MS-BASIC
OPTION BASE	statement		MS-BASIC
OUT	statement	GW-BASIC	MS-BASIC
PAINT	statement	GW-BASIC	
PEEK	function		MS-BASIC
PLAY	statement	GW-BASIC	
POINT	function	GW-BASIC	
POKE	statement		MS-BASIC
POS	function		MS-BASIC
PSET	statement	GW-BASIC	
PRESET	statement	GW-BASIC	
PRINT	statement		MS-BASIC
PRINT USING	statement		MS-BASIC
PRINT#	statement		MS-BASIC
PUT	statement	GW-BASIC	MS-BASIC
RANDOMIZE	statement		MS-BASIC
READ	statement		MS-BASIC
REM	statement		MS-BASIC
RENUM	statement		MS-BASIC
RESTORE	statement		MS-BASIC
RESUME	statement		MS-BASIC
RIGHT$	function		MS-BASIC
RND	function		MS-BASIC
RUN	statement		MS-BASIC

NAME	STATEMENT TYPE	WHERE DOCUMENTED	
SAVE	statement	GW-BASIC	MS-BASIC
SCREEN	statement	GW-BASIC	
SCREEN	function	GW-BASIC	
SGN	function		MS-BASIC
SIN	function		MS-BASIC
SOUND	statement	GW-BASIC	
SPACE$	function		MS-BASIC
SPC	function		MS-BASIC
SQR	function		MS-BASIC
STOP	statement		MS-BASIC
STR$	function		MS-BASIC
STRING$	function		MS-BASIC
SWAP	statement		MS-BASIC
TAB	function		MS-BASIC
TAN	function		MS-BASIC
TIME$	function		MS-BASIC
TRON and TROFF	statement		MS-BASIC
USR	function		MS-BASIC
VAL	function		MS-BASIC
VARPTR	function	GW-BASIC	MS-BASIC
WAIT	statement	GW-BASIC	MS-BASIC
WHILE...WEND	statement		MS-BASIC
WIDTH	statement	GW-BASIC	MS-BASIC
WRITE	statement		MS-BASIC
WRITE#	statement		MS-BASIC

INDEX